Комп'ютерна математика для фізиків

Костянтин В. Усенко

27 червня 2002 р.

Зміст

0.1 Передмова

Побудова комп'ютерної моделі кожного фізичного явища завжди починається з переліку конкретних фізичних величин, що саме ними характеризується досліджуване явище, та законів, що їм мають підкорятись ці величини. Закони формулюються, як рівняння для величин – характеристик явища і головною частиною задачі комп'ютерного моделювання є побудова розв'язків для цих рівнянь – суто математична задача. Власне кажучи, фізичною частиною в постановці задачі про побудову комп'ютерної моделі є саме визначення переліку необхідних величин, що характеризують явище, та законів – рівнянь для цих величин. Наступний етап – розв'язування рівнянь, є математичною задачею, але математичні методи, що необхідні для моделювання фізичних явищ, певною мірою відрізняється від таких для моделювання явищ економіки або лінгвістики, а навіть і біології, тому математика для фізиків давно вже відрізняється від математики для економістів, математики для біологів і навіть математики взагалі.

Комп'ютерна математика суттєво відрізняється від математики доком'ютерної хоча б з уваги на обмеженість можливих значень „натуральних" чисел в комп'ютерному зображенні, тому з появою обчислювальної техніки вона активно розвивається.

Поява обчислювальної техніки випала саме на ті часи, коли найважливішим її застосуванням було моделювання процесів, що відбуваються в ядерних реакторах та процесів вибуху і горіння, і першими областями комп'ютерної математики були саме ті, що вкрай необхідні для моделювання фізичних процесів. Звичайно ж, стандартні методи розв'язування задач фізики вимагали певної модифікації з урахуванням особливостей розрахунків за допомогою комп'ютера, але на початковому етапі, поки комп'ютери використовувались тільки для окремих надскладних задач, інтуїції спеціалістів вищого ґатунку вистачало, аби розібратись з цими особливостями без спеціальної теоретичної підготовки в галузі комп'ютерної математики. В ті часи обчислювальна математика, як така практично не відрізнялась від обчислювальної математики для фізиків.

Подальший розвиток обчислювальної техніки забезпечив її активне використання в інших, не фізичних галузях людської діяльності і останнім часом комп'ютерна математика головним чином переймається проблемами обробки інформації не числової природи. В сучасних підручниках з комп'ютерної математики значну частину посідають питання граматичного аналізу, проблеми кодування, проблеми пошуку інформації в великому масиві даних, а от проблеми пошуку розв'язків системи рівнянь залишаються поза увагою. Між тим саме розв'язування системи рівнянь є головним, чого потребує фізика від комп'ютера, як інструмента фізичних досліджень.

Під комп'ютерною математикою для фізиків мається на увазі область математики, що без неї не можна будувати комп'ютерні моделі фізичних явищ.

Останнім часом під комп'ютерною математикою дедалі частіше розуміється одна тільки її галузь - модифікована з урахуванням вимог сучасного

програмування і трохи поповнена відповідними прикладами галузь математики, відома також під назвою дискретної математики. Методи теорії кодування, алгоритми пошуку, впорядкування великих масивів – це все важливо і в фізичних дослідженнях, але не є обов'язковим для розуміння методів побудови комп'ютерних моделей фізичних явищ.

Взагалі ж орієнтація на використання комп'ютерів у великих обчисленнях властива всім галузям сучасного знання, тому комп'ютерної форми набувають практично всі галузі математики, а в першу чергу – та математика, що її використовують фізики, а за ними і всі, хто спирається в своїх дослідженнях на закони фізики та їх наслідки. Відтак до користувачів фізичного варіанту комп'ютерної математики потрапляють і спеціалісти з матеріалознавства, і спеціалісти з хімії, в першу чергу квантової, і спеціалісти з метеорології і так далі і так далі.

Досить часто ця гілка комп'ютерної математики кваліфікується, як математика обчислювальна, адже головною, визначальною рисою кожної комп'ютерної моделі фізичного явища є велика кількість обчислень, часто завелика, а інколи і просто занадто велика. Але обчислювальна математика, як така, турбується тільки алгоритмами, але не переймається питанням про практичне здійснення алгоритмів. Між тим від алгоритму розв'язування певної задачі, як такого і до кінцевої відповіді цієї задачі досить часто потрібно розв'язати певну кількість проблем, пов'язаних з комп'ютерною реалізацією алгоритму. Саме поєднання обчислювальної математики з принаймні елементарними уявленнями про теорію алгоритмів та практику програмування і є предметом комп'ютерної математики з точки зору фізики.

Першою і найважливішою відміною комп'ютерної реалізації алгоритму від його теоретичного прототипу є обмеження на точність зображення чисел в комп'ютері. Досить поширеною є думка, що хай воно й обмежене, але ж в разі потреби можна взяти подвійну точність або ще більшу... Хибність цієї думки полягає не в тому, що не можна взяти подвійну точність або ще більшу, а тільки в тому, що залишається невизначеним, коли ж саме потрібна вища точність, адже досить давно вже в програмах фізичних обчислень найчастіше терміни Real, Float або їм подібні замінено на Double. Насправді реальні обчислення із дійсними числами звичайно виконуються зараз в окремому каналі процесора, а цей канал навіть і не має гадки про існування чисел так званої звичайної точності – він для всіх обчислень використовує однакову точність типу Double. Згадка про точність залишається поки-що, як вказівка про довжину області пам'яті, що виділяється для збереження одного числа – 4, 6 або 8, а навіть і 10 байтів. З переходом на 64 – розрядну архітектуру і ця різниця зникає.

Насправді суттєвим є саме існування обмежень в зображенні дійсних чисел. Найважливішим наслідком цих обмежень є не просто сам факт існування чисел, що їх комп'ютерні зображення не відрізняються, а та обставина, що таких чисел аж надто багато. Дійсно, припустимо, що комп'ютер, що використовується для деякої задачі, має відповідно до стандарту IEEE 754 Double-Precision 53 біти для зображення мантиси m дійсного числа відповідно до форми зображення $1.m \cdot 2^p$, тоді наступним після одиниці буде

число $1 + 2^{-53}$ і різницю, начебто, можна вважати знехтовно малою. Але насправді між цими двома числами ще стільки ж чисел, скільки на всій числовій вісі!

Дійсно, поставимо кожному числу x з числової вісі у відповідність число y з цього проміжку за правилом $y = 1 + 2^{-54}(\tanh(x) + 1)$. Ця відповідність є взаємно-однозначною, тобто кожному числу y з околу одиниці відповідає точно одне число з дійсної вісі.

Наслідком такого обмеження є принципова неможливість реалізації в комп'ютерних обчисленнях таких звичайних для сучасної математики дій, як запис функції (якщо тільки мова не йде про системи аналітичних обчислень, але це вже не розрахункові системи, а системи синтаксичного аналізу, що оперують не з числами, а із словами). За означенням функція є відображенням множини означення на множину значень, але в фізичних дослідженнях про множину означення найчастіше припускають, що вона є множиною дійсних чисел, або областю в цій множині, але аж ніяк не набором окремих точок. В комп'ютерному зображенні функція означається табличкою значень: кожному можливому значенню аргументу відповідає рядок цієї таблички. Інколи про функцію можна вигадувати навіть, що вона є правилом розрахунку значень для кожного значення аргументу, але і в цьому разі всі можливі значення аргументу можуть належати не до множини дійсних чисел, а тільки до множини Double або Float, тобто до множини комп'ютерних чисел.

Основним інструментом комп'ютерного аналізу, як для фізика, замість звичайної функції дійсної змінної є функція дискретної змінної і ця обставина принципово відрізняє результати комп'ютерних обчислень від аналітичних.

Комп'ютерне обчислення завжди має результатом не одне число, а цілий інтервал чисел. Якщо обчислення здійснюється в одну дію, результатом його буде інтервал відносної довжини 2^{-53}, але якщо дій дві, то в найгіршому випадку інтервал подвоюється і це ще нормально, а от в найкращому випадку він просто лишається незмінним, і саме це вже погано. Якби комп'ютерні похибки могли не тільки зростати, а й зменшуватись, можна було б чекати, що в середньому похибки великої кількості операцій компенсуватимуться. Насправді ж вони можуть тільки збільшуватись, і по виконанні деякої кількості дій величина похибки має стати порівняною з величиною результату обчислень.

Комп'ютерні обчислення з надто великою кількістю дій є хибними.

Звичайно ж, цьому твердженню не вистачає принаймні однієї дрібнички – визначення того, яку саме кількість дій слід вважати надто великою. На перший погляд, ця кількість може здаватись просто недосяжною. Дійсно, якщо вважати, що за кожну дію втрачається один біт точності, то запасу 2^{-53} вистачило б на 2^{53} дій. Для сучасного домашнього комп'ютера з швидкодією в $10^9 = 2^{30}$ операцій в секунду це буде $2^{23} = 10^8$ секунд, або біля 10 років роботи. Комп'ютери стільки не живуть! Але сучасні суперкомп'ютери мають швидкодію на 5 порядків вищу, а це вже мова йде про тисячі секунд – час, набагато менший за потрібний для обчислення метеорологічних за-

дач. Насправді досить часто помилка стрибком зростає, коли віднімаються два числа, що мало відрізняються одне від одного, а в такому разі і на персональному комп'ютері можна отримати за розумний час зовсім нерозумні результати.

Комп'ютерні обчислення вимагають уваги до похибки обчислень.

Насправді завжди або майже завжди можна побудувати алгоритм, що буде принаймні частково зменшувати похибку, потрібно тільки мати на увазі, що обчислювальний алгоритм у викладенні для комп'ютера може відрізнятись від такого, що розрахований на обчислення вручну.

Наступна особливість комп'ютерної математики – проблема скінченності.

Звичайний аналіз ще з давньогрецьких часів знає, як здійснювати нескінченну кількість дій. Одним з найпростіших прикладів є апорія Зенона про Ахіллеса та черепаху. Ахіллесу на те, аби тільки добігти до черепахи, потрібна нескінченна кількість дій, оскільки, поки він добігатиме до того місця, де вона знаходилась, черепаха відповзе на деяку відстань, поки він пробігатиме цю відстань, вона знов відповзе і так до нескінченності. Єдине, що спасає Ахіллеса, це те, що час, який витрачатиметься на кожен крок, буде зменшуватись за геометричною прогресією, а суму такої прогресії можна обчислити аналітично і вийде, що на цю нескінченну кількість кроків Ахіллес витратить скінченний час. Якщо ж доручити комп'ютеру обраховувати суму геометричної прогресії, він буде робити це, аж поки не вийде з ладу, але так і не дорахується, що загальний час є скінченним. Звичайно ж, можна і комп'ютер навчити не рахувати суму членів, а завчасно обірвати нескінченну прогресію, замінивши її такою скінченною послідовністю, що її сума знехтовно мало відрізняється від суми геометричної прогресії. Саме необхідність такої заміни й відрізняє комп'ютерну математику від обчислювальної.

Навіть якщо алгоритм, що його потрібно здійснити, аби отримати потрібний результат, є скінченним, він може вимагати надто великої кількості дій. Ця кількість дій, само собою зрозуміло, залежить ще й від кількості даних, що ними оперує задача і що їх потрібно отримати. В найпростішому випадку, коли на кожне значення, що з них складається відповідь, вистачає лише однієї дії, кількість дій буде дорівнювати кількості потрібних значень. Якщо такими значеннями є розрахунки температури по одному значенню на кожен квадратний кілометр земної поверхні, то вийде 10^8 значень – досить велика величина. Зрозуміло, що перед постановкою такої задачі доцільно хоч приблизно оцінити, чи може комп'ютер за розумний час отримати відповідь, чи краще трохи зменшити вимоги і розраховувати температуру по одному значенню не на кожен квадратний кілометр, а на кожні сто. Така оцінка передбачує, що з якихось міркувань відомо, як саме залежить кількість дій від кількості результуючих значень. Найчастіше в цих оцінках оперують покажчиком степеню, до котрого потрібно підвести кількість даних, аби отримати потрібну кількість дій і кажуть про задачі лінійної складності (найкращий випадок), квадратичної, кубічної, тощо. Теорія алгоритмів доводить, що найгіршою оцінкою є факторіальна

складність. Саме такою є складність задачі обчислення розв'язку системи лінійних рівнянь за методом Крамера (методом визначників). Зрозуміло, що практично застосовні алгоритми розв'язування систем лінійних рівнянь повинні відрізнятись від методу визначників. Дійсно, існують типові алгоритми кубічної складності, що їх можна застосовувати до довільних систем, а для деяких спеціальних систем існують і більш ефективні алгоритми, навіть такі, що мають лінійну складність.

Проблеми, що їх має вирішувати комп'ютерна математика для фізиків, не можуть вирішуватись без використання якої-небудь мови програмування. На крайній випадок, такою мовою може бути просто мова машинних кодів і на початку комп'ютерної ери саме так програмування і здійснювалось. Навіть після появи допоміжних мов, таких, як Fortran, довгий час програмування здійснювалось з огляду на те, як саме програма буде виглядати в машинних кодах. Як на сьогоднішній стан розвитку комп'ютерної техніки, уявлення про машинні коди втратило сенс повністю, адже сучасні процесори не виконують тих команд, що містяться в програмі. Замість цього спеціалізована частина процесора перетворює кожен код в послідовність мікродій і саме ці мікродії виконуються окремими блоками процесора, інколи одночасно декілька в одному блоку, інколи одна в декількох блоках. Уявлення ж про деталі мікрокоду заховані в конструкції процесора і відомі тільки його розробникам, тому намагання аналізувати програму на якість з точки зору послідовності команд процесора приречені на невдачу. Сучасні транслятори розробляються найчастіше з урахуванням специфіки конкретних процесорів і в режимі з оптимізацією коду забезпечують майже таку ж ефективність програми, якою вона може бути в програмуванні в машинних кодах на рівні мікродій.

Починаючи з перших повноцінно 32 – розрядних процесорів, все програмування здійснюється виключно мовами високого рівня. Як на зараз, їх нараховується декілька десятків тільки серед активно використовуваних. На щастя, в обчислювальних задачах активно використовуються лише декілька. Найпопулярнішою є мова C++ в трьох реалізаціях – Microsoft, Borland та GCC (GNU Common Compilator). Перші дві реалізації розраховані на персональні комп'ютери під операційними системами Windows, тоді як остання, що розробляється об'єднанням програмістів для вільного розповсюдження, має версії практично для всіх операційних систем та апаратних платформ, в тому разі і надпотужних сучасних кластерів. Транслятор GCC є головним в вільно розповсюджуваних системах FreeBSD, Linux, в більшості комерційних реалізацій Unix, за винятком хіба що Sun, яка використовує в ядрі операційної системи Java – машину. Але і для цієї системи існує і досить ефективно працює відповідна версія транслятора GCC. Метою такої широкої адаптації GCC було і є забезпечення сумісності програмного забезпечення, точніше, апаратна незалежність програмного забезпечення, що створюється з розрахунку на GCC. Завдяки цьому програми, що транслюються GCC в операційній системі Linux, можна після ретрансляції використовувати і в Windows і на потужних суперкомп'ютерах. Звичайно ж, платформну незалежність виходить забезпечувати тільки для тих програм,

що не використовують спеціалізованих бібліотек графічного відтворення результатів. Саме з цієї причини в наукових програмах розрахункова частина та частина візуалізації найчастіше виділяються в окремі програми.

На час підготовки цієї книги є два принципово різних шляхи використання транслятора GCC в оточенні Windows. Перший спирається на пакет Cygwin, що забезпечує функціонування під Windows повного середовища Unix (він є аналогом командного рядка DOS), другий є відокремленою імплементацією трансляторів з мов C++ (Dev-C++) та Pascal (Dev- Pascal), розробленою в межах проекту програмного забезпечення з відкритим кодом корпорацією Bloodshed. Наведені далі приклади програм мовами C++ та Pascal перевірялися саме цими трансляторами. Їх інсталяційні пакети можна знайти на сторінці курсу програмування та математичного моделювання фізичного факультету Київського університету Тараса Шевченка.

Розділ 1

Алгебраїчні задачі.

Комп'ютерні обчислювальні методи найчастіше застосовуються до двох класів задач алгебри – до пошуку коренів складних рівнянь, тобто таких, для яких не існує прямих виразів для обчислення коренів, та до великих за розміром систем лінійних рівнянь. Ці два класи задач відрізняються не тільки застосовуваними методами, а й набором потрібних уявлень та термінів. Задачі першого класу, на відміну від задач лінійної алгебри, звичайно називають задачами на розв'язування нелінійних рівнянь. Окремою проблемою інколи виділяють задачі визначення коренів великих систем нелінійних рівнянь, хоча частіше такі задачі виділяють в окремий розділ – задачі оптимізації.

Проблеми, що суттєві для задач на пошук коренів, можна роздивитись вже на такому простому прикладі, як пошук коренів рівняння $x^2 = 2$. Звичайно вважається, що про корені цього рівняння ми знаємо практично все вже після середніх класів звичайної школи. Записати їх можна досить просто, поклавши $x_1 = \sqrt{2}$ та $x_1 = -\sqrt{2}$, але чи буде так само просто використати ці значення?

Запис $x_1 = \sqrt{2}$ насправді означає тільки, що вираховуючи різні вирази з цим числом, ми завжди можемо використати таку його властивість: $(x_1)^2 = 2$. Але чисел з такою властивістю є два, оскільки $(x_2)^2$ теж дорівнює двом і відрізнити, яке саме з цих двох чисел ми використовуємо в конкретному випадку, можна тільки за допомогою додаткової інформації про знак числа.

Використати значення кореня в таких виразах, як $\sqrt{2} - \frac{7}{5}$, вже трохи складніше, оскільки визначити знак цього виразу можна тільки після додаткових міркувань. І взагалі, цей вираз зовсім не нагадує число в звичайному його розумінні! Його треба ще обчислювати, підставляючи конкретне подання кореня з двох. Використовуючи різні наближені значення для $\sqrt{2}$, ми отримаємо, звичайно ж, різні числа. Так, наближення $\sqrt{2} \approx 1.4$ дасть нам $\sqrt{2} - \frac{7}{5} \approx 0$, хоча вже врахування наступної цифри в десятковому дробу для кореня дасть $\sqrt{2} - \frac{7}{5} \approx 0.01$.

Розглянутий приклад свідчить, що обчислення з дійсними числами, особливо з тими, що не є, зовсім випадково, раціональними, завжди дає тільки

наближені результати. Для кожного з таких чисел, хочемо ми того чи ні, ми завжди використовуємо тільки наближене (раціональне) значення, визначаючи, для того ж прикладу, що значення кореня рівняння $x^2 = 2$, яке ми позначаємо через $\sqrt{2}$, знаходиться в межах $\frac{7}{5} < \sqrt{2} < \frac{8}{5}$. Відповідно, якщо такі межі вже відомі, ми можемо уточнювати їх, звужуючи проміжок, в якому знаходиться корінь, і це уточнення має подовжуватись доти, доки неточність, обумовлена заміною кореня якою-небудь точкою проміжку, не стане несуттєвою для конкретної розв'язуваної задачі.

Саме в останньому твердженні і сконцентрований сенс задачі обчислення кореня. Конкретні задачі ніколи не потребують знання точного значення кореня рівняння, а врешті-решт, і точного значення довільного дійсного числа. На практиці для дійсних чисел використовуються або загальні властивості, подібні до (точного) виразу $x^2 = 2$, або достатньо точні наближення раціональними числами. Звичайно ж, виникає питання про можливість побудови потрібного наближення. Відповідь на це питання дає відома теорема з теорії чисел про щільність множини раціональних чисел. Вона стверджує, що множина раціональних чисел є всюди щільною, тобто що для кожного дійсного числа dзнайдеться хоча б одне раціональне число r, що відрізняється від цього дійсного числа не більше ніж на задану похибку ε; $|d - r| \leqslant \varepsilon$.

Отже, задача знаходження коренів алгебраїчного рівняння є проблемою тільки доти, доки не встановлені межі для кожного з коренів і полягає саме у встановленні меж, що відділяють один корінь від іншого, сусіднього, або ж у виділенні проміжків, в яких знаходиться точно один корінь. Насправді ця задача поділяється на задачу визначення областей, в яких знаходиться кожен з коренів – областей відокремленості кореня та задачу звуження області відокремленості, тобто задачу про уточнення кореня.

Задачі лінійної алгебри ставлять, на перший погляд, менше проблем, але насправді це просто проблеми інші за змістом. Систему неоднорідних лінійних рівнянь розв'язати зовсім просто, потрібно просто здійснити відповідну кількість дій. Але саме в кількості дій і полягає проблема Досить часто задача вміщує багато змінних і для вирахування кожної з них потрібно щонайменше декілька дій. Якщо система рівнянь розв'язується тільки один раз, то ці дії можна відтворити майже в будь-який спосіб, але й тоді кількість дій щонайменше пропорційна кількості змінних. Але найчастіше кількість дій пропорційна більш високому степеню кількості змінних. Якщо ж система лінійних рівнянь розв'язується, як частина більш складної задачі, як часто буває при моделюванні процесів в суцільному середовищі, кількість дій, що потрібні для розв'язування системи, є головною для оцінки обчислювальної складності задачі.

Ще однією проблемою задач лінійної алгебри є проблема похибки отриманого розв'язку. Дії з дійсними числами, якщо тільки ці числа не є цілком випадково степенями двійки, завжди супроводжуються похибкою округлення. Якщо дій багато, похибка накопичується і тому найтривіальніший метод розв'язування лінійних рівнянь – метод Крамерса, що потребує $(n + 1)!$ дій, вже для системи з 15 невідомими призведе до похибки, що в $16^{16} = 2^{64}$ разів перевищує похибку одного обчислення. Навіть якщо використовується

арифметика подвійної точності з відносною похибкою 2^{-56}, за рахунок накопичення похибки результати обчислень втрачають сенс. Разом з тим метод Гауса, що потребує тільки n^3 дій, збільшує похибку тільки в $16^3 = 2^{12}$разів і загальна відносна похибка $2^{-44} \sim 10^{-12}$ залишається в межах розумного.

Накопичення похибки може й не відбуватись, якщо алгоритму властива компенсація похибок. Тому для дуже великих систем лінійних рівнянь потрібно використовувати спеціальні алгоритми з компенсацією.

1.1 Нелінійні алгебраїчні рівняння

Задача знаходження кореня звичайного алгебраїчного рівняння $f(x) = 0$ може бути розв'язаною аналітично тільки для вузького класу задач (поліноми не вище четвертого степеню). Певні класи задач (тригонометричні, логарифмічні рівняння тощо) зустрічаються настільки часто, що для них досліджено та табульовано всі розв'язки, тобто побудовано обернену функцію $f^{-1}(x)$ - таку функцію, що для кожного значення з області означення прямої функції справджується співвідношення $f^{-1}(f(x)) \equiv x$. Тим не менше для довільної функції задача знаходження коренів залишається нерозв'язаною. Більше за це, для довільної функції не існує алгоритму побудови точного значення кореня. Частково відсутність алгоритму пов'язана з тим, що корені найчастіше належать до дійсних, але не натуральних чи хоча б раціональних чисел. Частково ж відсутність загального алгоритму знаходження коренів довільного рівняння обумовлена тим, що існує занадто багато різних функцій, що потребують різних методів знаходження коренів. Відповідно задача побудови алгоритму повинна ставитись окремо для кожного з таких методів і до того ж в ній потрібно поєднувати задачу знаходження області відокремленості для кожного з коренів та задачу уточнення значення кожного конкретного кореня.

Фізика досить часто потребує обчислення коренів функції, що є розв'язком диференціального рівняння (найчастіше другого порядку). В таких задачах стають на допомозі різноманітні форми теорем про перемежовування коренів – про те, що корені одного з розв'язків та корені іншого перемежовуються між собою. В інших задачах знаходженню областей відокремленості допомагає дослідження графіка функції. Обидва ці приклади демонструють штучність використовуваних методів. Активне використання штучних методів, в свою чергу, є свідоцтвом відсутності загальнопридатного методу відокремлення коренів.

Далі будуть розглянуті чотири типових методи, що часто використовуються при розв'язуванні нелінійних рівнянь з одним невідомим.

Два з них використовують явну інформацію про межі області відокремленості кореня і тому їх можна вважати такими, що тільки уточнюють значення кореня. Це – метод ділення навпіл та метод перетинів.

Третій метод, метод Ньютона не потребує попереднього визначення області відокремленості, але використовує явно не тільки саму функцію, а й її похідну – це є своєрідною платнею за відмову від попереднього визначення області відокремленості.

Четвертий метод, метод ітерацій, добре працює тільки в тому разі, якщо відоме наближене значення кореня.

1.1.1 Метод ділення навпіл

Цей метод є найпростішим з можливих методів обчислення коренів. Він спирається на припущення про те, що інтервал відокремленості кореня вже визначено і дозволяє просто уточнювати значення кореня.

Метод ділення навпіл, як витікає з його назви, полягає в звуженні вдвічі інтервалу відокремленості. З цією метою знак лівої частини рівняння в середині інтервалу відокремленості порівнюється зі знаками тієї ж функції на кінцях інтервалу і за новий інтервал відокремленості визнається та половина початкового інтервалу, в котрій функція змінює знак.

Оцінкою якості методу є кількість дій, що потрібні для звуження вихідного інтервалу до заданої величини. Після n повторень інтервал зменшується в 2^n разів, тож задану точність (ширину інтервалу) ε можна отримати за $\log_2\left(\frac{\Delta}{\varepsilon}\right)$ кроків. Тут Δ- початкова ширина інтервалу відокремленості. Звичайною оцінкою може бути зменшення до 10^{-6} інтервалу відокремленості одиничної довжини, що досягається за 20 кроків.

Отже, припустимо, що для заданої функції $f(x)$ відомо, що інтервал відокремленості обмежений точками a та b і функція на кінцях інтервалу має значення $f(a) > 0$ та $f(b) < 0$. Розглянемо значення функції в середині відрізку – в точці $c = \frac{1}{2}(a+b)$. Якщо значення в цій точці додатне $f(c) > 0$, ми визнаємо за новий інтервал відокремленості відрізок $[c, b]$, якщо від'ємне - $[a, c]$. На цьому крок обчислень закінчується і можна переходити до оцінки отриманої точності.

Особливим випадком є така несподівана подія, як обертання значення функції на нуль. Взагалі кажучи, такого начебто не може бути, оскільки всі дійсні числа, що можуть бути зображені в комп'ютері, є раціональними, а раціональні корені звичайно можна знайти аналітично. Насправді, досить часто простіше знайти корінь обчислювальним методом, за його значенням здогадатись, що він знаходиться аналітично і вже тоді прямою підстановкою перевірити здогадку. Думка про те, що аналітичне розв'язування рівняння простіше за обчислювальне, як така народилась в ті часи, коли обчислювальні методи здійснювались не в комп'ютері, а на папері. Якщо задача полягає просто в обчисленні кореня, обернення рівняння на нуль вже дає розв'язок, але якщо метою є ще й визначення точності, потрібне додаткове дослідження значень в околі знайденого кореня. Тому стандартної реакції програми на обертання значення функції на нуль побудувати не можна - програма повинна просто повідомляти про виняткову ситуацію.

Побудуємо тепер алгоритм методу.

Припустимо, що значення функції розраховуються окремою підпрограмою, виклик якої для значення аргументу x будемо позначати в стандартний спосіб $f(x)$.

Для роботи алгоритму потрібні, крім самої функції, також межі інтервалу відокремленості x_a та x_b, а також очікувана похибка кореня ε.

В процесі роботи програми будуть змінюватись значення лівої та правої меж інтервалу відокремленості x_{left}, x_{right} та центру цього інтервалу x_{centre}. Ці три величини є внутрішніми змінними програми.

Підготовча частина програми полягає в ініціалізації внутрішніх змінних

$$x_{left} \Leftarrow x_a, \quad x_{right} \Leftarrow x_b \tag{1.1}$$

Перший крок програми полягає в обчисленні положення центру інтер-

валу відокремленості

$$x_{centre} \Leftarrow \frac{1}{2}\left(x_{left} + x_{right}\right), \tag{1.2}$$

а наступний – в порівнянні знаків функції в центрі та на кінцях інтервалу.

Таке порівняння потребує обчислення значень функції в трьох точках, тобто на кожному кроці обчислення функції повинно було б здійснюватись тричі. Але знаки функції на кінцях інтервалу відокремленості є протилежними, тож досить порівнювати значення в центрі тільки з одним із таких значень (наприклад, лівим). Крім того, знак функції на кінці інтервалу відокремленості не змінюється при звуженні інтервалу (якщо тільки не буде зроблена помилка), тому він залишається таким самим, як і на початок роботи алгоритму і його можна не обчислювати на кожному кроці, якщо додати ще одну внутрішню змінну, наприклад так

$$s_{left} = sign\left(f\left(x_{left}\right)\right). \tag{1.3}$$

Тоді наступний крок полягає в перевірці знаку добутку значення функції в центрі та збереженого знаку функції на краю інтервалу відокремленості

$$if\left(s_{left} \cdot f\left(x_{centre}\right) > 0\right) \tag{1.4}$$

Якщо ця умова виконується, потрібно замінити значення лівого краю інтервалу відокремленості на значення центру і перейти до перевірки ширини нового інтервалу

$$x_{left} \Leftarrow x_{centre}; goto\ ??. \tag{1.5}$$

Якщо ж умова ?? не виконана, потрібно перевірити, чи не буде добуток меншим за нуль

$$if\left(s_{left} \cdot f\left(x_{centre}\right) < 0\right) \tag{1.6}$$

Якщо виконується ця умова, центр обирається за правий край інтервалу відокремленості і знов потрібно перейти до перевірки ширини нового інтервалу

$$x_{right} \Leftarrow x_{centre}; goto\ ?? \tag{1.7}$$

Якщо ж не виконується ні ?? ні ??, значення функції в центрі вже дорівнює нулю і робота завершена нестандартно, про що можна повідомити, виробляючи ознаку помилки спеціального типу (виконавча система мови програмування звичайно надає можливість викликати повідомлення про нестандартну ситуацію і зробити спеціальну частину програми для обробки такого виклику).

$$Error(zero) \tag{1.8}$$

Наступна дія програми полягає в перевірці ширини нового інтервалу відокремленості - порівнянні цієї ширини з очікуваною похибкою обчислень

$$if\left(\left(x_{right} - x_{left}\right) > \varepsilon\right) \tag{1.9}$$

Якщо ця умова виконується, потрібно повернутись на перший крок програми

$$goto\ ??, \tag{1.10}$$

якщо ж ні – роботу програми завершено стандартно, і вона має повернути значення нового центру

$$return\ \tfrac{1}{2}\left(x_{left} + x_{right}\right). \tag{1.11}$$

В процесі обчислень, незважаючи на спеціальні засоби щодо скорочення кількості викликів підпрограми обчислення значень функції, з'явилось подвоєння викликів – на кроці **??** і на кроці **??** підпрограма викликається для обчислення одного і того ж значення. Зменшити кількість викликів можна тільки в тому разі, якщо використати додаткову змінну – значення, що підлягає перевірці в обох цих пунктах.

По завершенні розробки алгоритму наступним етапом є переклад на конкретну мову програмування. На цьому етапі звичайним є уточнення алгоритму з урахуванням спеціальних можливостей використовуваної мови.

Найважливішою ознакою сучасних мов програмування є присутність розвинених засобів побудови різноманітних циклів, що виключає необхідність використання оператора переходу. Під час трансляції в команди процесора спеціалізовані оператори циклу модифікуються з урахуванням можливостей сучасних процесорів. Прикладом такої модифікації може бути підготовка прогнозу в конвеєрних процесорах (Pentium 2 і більш сучасних). Такі процесори починають виконувати наступну команду, не очікуючи завершення попередніх і тому повинні передбачати, який саме варіант продовження програми буде ймовірнішим по закінченні дії умовного оператора. Якщо прогноз виявився хибним, всі наступні дії відкидаються і процесор починає виконувати потрібну послідовність дій. Якщо глибина конвеєра сягає хоча б 5 команд (досить типове значення), наслідком помилкового прогнозу є уповільнення роботи програми в 5 разів. Транслятор звичайно очікує, що цикл буде виконуватись хоча б декілька разів і тому додає до коду прогноз на більшу ймовірність продовження циклу, ніж закінчення. Якщо ж використовується оператор безумовного переходу, транслятор не пропонує процесору жодного прогнозу.

Pascal пропонує три різновиди операторів циклу: цикл з перерахуванням, цикл з передумовою та цикл з післяумовою. Оскільки кількість проходжень циклу зарані не визначена, в розглядуваній програмі цикл з перерахуванням недоцільний. Вибір між циклами з перед – та післяумовою визначається найчастіше з того, коли саме вперше вираховується значення параметру циклу – до виконання тіла циклу чи після. Оскільки обидві межі інтервалу відокремленості кореня відомі ще до початку циклу, доцільним є саме цикл з передумовою, тож головною частиною програми буде такий цикл:

```
while (xright-xleft)>eps do Body;
```

Тіло циклу починається з обчислення координати середньої точки проміжку відокремленості $xcentre := (xright + xleft)/2$ (пункт **??** алгоритму) і знаку значення функції в середній точці $signf := sgn(signl * f(xcentre))$. Власне розрахунок знаку виділений в окрему функцію, оскільки в ній використовуються вкладені один в інший умовні оператори, а така конструкція зовсім не сприяє зручності від лагодження програми. Функція обчислення знаку має такий вигляд:

```
function sgn(x:Double):Integer;
  begin
    if x>0 then sgn:=1
           else if x<0 then sgn=-1
                       else sgn=0
  end;
```

Вона повертає значення +1, якщо її аргумент строго додатній, -1, якщо строго від'ємний і 0, якщо аргумент одночасно і не додатній і не від'ємний. Саме це додаткове значення дозволяє програмі використати такий дарунок долі, як цілком випадкове вгадування значення кореня.

Оскільки подальші дії суттєво відрізняються залежно від значення, що його повертає функція обчислення знаку, программа розгалуджується на три гілки відповідно до можливих варіантів. Таке багатоваріантне розлалудження мовою Pascal здійснюється за допомогою оператора вибору

```
Case sgn(signl*f(xcentre)) of
     1 :xleft:=xcentre;
    -1 :xright:=xcentre;
     else begin
          xleft:=xcentre;xright:=xcentre;
          end
end;
```

Перший з варіантів відповідає співпаданню знаків значень розв'язуваної функції в центрі та на лівому кінці відрізку. Відповідно корінь знаходиться праворуч від центру і за новий відрізок відокремленості кореня обирається відрізок від центру до правої межі.

Другий варіант відповідає протилежним знакам значень розв'язуваної функції в центрі та на лівому кінці відрізку. В такому разі корінь знаходиться ліворуч від центру. Останній варіант означає, що корінь знаходиться точно в центрі і обидві межі області відокремленості кореня також знаходяться в центрі.

На цьому тіло циклу закінчується і програма повертається до перевірки умови необхідності виконання циклу. Ця умова стане хибною, як тільки довжина відрізку відокремленості кореня стане меншою за заздалегідь визначену величину – очікувану похибку обчислення кореня eps.

Перевіримо тепер завершуваність циклу. Після кожного проходження тіла циклу довжина області відокремленості кореня стає вдвічі меншою, тому

після скінченної кількості проходжень вона стане меншою за яку завгодно зарані задану величину і теоретичний цикл має завершуватись завжди.

Звернімо тепер увагу на обмеження, що накладаються дискретністю комп'ютерного зображення дійсних чисел. Оскільки для зображення мантиси дійсного числа використовуються скінченна кількість бітів p, можна відрізнити одне від одного лише числа, що їх відношення є не меншим за $1+2^{-p}$. Якщо ж очікувана похибка менша за можливу різницю двох сусідніх чисел, умову завершення ніколи не можна буде задовольнити і цикл може перетворитись на нескінченний.

Програма в цілому має ще розділ ініціалізації та окремо – оголошення функції, що її корінь потрібно розрахувати.

```
program half;
var
   left,right,centre :Double;
   signc,signl       :Integer;
const
   eps=1e-6; xleft=0;xright=2;

function sgn(x:Double):Integer;
  begin
    if x>0 then sgn:=1
           else if x<0  then sgn:=-1
                        else sgn:=0;
  end;

function f(x:Double):Double;
  begin
    f:=x*exp(x)-1;
  end;

var
   n:Integer;

BEGIN
   n:=0;
   left:=xleft;right:=xright;
   signl:=Sgn(f(left));
   while right-left>eps
   do begin {main cycle}
      n:=n+1;
      centre:=(right+left)/2;
      signc:=Sgn(signl*f(centre));
      Writeln(n,signc);
      case signc of {half selector}
             1  :right:=centre; {to left}
```

```
           -1  :left:=centre; {to right}
         else begin
              right:=centre;left:=centre;
              end {exact value}
       end; {half selector}

    end; {main cycle}

   Writeln('res=',centre);
   ReadLn;
END.
```

Текст програми методу ділення навпіл.

Функція, що її корінь розраховується, означається в підпрограмі – функції f(x), межі області відокремленості – сталі , точність розрахунку задано сталою eps.

1.1.2 Метод перетинів

На відміну від методу ділення навпіл метод перетинів повніше використовує інформацію про значення функції. Від попереднього методу цей відрізняється тим, що замість центра відрізку відокремленості кореня використовує точку перетину з віссю абсцис прямої, що проходить розрахованими точками графіку функції. Ця точка розраховується за формулою

$$x_{middle} = \frac{x_{left} \cdot y_{right} + x_{right} \cdot y_{left}}{y_{right} - y_{left}} \tag{1.12}$$

і знаходиться ближче до того кінця відрізку відокремленості кореня, біля якого значення функції є меншим за абсолютною величиною. Замість оцінки нової області відокремленості за знаком функції в новообраній точці будується схема послідовних наближень. За цією схемою обираються дві початкові точки x_0, x_1, а кожна наступна точка – претендент на значення кореня розраховується за наведеною формулою

$$x_{k+1} = \frac{x_k \cdot y_{k-1} + x_{k-1} \cdot y_k}{y_{k-1} - y_k}. \tag{1.13}$$

Оцінкою точності є модуль різниці координат двох послідовних точок

$$mes = |x_{k+1} - x_k| . \tag{1.14}$$

Алгоритм обчислення трохи спрощується за рахунок зменшення кількості умовних операторів.

```
program secant;
    const
        eps=1e-10; x1=0;x2=2;
```

```
    var
       xk,xkk,xnew,yk,ykk :Double;

function f(x:Double):Double;
  begin
    f:=x*exp(x)-1;
  end;

BEGIN
   xk:=x1;xnew:=x2; yk:=f(xk);
   while (abs(xnew-xkk)>eps)
   do begin
      xk:=xkk;yk:=ykk;xkk:=xnew;
      ykk:= f(xkk);
      xnew:=(xk*ykk+yk*xkk)/(ykk-yk);
      end;
  WriteLn(xnew);
END.
```

Програма обчислення кореня методом перетинів.

В програмі найбільш заплутаною є частина з перестановкою значень координат двох попередніх та нової точки. Пересилання значень між змінними (Xk:=xkk;yk:=ykk;) можна позбутись, якщо використати двоелементні масиви з автоматичним перерахуванням індексів. Оголосимо дві координати попередніх точок та координату розраховуваної нової точки, а також значення функції в попередніх точках, як масиви з трьох елементів

```
x,y :array[0..2] of double.
```

Використаємо позначення k для індексу передостанньої точки, kk для індексу останньої, а n – для індексу нової точки і на початку циклу будемо робити такі привласнення значень індексів:

```
k:=kk; kk:=n; n:=(n+1) mod 3;.
```

Тоді індекс k завжди буде позначати передостанню точку, kk – останню, а n – нову.

Дійсно, якщо перед циклом задані значення n:=1;kk:=0, тоді на початку першого проходження циклу індекси стануть такими:

```
k<=0;kk<=1;n<=2,
```

як і потрібно для першого проходження. На друге проходження циклу індекси отримають нові значення

```
k<=(kk=1)=1;kk<=(n=2)=2;n<=(((n=2)+1)=3)mod 3 =0,
```

тому нове значення кореня буде розташоване замість передостаннього, яке вже більше не потрібне. Наступним проходженням значення стануть такими

```
k<=(kk=2)=2;kk<=(n=0)=0;n<=(((n=0)+1)=1)mod 3 =1
```

тобто відбуватиметься циклічна підстановка номерів індексів.

Доцільність використання такого циклічного буфера для використовуваних змінних залежить від багатьох факторів і значною мірою від потужності використовуваного комп'ютера. Якщо в комп'ютері використовує-

ться процесор, що не здатний обробити за один такт операцію пересилання дійсного числа (16- або 32- розрядний), заміна пересилання дійсних чисел пересиланням невеликих цілих є досить ефективною. Для 64 – розрядних процесорів або для процесорів із вбудованим співпроцесором дійсної арифметики витрати часу на пересилання дійсних або перепривласнення цілих практично однакові. Ще суттєвішою слід вважати ту обставину, що витрати часу на перепривласнення звичайно є знехтовно малими порівняно з витратами на обчислення значення функції, тож з точки зору оптимізації програми покращення за рахунок складної системи індексації буде практично непомітним, а сприйнятність програми суттєво погіршиться. Набагато суттєвішим є зберігати вже розраховані значення функції, і в такий спосіб зменшувати кількість викликів розрахунку функції (на кожному кроці один раз замість двох). Таке накопичення прискорює роботу програми практично вдвічі.

1.1.3 Метод Ньютона

За основу цього методу обрано найгрунтовнішу ідею математичного аналізу про заміну графіку функції графіком дотичної. Замість кореня рівняння

$$f(x) = 0 \tag{1.15}$$

розраховується корінь рівняння

$$f(x_k) + f'(x_k) \cdot (x - x_k) \tag{1.16}$$

і отримане значення

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{1.17}$$

обирається за претендента на звання кореня. Така послідовність завжди швидко збігається (майже завжди, з точністю до питання про існування кореня, як такого та про кратність коренів).

Важливою частиною цього методу є використання явного виразу для похідної. Заміна похідної на її наближення скінченними різницями перетворює цей метод на метод перетинів.

Алгоритм методу Ньютона є досить прозорим:

Повинні бути означеними дві підпрограми – функції для обчислення значень розв'язуваної функції та її похідної. В розглядуваному нижче прикладі це

$$f(x) = x \cdot \exp(x) - 1;\ \ f'(x) = (1 + x) \cdot \exp(x). \tag{1.18}$$

Далі потрібно означити сталу eps=1e-9, що задає потрібну точність, та дві змінні – координату x наступної точки – претенденту на корінь, та обчислюване зміщення step.

Власне дії програми починаються з ініціалізації змінних. Нульове наближення до кореня задається навмання x:=0, а значення кроку step:=1

обирається виключно з того міркування, що до початку обчислень ця величина повинна бути більшою за eps.

Виконуваною дією є цикл з передумовою. Умова виконання тіла циклу є порівнянням на більше абсолютної величини кроку та очікуваної похибки, а тіло циклу складається з обчислення наступного кроку за формулою

$$step = -\frac{f(x)}{f'(x)} \tag{1.19}$$

та зміни наближеного значення кореня

$$x <= x + step. \tag{1.20}$$

Виконання програми досить зручно здійснюється за допомогою системи Maple. Власне програма наведена далі разом з результатом обчислення

```
> ff:=x->x*exp(x)-1:
> fp:=x->(1+x)*exp(x):
> eps:=1.0e-9:x:=0:step:=1.0:n:=0:
> while(abs(step)>eps)
   do step:=evalf(-ff(x)/fp(x));x:=x+step; n:=n+1; od;

 step := 1.
 x := 1.
 n := 1
 step := -.3160602794
 x := .6839397206
 n := 2
 step := -.1064852434
 x := .5774544772
 n := 3
 step := -.01022473948
 x := .5672297377
 n := 4
 step := -.00008644115508
 x := .5671432965
 n := 5
 step := -.6152236303 10$^{ - 8}$
 x := .5671432903
 n := 6
 step := .1447585027 10$^{ - 9}$
 x := .5671432904
 n := 7
> Digits := 20:
```

Останній рядок містить інформацію, необхідну для обчислення коренів з похибкою, меншою за 10^{-10}.

Порівняння метода Ньютона з методом ділення навпіл чи метода перетинів свідчить про прискорену порівняно з іншими методами збіжність (кількість кроків дещо менша). Але для досить складних в обчисленні функцій ця перевага втрачається, оскільки метод Ньютона потребує на кожному кроці два складних обчислення – самої функції та її похідної.

1.1.4 Метод простих ітерацій

Цей метод полягає в використанні простого твердження про співпадіння коренів рівняння $f(x) = 0$ та рівняння $x = x + f(x)$. Останнє рівняння переписується у вигляді рекурентного виразу

$$x_{n+1} = x_n + f(x_n) \tag{1.21}$$

що дозволяє, починаючи з деякого x_0 розрахувати послідовність $\{x_n\}$. Якщо ця послідовність збігається, її границею є розшукуваний корінь. Критерій збіжності легко отримати, розглянувши відхилення членів послідовності від розв'язку $u_n = x_n - \tilde{x}$. Вважаючи ці відхилення малими, можна в першому наближенні замінити значення функції на перший член її розкладення в ряд Тейлора поблизу кореня

$$f(\tilde{x} + u_n) \approx f(\tilde{x})\,(= 0) + f'(\tilde{x})\,u_n. \tag{1.22}$$

Тоді для послідовності відхилень маємо рекурентний вираз

$$u_{n+1} \approx u_n + f'(\tilde{x})\,u_n = (1 + f'(\tilde{x}))\,u_n, \tag{1.23}$$

з якого витікає, що послідовність є геометричною прогресією із знаменником $q = 1 + f'(\tilde{x})$. Послідовність збігається, якщо знаменник за модулем менший за одиницю. Зрозуміло, що обчислити значення похідної в невідомій точці не можна, тому використанню методу простих ітерацій звичайно передує оцінка інтервалу можливих значень похідної в околі очікуваного значення кореня.

1.1.5 Коментарі

Всі перераховані методи насправді є ітераційними, тобто всі вони ґрунтуються на послідовному покращенні обраного зарані претендента на корінь. Вибір того чи іншого метода здійснюється на підставі уявлень про можливі властивості функції поблизу кореня. Найпростіший в реалізації (і найшвидший) метод простих ітерацій, на жаль часто не збігається, тому він не використовується в типових пакетах програм, таких як Maple чи Mathematica. Тому використання стандартних підпрограм пошуку коренів є доцільним тільки в тому разі, коли необхідно обчислити один чи декілька коренів. Якщо ж пошук коренів є частиною масової операції більш складної програми і суттєво впливає на швидкодію, доцільніше використовувати саморобні процедури, що враховували б специфіку розв'язуваного рівняння. Найшвидших результатів слід чекати від методу простої ітерації, але для цього методу завжди залишається питання про збіжність.

1.2 Системи лінійних рівнянь

Застосування обчислювальних методів до систем лінійних рівнянь

$$Ax = f \tag{1.24}$$

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix};\ x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix};\ f = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}, \tag{1.25}$$

є необхідним тільки з уваги на значний обсяг обчислень, що потрібний для розрахунку розв'язку. Досить часто задача на розв'язування системи лінійних рівнянь виникає, як складова частина більш складної задачі. В теоретичних побудовах в такому разі досить часто просто використовується твердження: „позначимо $\{x_k\}$ розв'язок системи". В більш практичних задачах необхідно, зрозуміло ж, використовувати чи то самі розв'язки, чи їх властивості. В такому разі найголовніший метод лінійної алгебри - метод визначників або метод Крамера, не може вважатись досить ефективним. Стандартний, тобто за означенням, метод обчислення визначника полягає в розкритті визначника, як суми добутків певного рядка або стовпчика на відповідні мінори

$$\begin{aligned} &\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} = a_{11} \begin{pmatrix} a_{22} & \cdots & a_{2,n} \\ \vdots & \ddots & \vdots \\ a_{n,2} & \cdots & a_{nn} \end{pmatrix} + \\ &+ \ldots + \\ &+ (-1)^{m-1} a_{1m} \begin{pmatrix} a_{21} & a_{2,m-1} & a_{2,m+1} & a_{2,n} \\ \vdots & \ldots & \ldots & \vdots \\ a_{n,1} & a_{n,m-1} & a_{n,m+1} & a_{nn} \end{pmatrix} + \\ &+ \ldots + \\ &+ (-1)^{n-1} a_{1m} \begin{pmatrix} a_{21} & \cdots & a_{2,n-1} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{nn-1} \end{pmatrix} \end{aligned} \tag{1.26}$$

Підрахуємо загальну кількість дій, що потрібні для обчислення визначника рангу n. Позначимо потрібну кількість дій $D(n)$, тоді відповідно до методу мінорів ця кількість дорівнює сумі n добутків послідовних членів рядка на кількість дій, що потрібна для обчислення одного мінору на одиницю меншого рангу. Оскільки мінор теж є визначником, потрібна для його обчислення кількість дій визначається тією ж величиною $D(n-1)$, тільки з рангом, на одиницю меншим. Тому для кількості дій маємо співвідношення $D(n) = nD(n-1)$. Таке рекурентне співвідношення тільки сталим множником може відрізнятись від факторіала, тобто отримуємо таку оцінку кількості дій $D(n) \sim n!$. Розв'язок системи лінійних рівнянь потребує розрахунку визначника самої системи та n додаткових визначників, тобто

загальна кількість дій пропорційна $(n+1)!$. Ця оцінка вже була використана для розрахунку обчислювальної складності задачі розв'язування системи 15 лінійних рівнянь. Трохи завищеною оцінкою факторіала є значення n^n, що і дає для цього прикладу $16^{16}{=}2^{64}{=}10^{18}$ дій. Навіть комп'ютер з швидкістю обчислень 10^9 дій за секунду(1 Гігафлоп) витратить 10^6 секунд (біля 10 діб) на здійснення цих дій.

Серед методів скорочення обчислень, потрібних для отримання розв'язку системи лінійних рівнянь, розрізняють стандартний метод Гауса (метод підстановок), штучні методи, що враховують спеціальні властивості окремих типів матриць, та ітераційні методи.

Метод Гауса вважається точним і він таким і лишається, поки обчислення здійснюються аналітично. В комп'ютерній реалізації метод Гауса стає наближеним, оскільки додаються похибки, обумовлені обмеженнями на комп'ютерне подання дійсних чисел. Навіть обчислення з раціональними числами найчастіше е наближеними, якщо тільки знаменник не є степенем двійки. Найточніше з стандартних зображень (double) використовує 56 бітів для мантиси числа, тому відносна похибка не менша за $2^{-57}\sim 10^{-17}$. За рахунок накопичення похибка методу Гауса, що потребує n^3 дій, вже для системи 100 рівнянь збільшується в $100^3{=}10^6$ разів, тому точність розрахунків, що здійснювались з використанням арифметики подвійної точності, відповідає одинарній точності зображення.

На відміну від „точного" метода Гауса, ітераційні методи виявляються відносно точнішими, оскільки на кожному кроці ітерації оцінюють саме похибку наближеного розв'язку.

1.2.1 Метод Гауса

Висхідною ідеєю метода є найпростіша підстановка, що використовується досить часто навіть при розв'язуванні системи з двох рівнянь. Розглянемо детально, як саме здійснюється метод підстановки.

Система

$$\begin{array}{lll} ax & +by & = e \\ cx & +dy & = f \end{array} \tag{1.27}$$

може бути спрощеною, якщо за допомогою першого рівняння записати вираз $x = \frac{e-by}{a}$ для першої невідомої, як функції другої і підставити в друге рівняння. Тоді система перетворюється на таку

$$\begin{array}{lll} ax & +by & = e \\ 0x & +\left(d-\frac{cb}{a}\right)y & = f-\frac{ce}{a} \end{array} \tag{1.28}$$

в якій друге рівняння розв'язується зовсім просто, а потім отриманий вираз для другої невідомої можна використати для знаходження першої (за допомогою першого рівняння).

Для системи з трьома невідомими подібну підстановку потрібно робити двічі і послідовність дій стає трохи плутанішою, а проміжні вирази – надто

непрозорими. Використання ж цього метода для систем з великою кількістю дій стає надто складним.

Суттєвого спрощення алгоритму можна досягти, якщо взяти до уваги, що наслідком підстановки є перетворення матриці системи до верхнього трикутного виду, адже до підстановки матриця мала загальну форму з усіма довільними коефіцієнтами

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{1.29}$$

а після підстановки – отримала так звану верхню трикутну форму, тобто таку, що нижче головної діагоналі всі коефіцієнти дорівнюють нулю

$$A_{tr} = \begin{pmatrix} a & b \\ 0 & \tilde{d} \end{pmatrix} \tag{1.30}$$

З курсу лінійної алгебри добре відомо, що до верхньої трикутної форми можна привести довільну матрицю, тобто потрібна підстановка завжди існує.

Якщо матрицю системи з якою завгодно кількістю невідомих перетворити на верхню трикутну, перетворена система $A_{tr}x = f_{tr}$ з верхньою трикутною матрицею

$$A_{tr} = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ 0 & a_{2,2} & \dots & a_{2,n} \\ & & \ddots & \vdots \\ 0 & 0 & & \\ 0 & 0 & 0 & a_{n,n} \end{pmatrix} \tag{1.31}$$

розв'язується зовсім просто:

Записавши явно систему перетворених рівнянь

$$\begin{array}{llll} a_{1,1}x_1 & +a_{1,2}x_2 & \dots & +a_{1,n}x_n = f_1 \\ 0 & a_{2,2}x_2 & \dots & +a_{2,n}x_n = f_2 \\ & & \ddots & \vdots \\ 0 & 0 & & \\ 0 & 0 & 0 & a_{n,n}x_n = f_n \end{array} , \tag{1.32}$$

легко побачити, що останнє рівняння розв'язується в одну дію, а як тільки став відомим його розв'язок, підстановкою отриманого значення передостаннє рівняння також перетворюється на рівняння з однією невідомою $a_{n-1,n-1}x_{n-1} = f_{n-1} - a_{n-1,n}x_n$. Взагалі рівняння для k -тої невідомої після підстановки попередньо обчислених значень невідомих з більшими номерами перетворюється на рівняння з однією невідомою і так само легко розв'язується. Загальна кількість дій, потрібних для обчислення розв'язків перетвореної системи, визначається, як кількість дій, потрібних для підстановки попередньо обчислених значень. Для останнього рівняння таких дій нема, для передостаннього – одна підстановка, для рівняння з номером $n-k-k$ підстановок, тож загальна кількість є сумою $1+2+3+..+n \sim n^2$ і пропорційна другому степеню кількості рівнянь замість факторіальної складності.

Трохи складнішим є перетворення матриці рівняння до верхнього діагонального вигляду.

Замість безпосереднього відтворення виразу для першої змінної через усі інші з наступною підстановкою можна просто використати можливість заміни в системі лінійних рівнянь одного з рівнянь лінійною комбінацією цього ж самого рівняння та будь-якого іншого. Якщо друге рівняння замінити на лінійну комбінацію його самого та першого рівняння, розв'язки системи не зміняться, а коефіцієнти заміни можна обрати в такий спосіб, щоб новий (перетворений) коефіцієнт $a_{2,1}$ дорівнював нулю. Це здійснюється заміною всіх коефіцієнтів та вільного члена другого рівняння за правилом

$$a_{2,k} \Leftarrow a_{2,k} - a_{1,k}\frac{a_{2,1}}{a_{1,1}}, \quad f_2 \Leftarrow f_2 - f_1\frac{a_{2,1}}{a_{1,1}} \tag{1.33}$$

Після такої заміни другого рівняння подібне перетворення потрібно зробити з третім, потім четвертим і так поки не будуть перебрані всі рівняння системи. Результатом перетворення буде еквівалентна початковій система лінійних рівнянь, в якій весь перший стовпчик заповнений нулями (крім верхнього рядка)

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix} \Rightarrow \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ 0 & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n,2} & \dots & a_{n,n} \end{pmatrix} \tag{1.34}$$

Далі лишається тільки застосувати цей алгоритм до підматриці, що починається с другого стовпчика та другого рядка

$$\begin{pmatrix} a_{2,2} & a_{2,3} & \dots & a_{2,n} \\ a_{3,2} & a_{3,3} & \dots & a_{3,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,2} & a_{n,3} & \dots & a_{n,n} \end{pmatrix} \Rightarrow \begin{pmatrix} a_{2,2} & a_{2,3} & \dots & a_{2,n} \\ 0 & a_{3,3} & \dots & a_{3,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n,3} & \dots & a_{n,n} \end{pmatrix} \tag{1.35}$$

і продовжувати доти, доки не лишиться „матриця" з одного рядка та одного стовпчика. Кожного разу в наступному стовпчику елементи нижче головної діагоналі будуть перетворюватись на нуль і результатом буде верхня трикутна матриця.

Підрахуємо тепер обчислювальну складність процесу перетворення матриці до трикутної форми. Перетворення одного рядка потребує такої кількості дій, скільки елементів міститься в рядку, а перетворення всіх рядків - n^2 дій. На наступному етапі перетворюються вже не всі елементи, а на один менше, тобто потрібно $(n-1)^2$ і так поки кількість елементів, що потребують перетворення, не дійде до одного. Сума квадратів, принаймні с точністю до старшого степеню, пропорційна кубу розмірності системи,

тож обчислювальна складність процесу перетворення до верхньої трикутної форми є кубічною. Оскільки процес обчислення розв'язків має тільки квадратичну складність, в цілому метод Гауса має кубічну складність.

Перед побудовою алгоритму розглянемо виняткові ситуації, що можуть заважати ефективному використанню методу.

Оскільки на кожному кроці відбувається ділення на черговий діагональний елемент, алгоритм не буде працювати, якщо цей елемент дорівнює нулю. Це може трапитись випадково і тоді непрацездатність алгоритму обумовлена його властивостями. Позбутися такого недоліку можна, просто помінявши місцями верхній рядок з яким-небудь іншим, в якому найлівіший елемент не дорівнює нулю. Якщо ж такий елемент відсутній легко побачити, що визначник матриці дорівнює нулю. Дійсно, визначник матриці, в якій в першому стовпчику відмінним від нуля є тільки елемент з головної діагоналі, дорівнює добуткові цього елемента на визначник його мінора. Якщо і мінор вже має перший стовпчик таким, що нижче діагонального всі елементи дорівнюють нулю, то визначник матриці дорівнює добуткові всіх елементів над діагоналлю та визначника мінора, що лишається. Якщо ж у якогось мінора цілий стовпчик дорівнює нулю, то і його визначник є нулем.

Ще одним важливим джерелом неефективності алгоритму може бути обчислювальна похибка. Вона буде тим більшою, чим більше наближаються один до одного значення, що віднімаються. Дійсно, якщо відняти, припустимо, від числа 1.994 , поданого з похибкою 0.005 як 1.99, число 1.976, задане з такою ж похибкою як 1.98, результатом буде 0.01, тоді як більш точне подання дало б 0.018 і відносна похибка тепер дорівнює 0.5. Тому важливим є зменшити можливість віднімання одне від одного близьких за величиною чисел. Цього можна досягти, перебираючи на кожному кроці найбільший за абсолютною величиною елемент лівого стовпчика і міняючи місцями відповідні рядки.

Ще одним важливим ускладненням чи, навпаки, спрощенням алгоритму є перетворення діагонального коефіцієнта на одиницю. Таке перетворення суттєво спрощує запис процесу обчислення розв'язків, а більш важливо, воно спрощує і запис алгоритму зведення до трикутного вигляду.

Наступним спрощенням/ускладненням є поєднання матриці коефіцієнтів та стовпчика правих частин рівнянь в одну матрицю, але вже не квадратну, а прямокутну розмірностей $n \times (n+1)$. Це дозволяє записати в одному циклі перетворення коефіцієнтів матриці та правих частин.

Отже, зведення до трикутного вигляду може здійснюватись, як послідовні перетворення на нулі елементів першого стовпчика матриці та її підматриць. Якщо виділити в окрему процедуру `Thre(k)` перетворення на нуль лівого стовпчика чергової підматриці, починаючи з k - того, то вся послідовність перетворень буде полягати в циклічному виклику цієї процедури. Такий виклик можна зробити рекурсивним з самої процедури, якщо закінчувати її умовним оператором `if(k<n) then Thre(k+1)`.

Тіло процедури повинно розпочинатись пошуком найбільшого з елементів стовпчика. Результатом пошуку буде номер цього елементу km – внутрішня змінна процедури, а в процесі пошуку буде потрібно зберігати зна-

чення, найбільше з вже перебраних. Для нього також потрібно підготувати внутрішню змінну `temp`.

Цикл пошуку виглядає зовсім просто

```
temp:=abs(a[k,k]);km:=k;
for kv:=k to n
do if abs(a[kv,k])>Temp
   then begin temp:=abs(a[kv,k]);km:=kv;end;
```

Умовний оператор в цьому циклі виконує `then` – блок тільки в тому разі, якщо наступний елемент більше за всі попередні. Тоді в тимчасовій змінній зберігається нове найбільше значення і зберігається номер цього рядка.

По закінченні циклу, якщо найбільше значення знайдене не в першому рядку, потрібно поміняти місцями перший рядок і той, в якому лівий елемент був найбільшим. Таку заміну можна зробити, перебравши всі елементи обох рядків, то знов потрібен цикл, але тепер він захоплюватиме й елементи стовпчика правих частин

```
if km<>k
  then for kv:=k to n+1
       do begin
          temp:=a[k,kv];a[k,kv]:=a[km,kv];a[km,kv]:=temp;
          end;
```

Ще один допоміжний цикл потрібен для перетворення діагонального елемента на одиницю.

```
temp:=a[k,k]; for kv:=k to n+1 do a[k,kv]:=a[k,kv]/temp;
```

Відтепер можна перебирати один за одним рядки і заміняти їх на такі лінійні комбінації з першим, що лівий елемент дорівнюватиме нулю.

Для перебору рядків та елементів рядка нам потрібні дві індексних змінних. Оскільки змінна km вже звільнилася від обов'язків зі збереження номера стовпчика з найбільшим елементом, її можна використати для перебору стовпчиків

```
for km:=k+1 to n
do begin
   temp:=a[km,k];
   for kv:=k to n+1 do a[km,kv]:=a[km,kv]-temp*a[k,kv];
   end;
```

Обчислення розв'язків можна здійснити в самій процедурі `Gauss`.

Елементи вектора – відповіді потрібно перебирати в зворотньому напрямку, від найбільшого до найменшого, тому цикл перебору елементів вектора буде мати вигляд

```
for kr:=n downto 1
do begin
```

```
     x[kr]:=a[kr,n+1];
     for ks:=n downto kr+1 do x[kr]:=x=kr]-a[kr,ks]*x[ks];
     end;
```

Єдине поки що випущене з уваги виключення полягає в перевірці значення визначника. Під час пошуку найбільшого елемента значення змінної temp буде нулем тільки в тому разі, якщо всі елементи відповідного стовпчика дорівнюють нулю. Отже, перевірка на нуль значення цієї змінної є необхідним методом контролю за існуванням розв'язку.

Програма

```
program GaussCalc;

const
    nm=80;
    dataname='data.txt';resname='res.txt';
var
   a:Array[1..nm,1..nm+1] of double;
   x:Array[1..nm] of double;
   N:Integer;

procedure Thre(k:Integer);
 var
    km,kv :Integer;
    temp  :Double;
  begin
     temp:=abs(a[k,k]);km:=k;
     for kv:=k to n
     do if abs(a[kv,k])>temp
        then begin
             temp:=abs(a[kv,k]);km:=kv;
             end;
     if temp=0 then Error;
     temp:=a[k,k];
     for kv:=k to n+1 do a[k,kv]:=a[k,kv]/temp;
     for km:=k+1 to n
     do begin
        temp:=a[km,k];
        for kv:=k to n+1 do a[km,kv]:=a[km,kv]-temp*a[k,kv];
        end;
     if(k<n) then Thre(k+1).
  end;

procedure Gauss;
  var
```

```
    kr,ks :Integer
  begin
     Thre(1);
     for kr:=n downto 1
     do begin
        x[kr]:=a[kr,n+1];
        for ks:=n downto kr+1 do x[kr]:=x=kr]-a[kr,ks]*x[ks];
        end;
  end;

procedure data;
  var
     ks        :Integer;
     datafile  :Text;
  begin
     assign(datafile,dataname);
     reset(datafile);
     n:=0;
     while not Eof(datafile)
     do begin
        n:=n+1;ks:=0;
        while not Eol(datafile)
        do begin ks:=ks+1; Read(datafile,a[n,ks]);end;
        ReadLn(datafile);
        end;
     close(datafile);
  end;

procedure Result;
 var
    k        :Integer;
    resfile  :Text;
 begin
    assign(resfile,resname);
    rewrite(resfile);
    for k:=1 to n do WriteLn(resfile,k, x[k]);
    close(resfile);
 end;

BEGIN
   Data;
   Gauss;
   Result;
END.
```

Інтерфейсна частина програми передбачає, що вхідні дані підготовлені

без помилок. Вхідний файл повинен складатись з послідовності рядків матриці коефіцієнтів, а кожен рядок закінчуватись відповідним членом правої частини рівняння. Кількість елементів рядка не перевіряється, а кількість рядків використовується, як розмірність системи рівнянь.

Результатом роботи програми є файл, що вміщує пари – номер/значення вектора – розв'язку системи.

1.2.2 Метод прогонки

В практичних задачах досить часто зустрічаються системи лінійних рівнянь з розрідженими матрицями, тобто такими, що мають більшість елементів рівними нулю. Для таких систем звичайно можна винайти штучні алгоритми, що добре використовують специфіку явного вигляду матриці. Більше того, в задачах математичної фізики досить часто навіть алгоритми побудови розв'язку підбираються в такий спосіб, аби використати певні штучні алгоритми.

Найбільш ефективним алгоритмом розв'язування систем лінійних рівнянь є метод прогонки. Він базується на припущенні, що матриця рівняння має ненульові елементи тільки на головній діагоналі та двох сусідніх до неї, тобто є тридіагональною.

$$A = \begin{pmatrix} a_{11} & a_{12} & 0 & ... & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ ... & ... & ... & ... & ... \\ 0 & 0 & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & a_{n,n-1} & a_{nn} \end{pmatrix} \tag{1.36}$$

Такі матриці зустрічаються, наприклад, в задачах сплайн – інтерполяції та в задачах розповсюдження тепла або електромагнітних хвиль.

Кожне рівняння тридіагональної матриці зв'язує між собою тільки три з великої кількості невідомих і його можна подати у вигляді так званого масового рівняння

$$a_k x_{k-1} + x_k + b_k x_{k+1} = f_k. \tag{1.37}$$

Слід зауважити, що перше і останнє рівняння тридіагональної матриці мають скорочений вигляд

$$\begin{cases} \mu x_0 + \nu x_1 = f_0 \\ \rho x_{n-1} + \sigma x_n = f_n \end{cases} \tag{1.38}$$

оскільки кількість невідомих вже вичерпана.

Найважливішою перевагою таких рівнянь є те, що для них методом прогонки можна отримати розв'язок за кількість дій, пропорційну вже не третьому навіть, як в методі Гауса, а першому степеню кількості невідомих. З урахуванням цієї обставини тридіагональні рівняння не дуже вже й відрізняються від системи незалежних рівнянь.

Головна ідея методу прогонки полягає в використанні припущення про існування рекурентного виразу, що пов'язує між собою послідовні компоненти вектора – розв'язку

$$x_{k+1} = \alpha_k x_k + \beta_k. \tag{1.39}$$

Якщо підібрати коефіцієнти цього рекурентного виразу в такий спосіб, щоб задовольнялось масове рівняння, можна отримати повний розв'язок всієї системи, спираючись на заранні визначене значення x_0. Дійсно, рекурентний вираз за відомим x_0 дозволяє обчислити x_1, за ним x_2 і так до останнього x_n. Зрозуміло, що отримані значення будуть задовольняти масове рівняння тільки в тому разі, якщо вдало підібрані коефіцієнти α_k та β_k рекурентного виразу.

Підставимо рекурентний вираз в масове рівняння

$$a_k x_{k-1} + x_k + b_k \left(\alpha_k x_k + \beta_k\right) = f_k \tag{1.40}$$

і звернімо увагу на те, що рекурентний вираз зв'язує між собою також попередню пару невідомих

$$x_k = \alpha_{k-1} x_{k-1} + \beta_{k-1} \tag{1.41}$$

Підставивши також і цей вираз в масове рівняння, отримуємо

$$a_k x_{k-1} + \left(1 + b_k \alpha_k\right)\left(\alpha_{k-1} x_{k-1} + \beta_{k-1}\right) + b_k \beta_k = f_k \tag{1.42}$$

або після спрощення

$$\left(a_k + \left(1 + b_k \alpha_k\right)\alpha_{k-1}\right) x_{k-1} + \left(1 + b_k \alpha_k\right)\beta_{k-1} + b_k \beta_k = f_k \tag{1.43}$$

Коефіцієнти обираються такими, аби масове рівняння задовольнялось для довільного розв'язку, тому окремо повинні дорівнювати нулю член з невідомим та вільний член

$$\begin{cases} a_k + \left(1 + b_k \alpha_k\right)\alpha_{k-1} = 0 \\ \left(1 + b_k \alpha_k\right)\beta_{k-1} + b_k \beta_k = f_k \end{cases} \tag{1.44}$$

Отримані в такий спосіб рівняння знов-таки є рекурентними виразами, що дозволяють розрахувати послідовності коефіцієнтів.

Розглянемо тепер кінцеві рівняння.

Перше з них

$$\mu x_0 + \nu x_1 = f_0 \tag{1.45}$$

буде сумісним з рекурентним виразом $x_1 = \alpha_0 x_0 + \beta_0$ тільки в тому разі, якщо x_0 задовольняє рівняння $\mu x_0 + \nu\left(\alpha_0 x_0 + \beta_0\right) = f_0$, або, після спрощення

$$x_0 = \frac{f_0 - \nu\beta_0}{\mu + \nu\alpha_0} \tag{1.46}$$

Це рівняння задає початковий елемент для обчислення послідовності значень розв'язку за рекурентним виразом.

Друге кінцеве рівняння

$$\rho x_{n-1} + \sigma x_n = f_n \tag{1.47}$$

сумісне з відповідним рекурентним виразом

$$x_n = \alpha_{n-1} x_{n-1} + \beta_{n-1} \tag{1.48}$$

тільки в тому разі, якщо задовольняється для яких завгодно значень x_{n-1}, x_n. Це можливо за умови, що вираз

$$\rho x_{n-1} + \sigma \left(\alpha_{n-1} x_{n-1} + \beta_{n-1}\right) = f_n \tag{1.49}$$

задовольняється незалежно від розрахованого значення x_{n-1}. Тому повинні задовольнятись співвідношення

$$\begin{cases} \rho + \sigma\alpha_{n-1} = 0 \\ \sigma\beta_{n-1} = f_n \end{cases} \tag{1.50}$$

Ці два співвідношення дозволяють розрахувати початкові значення для рекурентних виразів щодо коефіцієнтів.

Підсумковий алгоритм має таку структуру

З співвідношень

$$\begin{cases} \alpha_{n-1} = -\frac{\rho}{\sigma} \\ \beta_{n-1} = \frac{f_n}{\sigma} \end{cases} \tag{1.51}$$

обчислюємо початкові значення для послідовностей коефіцієнтів, далі за рекурентними виразами

$$\begin{cases} a_{k-1} = -\frac{\alpha_k}{(1+b_k\alpha_k)} \\ \beta_{k-1} = \frac{f_k - b_k\beta_k}{1+b_k\alpha_k} \end{cases} \tag{1.52}$$

всі потрібні коефіцієнти (k пробігає значення від $n-2$ до 0).

За виразом

$$x_0 = \frac{f_0 - \nu\beta_0}{\mu + \nu\alpha_0} \tag{1.53}$$

обчислюється початковий член послідовності розв'язків і, нарешті, за рекурентним виразом

$$x_{k+1} = \alpha_k x_k + \beta_k \tag{1.54}$$

- всі члени цієї послідовності (k пробігає значення від 0 до $n-1$).

Кожен етап алгоритму має лінійну складність, тож загальна кількість дій, потрібна для обчислення розв'язку, є пропорційною кількості рівнянь.

Реалізація методу прогонки в Maple

```
> restart:
> path:="path to maple\\sweep\\":
> dataname:="data.txt":resname:="res.txt":
> source:=cat(path,dataname):target:=cat(path,resname):
```

підготовка даних про файл з параметрами та файл результатів

```
> dat:=readdata(dataname,4);nmax:=nops(dat);
```

$$dat := [[1., .5, 1.], [.5, 1., .5, .5], [.5, 1., .5, 0.], [.5, 1., .5, 0.], [.5, 1., .5, 0.], [.5, 1., 0.]]$$

$$nmax := 6$$

зачитування даних з файлу. Кожен рядок містить коефіцієнти чергового рівняння системи. В першому рядку заносяться a, b, f з рівняння $ax_1 + bx_2 = f$, в другому - a, b, c, f з рівняння $ax_1 + bx_2 + cx_3 = f$, в k-тому - відповідно з рівняння а $x_k + bx_{k+1} + cx_{k+2} = f$. Останній рядок містить коефіцієнти r, s, f останнього рівняння $rx_{n-1} + sx_n = f$

```
> alpha:=array(2..nmax):beta:=array(2..nmax):
> x:=array(1..nmax):
```

підготовка масивів для рекурсивних коефіцієнтів та результату

```
> am:=[seq(dat[i,2],i=2..nmax-1)];
> au:=[seq(dat[i][3],i=2..nmax-1)];
> ad:=[seq(dat[i][1],i=2..nmax-1)]; f:=[seq(dat[i][4],i=2..nmax-1)];
```

підготовка даних в формат, зручніший для обчислення: am - діагональні елементи, au - елементи над діагоналлю, ad - під діагоналлю, f - права частина рівняння. Перебрані всі члени масового рівняння

```
> a:=dat[1][1]:b:=dat[1][2]:f1:=dat[1][3]:
> r:=dat[nmax][1]:s:=dat[nmax][2]:fn:=dat[nmax][3]:
```

коефіцієнти першого та останнього рівнянь

```
> alpha[nmax]:=-r/s:beta[nmax]:=fn/s: for
> k from nmax-1 to 2 by -1 do
> alpha[k]:=-ad[k-1]/(au[k-1]*alpha[k+1]+am[k-1]);
> beta[k]:=(f[k-1]-au[k-1]*beta[k+1])/(au[k-1]*alpha[k+1]+am[k-1]);od:
```

розрахунок рекурсивних коефіцієнтів: початкові елементи та обчислення членів послідовності

```
> x[1]:=(f1-b*beta[2])/(a+b*alpha[2]): for
> k from 2 to nmax do x[k]:=alpha[k]*x[k-1]+beta[k]; od:
```

Розрахунок відповіді за рекурсивною формулою

```
> rtable(x);
> writedata(target,x);
```

Відтворення відповіді у вигляді послідовності та запис у файл

Зміст файлу вхідних даних

```
   1   0.5     1
 0.5     1   0.5   0.5
 0.5     1   0.5     0
 0.5     1   0.5     0
 0.5     1   0.5     0
 0.5     1     0
```

1.3 Метод простої ітерації

Ітераційні методи відрізняються від прямих тим, що в них ставиться за мету обчислення не точного, а наближеного розв'язку. З уваги на похибки округлення прямі методи також дають тільки наближений розв'язок, тому ітераційні методи, принаймні для великих систем, можуть давати більш точний розв'язок, ніж прямі. В усякому разі, точність розв'язку, отриманого ітераційними методами, є контрольованою величиною.

Ітераційні методи спираються на оцінку похибки наближеного розв'язку

$$\tilde{x} = \begin{pmatrix} \tilde{x}_1 \\ \vdots \\ \tilde{x}_n \end{pmatrix} \tag{1.55}$$

за допомогою нев'язку рівняння $v = A\tilde{x} - f$ і використання цього самого нев'язку для уточнення розв'язку.

Метод прямої ітерації використовує нев'язок, як напрямок, в якому шукається поліпшене значення наближеного розв'язку, тобто за кращий наближений розв'язок обирається величина $x_{best} = \tilde{x} + \tau v$. Важливою частиною методу та його модифікацій є вибір масштабного множника τ для кроку ітерації.

Аналітичним обґрунтуванням методів ітерації є різноманітні форми для обернення матриць, адже задача розв'язку лінійного рівняння

$$Ax = f \tag{1.56}$$

може бути переформульована, як задача пошуку оберненої матриці A^{-1} і обчислення розв'язку за її допомогою за формулою

$$x = A^{-1}f \tag{1.57}$$

Якщо, наприклад, ввести матрицю $B = I - \frac{1}{d}A$, то задача обернення матриці перетворюється на задачу розрахунку виразу

$$A^{-1} = \frac{d}{I - B} = d\frac{I}{I - B} \tag{1.58}$$

і за умови збіжності ряду може бути розв'язана за допомогою розкладення в ряд

$$A^{-1} = d\left(I + B + B^2 + ...\right) \tag{1.59}$$

Оцінимо умови збіжності ряду. З цією метою розглянемо базис, в якому матриця B діагональна, тобто має вигляд

$$B = \begin{pmatrix} b_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & b_n \end{pmatrix} \tag{1.60}$$

В цьому базисі вираз для оберненої матриці, як ряду матриць перетворюється на матрицю рядів

$$A^{-1} = d \begin{pmatrix} 1 + b_1 + b_1^2 + \dots & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & 1 + b_n + b_n^2 + \dots \end{pmatrix} \tag{1.61}$$

і кожен з компонентів цієї матриці збігається, якщо всі власні значення b_n менші за одиницю. Параметр d і був введений в означення матриці B саме з метою забезпечення збіжності ряду.

Спробуємо оцінити, яке саме значення потрібно обрати для параметру збіжності d, аби всі ряди гарантовано збігались. Якби можна було використати власні значення матриці B, умови $\max|\lambda_B| < 1$ було б цілком достатньо, але власні значення матриці B відомі нам тільки в тому разі, якщо відомі власні значення висхідної матриці A, а в такому разі звичайно відомо і про власні вектори і задачу обернення можна вважати практично розв'язаною. Тому потрібно було б мати більш загальні критерії для вибору величини параметру збіжності d. На жаль, загальнопридатного способу аналізу збіжності не існує. Найчастіше використовуються штучні оцінки, властиві конкретним класам рівнянь.

Припускаючи, що послідовність матриць

$$A^{-1} = d\left(I + B + B^2 + ...\right) \tag{1.62}$$

збігається, розв'язок можна зобразити у такому вигляді

$$x = A^{-1}f = d\left(I + B + B^2 + \dots\right) f = d\left(f + Bf + B^2f + \dots\right), \tag{1.63}$$

що йому досить легко надати рекурсивної форми

$$\begin{cases} x_0 = 0 : u_0 = df; \\ x_1 = x_0 + u_0; u_1 = Bu_0; \\ \vdots \end{cases} \tag{1.64}$$

Збіжності суми матриць відповідає збіжність до нуля послідовності векторів $\{u_k; u_0 = df, u_{k+1} = Bu_k; k = 0 \dots \infty\}$

Вважаючи цю збіжність рівномірною, слід чекати, що починаючи з деякого номера компоненти векторів $u_{k>bound}$ будуть порівнюваними за величиною з обчислювальною похибкою. Тоді скінченну суму

$$\tilde{x} = \sum_{k=0}^{bound} u_k, \tag{1.65}$$

можна обрати за розв'язок рівняння, оскільки всі спроби отримати точніший вираз для розв'язку будуть відрізнятись від цієї суми не більше, ніж на похибку обчислень.

Мистецтво використання ітераційних методів полягає не тільки в доборі величини масштабного множника, а й в організації способів обчислення наступних членів ряду, або безпосередньо наступних наближень до розв'язку.

Нехтуючи можливим масштабним множником, підставимо означення допоміжної матриці B безпосередньо в рівняння

$$(I - B)\, x = f, \tag{1.66}$$

або

$$x = f + Bx. \tag{1.67}$$

Вважаючи внесок другого члена рівняння малим, за нульове наближення обираємо $x_0 = f$ праву частину висхідного рівняння, тоді першим наближенням буде $x_1 = f + Bx_0$, а для наступних наближень маємо рекурентне співвідношення

$$x_{k+1} = f + Bx_k \tag{1.68}$$

Серед ітераційних методів найпоширенішими є схема простої рекурсії, коли на кожному кроці спочатку повністю обчислюються нові значення за попередніми, а вже потім відбувається заміна попереднього претендента на розв'язок новим, та схема Зейделя, за якою кожна нова компонента вектора – відповіді одразу заміняє стару. При обчисленні кожної чергової компоненти одночасно використовується частина компонент розв'язку від попереднього кроку – це ті компоненти, що будуть обчислюватись пізніше, та частина компонент нового претенденту на розв'язок, а саме ті компоненти, що були тільки-но розраховані.

Пряма рекурсія потребує використання додаткової векторної змінної, що повинна накопичувати значення нового претенденту на розв'язок, тоді як схема Зейделя використовує тільки один масив для розв'язку і цим економічніша.

На практиці ітераційні алгоритми використовуються в задачах математичної фізики, для яких в матриці рівняння одразу за постановкою виділяється матриця B, що до того ж має тільки малі за величиною компоненти.

Обчислення за методом прямої рекурсії потребують двох векторів – претендентів на відповідь, що ми їх позначимо як $\left\{x_m^{(old)}\right\}$ та $\left\{x_m^{(new)}\right\}$.

Схема прямої рекурсії полягає в обчисленні компонента за компонентою всіх елементів нового претендента на відповідь

$$\begin{cases} x_1^{(new)} = x_1^{(old)} + b_{1,1}x_1^{(old)} + \ldots + b_{1,n}x_n^{(old)} \\ \vdots \\ x_n^{(new)} = x_n^{(old)} + b_{n,1}x_1^{(old)} + \ldots + b_{n,n}x_n^{(old)} \end{cases} \tag{1.69}$$

По закінченні обчислення або паралельно з ним, це вже несуттєво, обчислюється сума квадратів відхилень нового та старого претендентів на розв'язки

$$err^2 = \left(x_1^{(new)} - x_1^{(old)}\right)^2 + \ldots + \left(x_n^{(new)} - x_n^{(old)}\right)^2 \tag{1.70}$$

і за величиною цієї суми робиться висновок про необхідність продовження ітерацій. Наостанок в окремому циклі значення компонент нового претендента на розв'язок передаються старому і тільки на цьому закінчується один крок ітерації.

Схема Зейделя використовує один масив, тому на кожному кроці вираховується тимчасова величина, що використовується і для оцінки точності чергового претендента на розв'язок і для заміни однієї компоненти розв'язку новою

$$\begin{cases} tmp = b_{1,1}x_1 + \ldots + b_{1,n}x_n; err <= err + tmp^2; x_1 <= x_1 + tmp; \\ \vdots \\ tmp = b_{n,1}x_1 + \ldots + b_{n,n}x_n; err <= err + tmp^2; x_n <= x_n + tmp; \end{cases} \tag{1.71}$$

Відповідна цій схемі рекурсивна формула повинна була б мати вигляд

$$x_m^{(k+1)} = \sum_{p=1}^{m-1} b_{m,p}x_p^{(k+1)} + \sum_{p=m}^{n} b_{m,p}x_p^{(k)} \tag{1.72}$$

Програмна реалізація обох рекурсивних схем принципових труднощів не доставляє.

Розділ 2

Аналіз

2.1 Особливості комп'ютерного аналізу

Специфіка комп'ютерного аналізу функцій полягає в неможливості абстрактного аналізу довільних функцій. Комп'ютерні розрахунки завжди є конкретними, застосовними до певних конкретних чисел, що мають відбивати властивості досліджуваних функцій. В деяких поодиноких випадках для означення досліджуваної функції можна застосовувати конкретні алгоритми (степеневі, тригонометричні функції тощо), але найчастіше використовуються наближені подання функцій. Так чи інакше функція в комп'ютерному поданні зображається у вигляді певної послідовності чисел, що її можна тлумачити в різні способи.

Окремою частиною комп'ютерного аналізу є програмна реалізація звичайних правил математичного аналізу, таких як правила обчислення похідних чи перетворення інтегралів. Відповідні програмні засоби просто виконують з меншою, ніж в людському виконанні, ймовірністю похибки ті ж самі дії, що їх виконувала б людина.

Найчастіше для комп'ютерного зображення функції використовується таблиця її значень. Ця таблиця з необхідністю має скінченний розмір, а в скінченній таблиці вже просто неможливо відтворити значення функції на всій множині дійсних чисел, оскільки кількість дійсних чисел суттєво більша за скінченну.

Інший спосіб – розкладення досліджуваної функції в ряд по інших, добре відомих функціях. Прикладом може бути подання функції її рядом Тейлора. Такі розкладення майже завжди потребують нескінченної кількості членів, але досить часто можна досягти задовільної точності зображення функції, обмежившись потрібною для цієї точності обмеженою кількістю членів ряду.

Що один, що другий спосіб завжди є тільки наближеними зображеннями функції, якщо під функцією розуміти звичайну дійсну функцію дійсної змінної.

Табличне подання функції можна тлумачити і як точне, але потрібно замінити стандартне уявлення про функцію, як таку, що кожному дійсному значенню з усієї множини дійсних чисел ставить у відповідність певне значення. Обмеженням на область означення функції приділяється певна увага і в звичайному математичному аналізі. Найпростішим прикладом може бути така функція, як обернена залежність $f(x) = \frac{1}{x}$. Для неї припустимою областю означення є майже вся числова вісь, крім однієї точки – $x = 0$.

Табличне ж означення функції буде точним тільки в тому разі, якщо функція означена тільки в тих точках, що присутні в табличці і зовсім не означена в усіх інших. Більш формально, областю означення такої функції є скінченна підмножина точок числової вісі. Для такої функції взагалі не означене уявлення про неперервність, відсутні такі характеристики, як диференціал або похідна, первісна.

Таким чином, використовувані в комп'ютерному аналізі функції якісно відрізняються від таких, що застосовуються в звичайному математичному аналізі.

Функцією дискретної змінної називається правило, за яким для кожної точки з дискретної множини ставиться у відповідність певне число.

Особливістю функцій дискретної змінної є відсутність звичайних для математичного аналізу уявлень про граничний перехід по точках області визначення. Оскільки кількість точок області визначення скінченна, заміною граничного переходу може бути повний перебір всіх точок, але така процедура не дозволяє робити висновки про властивості функції в малому околі (сам окіл не означений). Певною заміною похідної може бути різниця значень функції в сусідніх точках, заміною інтегралу – сума значень по певній кількості точок.

Використання комп'ютерного аналізу для задач звичайного математичного аналізу потребує побудови певних відповідностей між звичайними функціями та функціями дискретної змінної.

Побудувати **відображення звичайної функції $f(x)$ функцією дискретної змінної** досить просто – достатньо в області означення задати скінченну мережу $\{x_0, x_1, \ldots, x_N\}$ і розрахувати значення функції в кожному вузлі мережі $\{f_0 = f(x_0), f_1 = f(x_1), \ldots, f_N = f(x_N)\}$. Послідовність пар $\{x_k, f_k; k = 0 \ldots N\}$ і буде функцією дискретної змінної, що відображує задану функцію неперервної змінної.

Зворотне відображення є не настільки простим. Якщо задана функція дискретної змінної, тобто множина пар $\{x_k, f_k; k = 0 \ldots N\}$ - значень функції f_k в вузлах мережі x_k, їй може відповідати не одна, а багато різних функцій неперервної змінної.

На малюнку зображено графік функції деякої дискретної змінної - послідовність точок, координати яких даються парами значень (x_k, f_k) з таблички функції, та два графіки функцій неперервної змінної, що проходять тими самими точками – вузлами, але відрізняються в усіх інших точках координатної площини. Цей приклад наочно демонструє найсуттєвішу **проблему обчислювального аналізу** – неоднозначність відображення функцій дискретної змінної функціями неперервної змінної.

Степінь неоднозначності можна з'ясувати, якщо взяти до уваги, що існує досить багато функцій, що обертаються на нуль в усіх вузлах заданої мережі. Припустимо, що $\tilde{f}(x)$ - неперервна всюди функція, що відображує задану функцію дискретної змінної $\{x_k, f_k; k = 0 \dots N\}$ в тому розумінні, що вона співпадає з нею в усіх вузлах $\tilde{f}(x_k) = f_k; k = 0 \dots N$. Тоді для довільної неперервної функції $g(x)$ вираз $\Rightarrow f(x) = \tilde{f}(x) + g(x) \cdot (x - x_0)(x - x_1) \dots (x - x_N)$ також буде неперервною функцією і буде відображувати ту ж саму функцію дискретної змінної. Дійсно, в кожному вузлі за рахунок відповідного множника $(x - x_k)$ додатковий член обертається на нуль і значення обох функцій в вузлах співпадають (а в усіх точках між вузлами можуть відрізнятись на яку завгодно величину).

Таким чином, неперервних функцій неперервної змінної, що відображують задану функцію дискретної змінної стільки, скільки існує різних неперервних функцій $g(x)$.

Так само, якщо обмежитись диференційовними функціями, відображень буде співпадати с кількістю всіх диференційовних функцій. Навіть якщо обмежуватись аналітичними функціями, кількість різних аналітичних відображень заданої функції дискретної змінної співпадатиме з кількістю всіх аналітичних функцій.

Причиною проблеми є занадто мала кількість інформації, що закладена в функцію дискретної змінної, порівняно з функціями неперервної змінної. Функції неперервної змінної розрізняються своїми значеннями в усіх точках числової вісі. Навіть якщо мова йде про аналітичні функції, вони розрізняються значеннями коефіцієнтів ряду Тейлора, а таких для довільної функції – нескінченна (рахівна) послідовність. Функція ж дискретної змінної задається скінченною множиною значень. За їх допомогою можна визначити тільки скінченну кількість коефіцієнтів ряду Тейлора, але якщо від нескінченної величини відняти скінченну, залишається все одно нескінченна величина.

Однозначності відображення функції дискретної змінної функціями неперервної змінної можна досягти тільки в тому разі, якщо штучно обмежити множину функцій, серед яких буде шукатись відображення. Такі **штучно обмежені множини функцій називають класами**. Обмежуючись певним класом функцій, можна поставити задачу інтерполяції – задачу про побудову функції заданого класу, значення якої в усіх вузлах співпадають із значеннями заданої функції дискретної змінної.

Сама постановка задачі інтерполяції накладає певні обмеження на властивості припустимих класів функцій. Зрозуміло, що кількість різних функцій в припустимому класі не повинна перевищувати кількості можливих функцій дискретної змінної. Остання може бути оцінена, як векторний простір розмірності $N + 1$, оскільки для заданої мережі функції можуть розрізнятися значеннями в кожному з вузлів. Таким чином, **клас функцій**, в якому розв'язується задача інтерполяції, повинен бути **ізоморфним векторному просторові** відповідної розмірності.

Використовуючи той чи інший клас функцій, можна отримувати рі-

зні розв'язки задачі інтерполяції, тому важливо звертати увагу на відповідність обраного класу розв'язуваній задачі, але не в математичному, а власне в фізичному сенсі.

Другою важливою специфічною рисою комп'ютерного аналізу є присутність **похибки** в поданні чисел і, відповідно, в **результатах обчислень**. Ця похибка робить недоцільним точне відображення комп'ютерних функцій дискретної змінної функціями неперервної змінної. Похибка може виникати і з інших причин, в першу чергу, як наслідок вимірювальних похибок. В тих задачах, де похибка в значеннях функції дискретної змінної є помітною, розв'язки задачі інтерполяції втрачає сенс, більш корисним є використання **апроксимації – наближення функції дискретної змінної певною функцією неперервної змінної**.

В фізичних дослідженнях результатами різноманітних експериментів найчастіше є табличка виміряних значень, що характеризують взаємозв'язок двох (або більшої кількості) вимірюваних величин. Така табличка в математичному тлумаченні є означенням деякої функції дискретної змінної. Оскільки вимірювання завжди здійснюються з певною точністю, задача про побудову відповідної функції неперервної змінної є задачею апроксимації. Клас функцій, в якому шукається розв'язок задачі апроксимації, визначається теоретичними відомостями про досліджуване явище, а точність апроксимації визначається точністю вимірювальних приладів.

Задача апроксимації відрізняється від задачі інтерполяції, в першу чергу, тип, що *розв'язок шукається в класі функцій, вужчому за потрібний для інтерполяції.* За такого обмеження кількості вільних параметрів апроксимуючої функції не вистачає для розв'язку задачі інтерполяції – жодна з функцій заданого класу не може задовольнити умови інтерполяції. Вибір апроксимуючої функції здійснюється за певним критерієм найкращої апроксимації. Сам *критерій є додатковою характеристикою задачі апроксимації,* вибір його не входить в задачу, а здійснюється з міркувань нематематичного змісту.

Найпопулярнішим критерієм в задачах апроксимації, принаймні серед фізиків, є **критерій найменших квадратів**. За цим критерієм характеристикою якості апроксимації є вектор похибок – різниць значень апроксимуючої функції та функції дискретної змінної в кожному вузлі $\{\varepsilon_k = f(x_k) - f_k; k = 0 \dots N\}$, а мірою апроксимації – довжина цього вектору

$$\Phi[f(x)] = \sum_{k=0}^{N} (f(x_k) - f_k)^2$$

З математичної точки зору така сума квадратів вже не є функцією, оскільки вона означена не на просторі чисел або векторному просторі, а на просторі функцій. **Таке відображення функції в число має назву „функціонал",** тож цю суму називають функціоналом похибки апроксимації.

Після означення функціоналу похибки цілком слушною є постановка задачі про знаходження такої апроксимуючої функції, що надає мінімум

цьому функціоналу. Методи, що розв'язують задачу апроксимації, як задачу мінімізації наведеного функціоналу похибки, отримали назву методів найменших квадратів.

Третьою специфічною задачею комп'ютерного аналізу є задача обчислювального інтегрування. Добре відомо, що переважна більшість інтегралів не обчислюється явно. Всі чи майже всі інтеграли, що для них можна записати явний вираз в елементарних функціях, перераховані в довідниках і зібрані в пакетах аналітичних обчислень. Для певної частини інтегралів, що досить часто використовуються, існують вирази через спеціальні функції, що саме й означалися через подібні інтегральні подання. Переважна ж більшість інтегралів не вираховується а ні в елементарних, а ні в спеціальних функціях. Для таких інтегралів і застосовуються методи обчислювального інтегрування.

Центральною *ідеєю всіх обчислювальних методів інтегрування є заміна інтеграла сумою значень інтегровної функції*, обчислених в окремих точках проміжку інтегрування. Власне заміна функції набором її значень в окремих точках є перетворенням функції неперервної змінної на функцію дискретної змінної. Подальше обчислення інтеграла можливе тільки в тому разі, якщо здійснюється зворотне перетворення на функцію неперервної змінної, тобто інтерполяція або апроксимація отриманої функції дискретної змінної. Зрозуміло, що використання апроксимації не є аж надто доцільним, оскільки втрачається частина інформації про отримані вже значення функції дискретної змінної. Тому використовується та чи інша інтерполяція, а інтерполююча функція може бути обрана з класу функцій, що їх можна інтегрувати явно.

У цієї ідеї можна явно виділити два місця, де замість точних методів використовується не повністю обґрунтований вибір варіантів – вибір мережі, тобто набору вузлів, в яких обчислюються значення функції, та вибір класу функцій, в якому будується інтерполююча функція.

Ще однією суттєвою проблемою комп'ютерного аналізу є питання про можливість обчислення аналогів похідних функції дискретної змінної.

В задачах аналізу результатів експериментальних досліджень досить часто метою вимірювань є визначення не самої вимірюваної величини, а її першої або навіть другої похідної. За приклад можна навести добре відому машину Атвуда, доповнену будь-яким приладом поточної реєстрації координат. Результатом вимірювання є послідовність пар „координата – момент часу", що утворює функцію дискретної змінної. Фізично змістовною величиною є не сама ця залежність, а тільки одна її характеристика, а саме прискорення, що означається, як друга похідна координати. Неузгодженість результату вимірювання та теоретичної моделі полягає в невідповідності математичних моделей теорії та експерименту, адже теорія використовує припущення про те, що залежність координати від часу є функцією неперервної змінної, тоді як результатом вимірювань є функція дискретної змінної. Тому порівнянню результатів вимірювання з теорією повинне передувати перетворення функції дискретної змінної на функцію неперервної – апроксимація результатів вимірювань. Одним з параметрів апроксимую-

чої функції і буде величина прискорення.

2.2 Поліноміальна інтерполяція.

Історично задача інтерполяції виникла, як задача поповнення таблички значень функції, що її важко обчислити. В докомп'ютерну еру великою популярністю користувались таблиці Брадіса, що вміщували значення тригонометричних функцій, логарифмів та експонент з точністю до 7 цифр. Значення функції для аргументу, що не співпадає з табличним, розраховувались за допомогою інтерполяції. Найчастіше використовувалась лінійна інтерполяція, але для більш точних розрахунків в таблицях наводились ще й додаткові коефіцієнти для квадратичної інтерполяції.

Узагальненням такої інтерполяції є інтерполяція в класі поліномів певного степеню.

В цьому класі **параметрами інтерполюючої функції є коефіцієнти поліному**, що їх рівно на один більше, ніж степінь полінома. Оскільки параметрів інтерполяції повинно бути принаймні не більше (і не менше), ніж вузлів інтерпольовної функції, степінь полінома має бути на одиницю менша за кількість вузлів.

Отже, задача поліноміальної інтерполяції ставиться в такий спосіб:

Задано таблицю значень функції дискретної змінної $\{x_k, f_k; k = 0 \dots N\}$. Необхідно для деякої точки $x_0 < x < x_N$, що не співпадає з жодним з вузлів $x \neq x_k; k = 1 \dots N-1$, знайти значення полінома $P_N(x)$ такого, що його значення в кожному з вузлів співпадає із значенням заданої функції $P_N(x_k) = f_k; k = 0 \dots N$.

Розглянемо в першу чергу питання про **існування розв'язку** цієї задачі. Обчислення значення поліному можливе, якщо відомі його коефіцієнти, тобто задача, явно або неявно, зводиться до визначення коефіцієнтів полінома. Поліном степеню N має $N+1$ коефіцієнт, тому умови інтерполяції – рівняння $P_N(x_k) = f_k; k = 0 \dots N$ присутні в такій самій кількості, як і невідомі і задача може мати розв'язок, хіба що рівняння будуть несумісними. Тому є сенс досліджувати систему рівнянь інтерполяції на сумісність. Запишемо цю систему. Позначимо через $a_0, a_1, \dots a_N$ коефіцієнти полінома при нульовому, першому і так далі ступені аргументу, тоді поліном можна подати у вигляді

$$P_N(x; a_0, \dots, a_N) = a_0 + a_1 x + \dots + a_N x^N \tag{2.1}$$

(тут явно підкреслено залежність значень поліному від коефіцієнтів), а рівняння інтерполяції перетворюються на систему

$$\begin{cases} a_0 + a_1 x_0 + \dots + a_N x_0^N = f_0 \\ \dots \\ a_0 + a_1 x_N + \dots + a_N x_N^N = f_N \end{cases} \tag{2.2}$$

$N+1$ лінійних рівнянь. Ця система є лінійною неоднорідною і має або єдиний розв'язок, або жодного або безліч залежно від величини визначника матриці системи. Випишемо цю матрицю

$$M = \begin{pmatrix} 1 & x_0 & \dots & x_0^N \\ 1 & x_1 & \dots & x_1^N \\ \vdots & \ddots & \ddots & \vdots \\ 1 & x_N & \dots & x_N^N \end{pmatrix} \tag{2.3}$$

Легко бачити, що цей визначник є поліномом степеню N відносно кожної із координат вузлів, наприклад, відносно x_0 . Такий поліном має не більше ніж N нулів. Знайдемо ці нулі. Якщо x_0 дорівнює координаті одного з інших вузлів, наприклад, x_k, два рядки визначника співпадають і він дорівнює нулю. Бачимо, що визначник дорівнюватиме нулю, якщо співпадають координати двох (або більше) вузлів і відрізняється від нуля, якщо всі вони різні.

Якщо координати двох вузлів співпадають, система рівнянь інтерполяції буде несумісною або сумісною залежно від того, співпадають або не співпадають значення функції в цих вузлах. Якщо не співпадають, розв'язку не існує (функція не може мати два значення в одній точці), якщо співпадають, в таблиці значень просто продубльовано один рядок і степінь поліному є завищеним – задача інтерполяції розв'язується неоднозначно.

На підсумок, маємо таке твердження щодо розв'язуваності задачі поліноміальної інтерполяції:

Задача поліноміальної інтерполяції має єдиний розв'язок в класі поліномів степеню, на одиницю меншого за кількість вузлів.

Метод Ньютона.

Практичні методи поліноміальної інтерполяції розраховані на обчислення значення в одній точці, тому в них робиться наголос на обчисленні без розрахунку коефіцієнтів інтерполяційного полінома.

Метод Ньютона є найпростішим в ручному виконанні. Він полягає в поступовому підвищенні степеню інтерполяції. За початкову обирається лінійна інтерполяція за значеннями в двох вузлах, найближчих до точки обчислення. Якщо надати цим вузлам номери 0 та 1, маємо очевидний вираз для лінійної інтерполяції

$$f_1(x) = f_0 + \frac{f_1 - f_0}{x_1 - x_0}(x - x_0) \tag{2.4}$$

Перший член цього виразу можна сприймати, як інтерполяцію поліномом нульового степеню, тоді другий є внеском від першої точки, що має значенням добуток полінома першого степеню, рівного нулю в першому вузлі $(x - x_0)$, на множник, що забезпечує задане значення в першому вузлі. Інтерполяційний поліном, що враховує вузол 2, можна побудувати в подібний спосіб, якщо до цього полінома додати вираз

$$\Delta f_2(x) = (f_2 - f_1(x_2)) \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} \tag{2.5}$$

Дріб в цьому виразі є поліномом другого степеню, що дорівнює нулю в попередніх вузлах, а за рахунок знаменника обертається на одиничку в останньому вузлі. Множник перед дробом додає до значення попереднього інтерполяційного полінома значення, що гарантує потрібне значення в другому вузлі.

Тепер можна записати рекурентний вираз для інтерполяційного полінома степеню $k+1$ через розраховане попередньо значення інтерполяційного полінома степеню k

$$f_{k+1}(x) = f_k(x) + (f_{k+1} - f_k(x_{k+1})) \frac{(x-x_0)\dots(x-x_k)}{(x_{k+1}-x_0)\dots(x_{k+1}-x_k)} \tag{2.6}$$

Підрахуємо обчислювальну складність методу. Припустимо, що для розрахунку інтерполяційного полінома к-того степеню потрібно $D(k)$ дій. Тоді обчислення наступного полінома потребує ще стільки ж дій (значення обчисленого полінома в новому вузлі) та ще $3k$ дій для обчислення дробу, тому

$$D(k+1) = D(k) + 3k = \tfrac{3}{2}k(k-1) + D(0) \tag{2.7}$$

загальна кількість дій пропорційна другому степеню кількості вузлів.

Метод Лагранжа

Цей метод відрізняється тим, що значення інтерполяційного полінома вираховуються одразу по всіх вузлах, як сума членів, що мають потрібне значення в одному вузлі і дорівнюють нулю в усіх інших. Запишемо потрібний поліном для к-того вузла. З цією метою побудуємо в першу чергу поліном, що обертається в нуль в усіх вузлах, крім потрібного

$$P_k(x) = (x-x_0)\dots(x-x_{k-1})(x-x_{k+1})\dots(x-x_N) \tag{2.8}$$

Дійсно, кожен з множників в цьому поліномі обертається на нуль в черговому вузлі, і тільки в одному - потрібному вузлі відповідний множник відсутній. Розділимо тепер цей поліном на його ж значення в к-тому вузлі і помножимо на значення функції в цьому вузлі

$$L_k(x) = f_k \frac{(x-x_0)\dots(x-x_{k-1})(x-x_{k+1})\dots(x-x_N)}{(x_k-x_0)\dots(x_k-x_{k-1})(x_k-x_{k+1})\dots(x_k-x_N)} \tag{2.9}$$

Отриманий поліном дорівнює нулю в усіх вузлах, крім к-того, а в цьому вузлі має необхідне значення.

Інтерполяційний поліном Лагранжа утворюється, як сума побудованих поліномів для кожного з вузлів

$$L(x) = \sum_{k=0}^{N} f_k \frac{(x-x_0)\dots(x-x_{k-1})(x-x_{k+1})\dots(x-x_N)}{(x_k-x_0)\dots(x_k-x_{k-1})(x_k-x_{k+1})\dots(x_k-x_N)} \tag{2.10}$$

Головним недоліком поліноміальної інтерполяції є її непридатність для інтерполяції функцій з великою (більше десятка) кількістю вузлів. Поліноми високого степеню частіше за все досить швидко змінюються між вузлами і їх поведінка може надто сильно відрізнятись від очікуваної. Застосування поліноміальної інтерполяції в ручних обчисленнях не призводить до суттєвих помилок, оскільки потрібний степінь полінома оцінюється інтуїтивно. Для комп'ютерних обчислень інтуїтивна оцінка адекватності алгоритму неприйнятна, тому частіше використовуються інші методи. Власне кажучи, під час ручного поповнення таблиці з інтерполяцією, складнішою за лінійну, завжди використовувалась не вся таблиця значень, а тільки та її частина, що безпосередньо оточує точку, для якої розраховується значення. В інших точках, що належать до іншого проміжку між вузлами, інтерполяція здійснювалась по іншій підмножині таблиці значень. Практично замість поліноміальної завжди використовувалась і використовується зараз кусково–поліноміальна інтерполяція, за якою на різних відрізках застосовуються різні поліноми.

Програмна реалізація

версія Pascal

```
program lagrange;
  var
    sx,sy :Array[0..10000] of Double;
    N     :Integer;
function Getlag(x:Double):Double;
   var
     i,ii     :Integer;
     res,temp :Double;
  begin
       for i:=0 to N
       do if x=sx[i]
          then begin Getlag:=sy[i];Exit;end;
       res:=0;
       for i:=0 to N
       do begin
          temp:=sy[i];
          for ii:=0 to i-1
          do temp:=temp*(x-sx[ii])/(sx[i]-sx[ii]);
          for ii:=i+1 to N
          do temp:=temp*(x-sx[ii])/(sx[i]-sx[ii]);
          res:=res+temp;
          end;
       Getlag:=res;
  end;
```

```
procedure GetData;
  var
     infile :Text;
  const
     name='data.txt';
  begin
     N:=-1;
     assign(infile,name);
     reset(infile);
     while not Eof(infile)
     do begin
        Inc(N);ReadLn(infile, sx[N] ,sy[N]);
        end;
     close(infile);
     if N<=0
     then begin
          WriteLn('Errors in input file');
          Halt;
          end;
  end;

procedure PutRes;
  var
    outfile :Text;
    i,ii    :Integer;
    x,h     :Double;
  const
    name='res.txt';
  begin
      assign(outfile,name);
      rewrite(outfile);
      Writeln(outfile,'Interpolated values');
      for i:=0 to N-1
      do begin h:=(sx[i+1]-sx[i])/5;
         for ii:=0 to 4
         do begin
            x:= sx[i]+ii*h;
            WriteLn(outfile,x:16:6,' ',Getlag(x):22:8);
            end;
         end;
       WriteLn(outfile,sx[N]:16:6,' ',Getlag(sx[N]):22:8);
       close(outfile);
   end;

BEGIN
   GetData;
```

```
   PutRes;
END.
```

В програмі передбачено, що вхідні дані задачі зачитуються з файлу, що має назву 'data.txt' і знаходиться в тій же директорії, що й сама програма. Результат роботи програми записується в інший файл – з назвою 'res.txt'. Вхідний файл не перевіряється на можливі помилки в організації даних, тобто вважається, що в кожному рядку записані послідовно два числа – значення аргументу та функції чергового вузла інтерполяції.

Вхідні дані зачитуються в два масиви, а спеціальна змінна N вираховує кількість зачитаних рядків. Далі програма використовує саме це значення для фактичного розміру масиву.

версія C++

```
#include <stdio.h>
#include <iostream.h>
#define bufsize 1000
  double ax[bufsize];
  double ay[bufsize];
  int n;

double value(double arg){
   double res=0;
   double tmp;
   for(int k=0;k<n;k++){
   tmp=ay[k];
   for(int kk=0;kk<n;kk++){
    if(kk!=k){tmp*=(arg-ax[kk])/(ax[k]-ax[kk]);};
   };
   res+=tmp;
   };
   return res;
};//value

int main(){
  for(n=0;(n<bufsize)&&(cin>>ax[n]>>ay[n]);n++);
  double step=(ax[n-1]-ax[0])/n/10;
  for(double x=ax[0];x<(ax[n-1]+step);x+=step){
  cout<<x<<"\t"<<value(x)<<"\n";};
  return 0;
}
```

Мова C++ дозволяє суттєво скорочувати текст програми і не тільки за рахунок скороченого синтаксису. Адресація вхідних та вихідних даних в цій

програмі здійснюється не відкриттям відповідних файлів, а переспрямовуванням вхідних та вихідних потоків

```
lagrange.exe < data.txt >res.txt
```

в командному рядку.

Ще одна версія програмної реалізації – оболонка Maple, що поєднує засоби не тільки для аналітичного та звичайного програмування, а й для графічного відтворення результатів

```
> restart:with(plots):
> path:="C:\\Documents and Settings\\const.US\\My
> Documents\\work\\tutor\\samples\\maple\\lagr\\":
> dataname:="data.txt":resname:="res.txt":
> source:=cat(path,dataname):target:=cat(path,resname):

Warning, the name changecoords has been redefined
```

версія Maple

підготовка даних про файл з параметрами та файл результатів

```
> dat:=readdata(dataname,2);nmax:=nops(dat);
```

$$dat := [[0., 10.], [1., 9.], [3., 7.], [5., 5.], [8., 2.], [10., 0.]]$$

$$nmax := 6$$

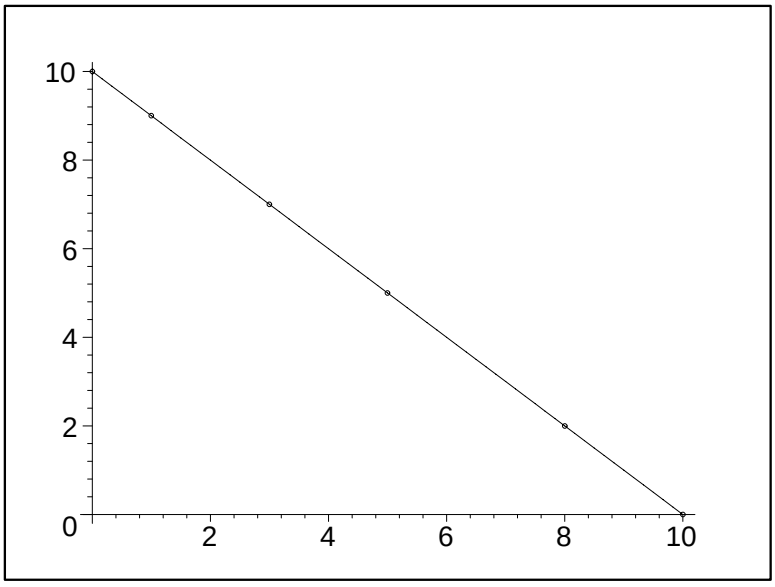


Рис. 2.1: Графік розв'язку задачі інтерполяції з лінійною пропорційністю

Зачитування даних з файлу. Кожен рядок містить дві координати чергового вузла.

0	1	3	5	8	10
10	9	7	5	2	0

Табл. 2.1: Інтерполяція з лінійною пропорційністю

```
> lagr:=proc(x) local tmp,res,k,kk;global
> dat,nmax; res:=0; for k from 1 to nmax #початок зовнішнього циклу
> do
> tmp:=dat[k][2]; # значення функції в черговому вузлі
> for kk from 1 to k-1 # обчислення
> першої частини поліному do
> tmp:=tmp*(x-dat[kk][1])/(dat[k][1]-dat[kk][1]); od;
> for kk from k+1 to nmax # обчислення другої частини поліному
> do tmp:=tmp*(x-dat[kk][1])/(dat[k][1]-dat[kk][1]); od;
> res:=res+tmp; # значення чергового поліному додається до
> результату od;# закінчення зовнішнього циклу RETURN(res);
> end:
```

```
> points:=[]:xmin:=dat[1][1]:xmax:=dat[nmax][1]:
> numpoints:=nmax*10.0;step:=(xmax-xmin)/numpoints: # підготовка
> циклу обчислення значень функції # в усіх точках з координатами
> $x_k= xmin+k step$ # points накопичує координати точок [current,y]
> current:=xmin: while xmax>current do current:=current+step;
> y:=lagr(current);points:=[op(points),[current,y]];
> od:
```

$$numpoints := 60.0$$

```
> PLOT(CURVES(points),POINTS(op(dat),SYMBOL(CIRCLE)));
> #CURVES будує лінію за точками з масиву points #POINTS будує
> окремі точки послідовності op(dat)
> #SYMBOL(CIRCLE) задає форму відтворення точок
> writedata(target,points);#запис у файл
```

0	1	3	5	8	10
0	9	21	25	16	0

Табл. 2.2: Інтерполяція для квадратичної залежності

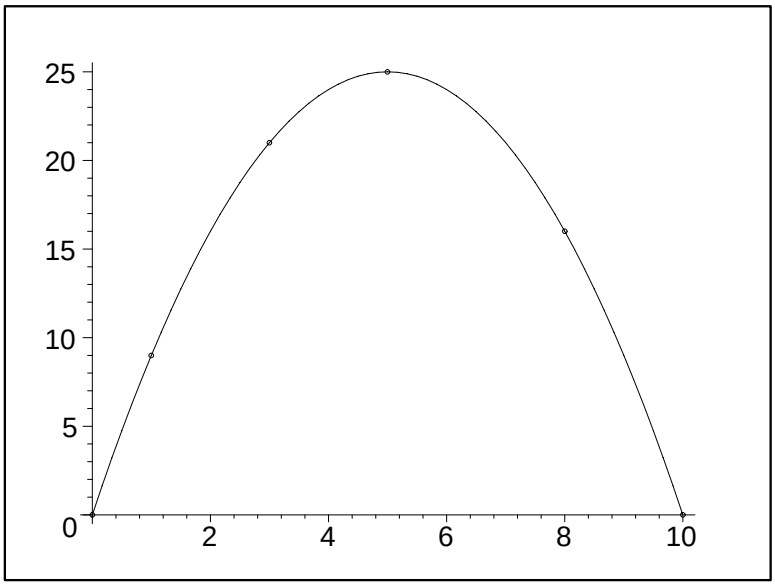


Рис. 2.2: Графік розв'язку задачі інтерполяції для квадратичної залежності

2.3 Сплайн-інтерполяція

Головним недоліком поліноміальної інтерполяції є її непридатність для інтерполяції функцій з великою (більше десятка) кількістю вузлів. Поліноми високого степеню частіше за все досить швидко змінюються між вузлами і їх поведінка може надто сильно відрізнятись від очікуваної. Застосування поліноміальної інтерполяції в ручних обчисленнях не призводить до суттєвих помилок, оскільки потрібний степінь полінома оцінюється інтуїтивно. Для комп'ютерних обчислень інтуїтивна оцінка адекватності алгоритму неприйнятна, тому частіше використовуються інші методи. Власне кажучи, під час ручного поповнення таблиці з інтерполяцією, складнішою за лінійну, завжди використовувалась не вся таблиця значень, а тільки та її частина, що безпосередньо оточує точку, для якої розраховується значення. В інших точках, що належать до іншого проміжку між вузлами, інтерполяція здійснювалась по іншій підмножині таблиці значень. Практично замість поліноміальної завжди використовувалась і використовується зараз кусково–поліноміальна інтерполяція, за якою на різних відрізках застосовуються різні поліноми.

Уявлення про сплайни виникло в зв'язку з потребами комп'ютерної обробки геометричних об'єктів – в першу чергу ліній. Сплайни активно використовуються не тільки в задачах комп'ютерного моделювання, а й безпосередньо в алгоритмах побудови комп'ютерного зображення, наприклад, під час перебудови графічного зображення з малої роздільної здатності (такої, як екранна) на велику, властиву для принтерів. В цих задачах визначальною рисою є велика кількість вузлів, за якими повинна відбуватись інтерполяція. Типовим прикладом є побудова літер в процесі підготовки документу до друку. Зображення літер звичайно будується за певними алгоритмами з не такою вже й великою кількістю інформації про кожну окрему літеру – файли з шрифтами (фонти) звичайно вміщують тільки інформацію про деяку кількість точок контуру. Звичайне ж зображення літери на папері може сягати 300 – 1000 точок висоти і майже стільки ж – ширини. Під час підготовки до друку ключові точки контуру доповнюються до повного набору точок зображення літери і в процесі доповнення використовуються алгоритми, що побудовані саме на підставі розв'язування задачі інтерполяції.

Суттєвою особливістю такого типу задач є необхідність побудови повністю програмованих алгоритмів, що не передбачають втручання людини. Поліноміальна інтерполяція для таких задач є повністю неприйнятною (важко уявити собі властивості полінома, степінь якого сягає декількох сот), а для кусково-поліноміальної інтерполяції потрібен алгоритм узгодження поліномів в точках з'єднання.

Під сплайном в широкому розумінні цього терміну розуміється функція, для якої область означення поділена на послідовність відрізків. На кожному відрізку має місце свій алгоритм обчислення значень функції, а на межі, що з'єднує/роз'єднує кожні два сусідні відрізки, задані умови спряження. Найчастіше алгоритм обчислення сплайну на кожному проміжку є поліноміальним, а умовами спряження є неперервність в кожному вузлі самого сплайну і декількох (але не всіх) похідних.

Найпростішим прикладом є лінійний неперервний сплайн. Він є лінійною функцією, тобто поліномом першого степеня, на кожному відрізку, і є неперервним в вузлах. Графік такого сплайну між кожними двома вузлами є відрізком прямої, а в цілому утворює ламану, що проходить через вузли сплайну.

Лінійний сплайн характеризується, на кожному відрізку, двома параметрами

– коефіцієнтами лінійної залежності

$$y(x) = a_k + b_k x;\ x_k \leqslant x \leqslant x_{k+1}. \tag{2.11}$$

Загальна кількість таких коефіцієнтів вдвічі перевищує кількість проміжків, тобто для функції з $N+1$ вузлами дорівнює $2N$.

Задача інтерполяції лінійними сплайнами полягає в находженні всіх коефіцієнтів на підставі таких умов:

Умова інтерполяції

$$s(x_k) = f_k; \forall k = 0 \ldots N \tag{2.12}$$

(графік сплайну проходить повз всі вузли);

Умова неперервності

$$s(x_k - 0) = s(x_k + 0); \forall k = 1 \ldots N - 1 \tag{2.13}$$

(графік сплайну є неперервним в усіх внутрішніх вузлах).

Кількість умов інтерполяції ($N+1$ рівняння) та неперервності ($N-1$ рівняння) співпадає з кількістю невідомих коефіцієнтів і можна сподіватись, що розв'язок задачі існує.

Дослідження питання про існування розв'язку можна здійснити прямою побудовою розв'язку. Розглянемо умови інтерполяції на двох кінцях якого-небудь відрізка з номером m. Для внутрішніх точок цього відрізка сплайн є лінійною функцією

$$y(x) = a_m + b_m x;\ x_m \leqslant x \leqslant x_{m+1} \tag{2.14}$$

і за неперервністю на кінцях відрізка має значення

$$y(x_m) = a_m + b_m x_m = f_m; y(x_{m+1}) = a_m + b_m x_{m+1} = f_{m+1}. \tag{2.15}$$

Ці два рівняння дозволяють повністю визначити коефіцієнти сплайну

$$a_m = \frac{f_m x_{m+1} - f_{m+1} x_m}{x_{m+1} - x_m}, b_m = \frac{f_{m+1} - f_m}{x_{m+1} - x_m}, \tag{2.16}$$

або

$$y(x) = y_m + (y_{m+1} - y_m) \frac{x - x_m}{x_{m+1} - x_m};\ x_m \leqslant x \leqslant x_{m+1} \tag{2.17}$$

тобто задача завжди має розв'язок, якщо тільки можна виконувати всі дії в цих виразах, тобто коли жоден із знаменників не дорівнює нулю. Таким чином, задача інтерполяції лінійними сплайнами завжди має єдиний розв'язок, якщо аргументи сусідніх вузлів не співпадають.

Умова неперервності виконується автоматично.

2.3.1 Кубічні сплайни

Найчастіше на практиці використовуються кубічні сплайни, що на кожному відрізку означаються, як поліноми третього степеню. Для них умова неперервності самих сплайнів доповнюється умовами неперервності першої та другої похідних. Перевагою кубічних сплайнів є більша гладкість графіків функції порівняно з лінійними.

Неперервності двох похідних вистачає для пересічної більшості задач інтерполяції.

Кубічні сплайни потребують вже не двох, а чотирьох параметрів на кожному проміжку інтерполяції, оскільки довільний поліном третього степеню має чотири коефіцієнти. Загальним виразом для кубічного сплайну є кусочно-поліноміальна функція

$$s(x) = \left\{ a_k + b_k x + c_k x^2 + d_k x^3;\ x_k \leqslant x \leqslant x_{k+1} \right\}, \tag{2.18}$$

що має $4N$ коефіцієнтів. Відповідно повністю визначити сплайн можна тільки тоді, коли відомі всі ці $4N$ коефіцієнтів. Наприклад, для сплайну на 1000 вузлів потрібно задати 4000 значень коефіцієнтів.

Найчастіше кількість вузлів, звичайно, значно менша, але й для 100 вузлів вже потрібно 400 коефіцієнтів, а для них потрібно задати принаймні стільки ж $4N$ рівнянь, а потім їх розв'язати і вже тільки після цього можна вираховувати значення сплайну.

Рівняння інтерполяції

$$s(x_k) = f_k;\ k = 0 \ldots N \tag{2.19}$$

разом з умовами неперервності

$$\begin{cases} s(x_k + 0) = s(x_k - 0) \\ s'(x_k + 0) = s'(x_k - 0) \\ s''(x_k + 0) = s''(x_k - 0) \end{cases};\ k = 1 \ldots N - 1 \tag{2.20}$$

утворюють систему з

$$(N + 1) + 3(N - 1) = 4N - 2 \tag{2.21}$$

рівнянь, тобто не вистачає двох рівнянь. Ці рівняння – додаткові умови задачі інтерполяції, можуть задаватись в різний спосіб залежно від вимог конкретної задачі. Найчастіше використовується умова обернення на нуль других похідних на кінцях області означення (стандартна умова)

$$\begin{cases} s''(x_0) = 0 \\ s''(x_N) = 0 \end{cases} \tag{2.22}$$

а для функцій із замкненою областю означення (функції на колі) –умова циклічності

$$\begin{cases} x_0 = x_N \\ s(x_0) = s(x_N) \\ s'(x_0) = s'(x_N) \\ s''(x_0) = s''(x_N) \end{cases} \tag{2.23}$$

Перша з умов циклічності насправді є умовою на циклічність інтерпольованої функції, тож додаткових умов знов лишається тільки дві.

Так само, як і для лінійних сплайнів, розв'язок задачі інтерполяції кубічними сплайнами існує та єдиний, якщо тільки координати всіх вузлів є різними.

Отже, задача інтерполяції кубічними сплайнами потребує знаходження розв'язку системи $4N$ лінійних рівнянь. Найпрямолінійнішим, але й найнеефективнішим методом для її розв'язування є, мабуть, метод Гауса, що вимагатиме $64N^3$ обчислень – суттєво більше, ніж для побудови інтерполяції суцільним поліномом (за методом Лагранжа необхідно $\sim N^2$ обчислень). Матриця системи рівнянь, що їх необхідно розв'язувати для обчислення сплайну, містить велику кількість нулів

(про такі матриці кажуть, що вони є розрідженими). Кожне з рівнянь неперервності вміщує не більше, як 8 коефіцієнтів поліномів з двох сусідніх проміжків, тільки що різні рівняння пов'язують між собою коефіцієнти з різних пар сусідніх проміжків. Використання методу Гауса для розріджених матриць вкрай неефективне, оскільки значною частиною дій буде множення на нуль або віднімання нуля.

Структура рівнянь, що задають зв'язок між сусідніми наборами коефіцієнтів, нагадує ті, що зустрічаються в методі прогонки і тому можна очікувати, що повинні існувати методи розв'язування, що матимуть лінійну складність.

2.3.2 Базові сплайни

З метою пошуку ефективного методу побудови інтерполяційного сплайна розіб'ємо задачу на дві підзадачі – задачу задовільнення системи умов неперервності та власне задачу інтерполяції. Таке розбиття означає, що ми поділяємо всю систему рівнянь на дві підсистеми в такий спосіб, що одну з них можна розв'язати незалежно від того, яким має бути розв'язок іншої. Тоді в процесі пошуку розв'язку другої ми можемо використати вже відомий розв'язок першої підсистеми.

Метод базових сплайнів ґрунтується на виділенні в першу підсистему умов неперервності, а в другу – власне рівнянь інтерполяції.

Розглянемо спочатку, виключно з пояснювальною метою, лінійні сплайни.

Припустимо, що ми можемо побудувати систему базових сплайнів, тобто таких, що кожен з них дорівнює нулю майже в усіх вузлах, крім одного, тобто існує система сплайнів така, що

$$B_k(x_m) = \begin{cases} 1; m = k \\ 0; \forall m \neq k \end{cases} . \tag{2.24}$$

Тоді довільна лінійна комбінація цих сплайнів теж буде сплайном і буде мати такі значення в вузлах інтерполяції

$$s(x_m) = \sum_{k=0}^{N} s_k(B_k(x_m)) = s_m, \tag{2.25}$$

і система рівнянь інтерполяції перетворюється на систему незалежних рівнянь

$$s(x_m) = f_m \Rightarrow s_m = f_m \tag{2.26}$$

Доведемо, що складність задачі побудови базових сплайнів не перевищує лінійну.

Систему базових сплайнів утворюють трикутні сплайни, що дорівнюють нулю на всіх проміжках, крім двох, а саме тих, що мають на одному з кінців вузол, в якому даний конкретний базовий сплайн дорівнює одиниці. Для довільного базового сплайну, крім кінцевих, маємо такий вираз

$$B_k(x) = \begin{cases} \frac{x - x_{k-1}}{x_k - x_{k-1}}; x_{k-1} \leqslant x \leqslant x_k \\ \frac{x_{k+1} - x}{x_{k+1} - x_k}; x_k \leqslant x \leqslant x_{k+1} \\ 0; x < x_{k-1} \cup x > x_{k+1} \end{cases} . \tag{2.27}$$

Для кінцевих базових сплайнів ці вирази додатково спрощуються

$$B_0(x) = \begin{cases} \frac{x_1 - x}{x_1 - x_0}; x_0 \leqslant x \leqslant x_1 \\ 0; x > x_1 \end{cases}$$
$$B_N(x) = \begin{cases} \frac{x - x_{N-1}}{x_N - x_{N-1}}; x_{N-1} \leqslant x \leqslant x_N \\ 0; x < x_{N-1} \end{cases} . \quad (2.28)$$

Оскільки заповнення проміжків нулями не потребує спеціальних дій, для побудови кожного окремого базового сплайну потрібна стала кількість дій, що не зростає з кількістю вузлів, тобто побудова одного базового сплайну є задачею нульової складності, а побудова всієї системи базових сплайнів має лінійну складність. Відповідно, і побудова всього лінійного сплайну має лінійну складність.

2.3.3 Кубічні базові сплайни

Виявляється, що не можна побудувати систему таких кубічних базових сплайнів, що кожен з них дорівнює нулю майже в усіх вузлах, крім одного.

Дійсно, умова обернення сплайну на нуль разом з двома похідними визначить нам вже три з чотирьох коефіцієнтів сплайну. Відповідно сплайн, що обертався б на нуль, та ще й разом з першою та другою похідними в вузлах $k-1$ та $k+1$, має всього два вільних параметри, а на нього накладаються чотири умови в вузлі k - три умови неперервності та умова на одиничне значення. Протиріччя виникає вже в умові неперервності перших похідних. Дійсно, поліном третього степеню, що обертається на нуль разом з двома похідними в вузлі $k-1$, має загальний вигляд $\alpha(x - x_{k-1})^3$ і зростає зі зростанням аргументу, тож в вузлі k він має додатню похідну (якщо сам поліном додатній в цьому вузлі), а от поліном, що обертається на нуль в вузлі $k+1$, має загальний вигляд $\beta(x_{k+1} - x)^3$ і його похідна в вузлі k буде від'ємною (знов таки, якщо сам поліном додатній в цьому вузлі).

Якщо ж за базові обрати сплайни, що відрізняються від нуля не на двох тільки, а на чотирьох проміжках, кількості параметрів (4*4=16) вже достатньо, аби задовольнити всі необхідні умови, а саме: по три умови на дальніх вузлах (нуль разом з двома похідними), по три умови неперервності в трьох внутрішніх вузлах і одна умова на значення сплайну в центральному вузлі – разом 16. Тоді значення базового сплайну в двох вузлах поруч з центральним будуть вже такими, які дасть розв'язок всієї системи рівнянь неперервності і можна записати, що базовий сплайн з номером k має в кожному з вузлів мережі такі значення

$$B_k(x_m) = \begin{cases} 1; m = k \\ b_k(\pm 1) \\ 0; \forall m \neq k, k \pm 1 \end{cases} \quad (2.29)$$

Довільна лінійна комбінація базових сплайнів може бути означена за допомогою сплайн-коефіцієнтів $\{s_k; k = -1 \ldots N+1\}$, як сума

$$s(x) = \sum_{k=-1}^{N+1} s_k B_k(x). \quad (2.30)$$

Доцільно зауважити, що сплайн-коефіцієнти $\{s_k; k = -1 \ldots N+1\}$ відрізняються від коефіцієнтів сплайну, що задають вирази

$$s(x) = \left\{ a_k + b_k x + c_k x^2 + d_k x^3;\ x_k \leqslant x \leqslant x_{k+1} \right\} \quad (2.31)$$

для поліномів на кожному відрізку.

Лінійна комбінація теж буде сплайном, тобто буде функцією, що неперервна разом з двома похідними і на кожному проміжку є поліномом не вище третього степеню. Ця комбінація буде мати такі значення в вузлах інтерполяції

$$s\left(x_m\right) = \sum_{k=-1}^{N+1} s_k B_k\left(x_m\right) = s_m + b_m\left(1\right) s_{m+1} + b_m\left(-1\right) s_{m-1} \tag{2.32}$$

Система рівнянь інтерполяції перетворюється на тричленну систему

$$s\left(x_m\right) = f_m \Rightarrow s_m + b_m\left(1\right) s_{m+1} + b_m\left(-1\right) s_{m-1} = f_m \tag{2.33}$$

з тридіагональною матрицею.

Для тридіагональної системи добре відомий алгоритм лінійної складності – метод прогонки, тож виходить, що для кубічних сплайнів задача знаходження сплайн–коефіцієнтів має лінійну складність.

Оцінка повної обчислювальної складності задачі сплайн-інтерполяції методом базових сплайнів вміщує, крім оцінки складності визначення сплайн–коефіцієнтів, ще й оцінку складності побудови базових сплайнів. В загальному випадку ця складність повинна буди не менше ніж лінійною, оскільки потрібно побудувати базові сплайни в кількості, що майже співпадає з кількістю вузлів, хіба що на два більша.

Доведемо, що складність задачі побудови базових сплайнів не перевищує лінійну.

По-перше, звернімо увагу на ту обставину, що для кубічних сплайнів система рівнянь неперервності містить тільки $3N-3$ рівняння з повної системи $4N$ рівнянь, тому в розкладенні довільного сплайну по базових повинно бути $4N-3N+3=N+3$ вільних параметри і відповідно $N+3$ члени ряду. Саме з метою забезпечення потрібної кількості вільних параметрів послідовність кубічних базових сплайнів починається з члена $B_{-1}\left(x\right)$ і закінчується членом $B_{N+1}\left(x\right)$. Ці базові сплайни повинні були б мати одиничне значення в уявних вузлах мережі x_{-1},x_{N+1}. Рівняннями для сплайн-коефіцієнтів, крім $N+1$ рівнянь інтерполяції, є ще дві додаткові кінцеві умови, що разом і мають визначити $N+3$ сплайн-коефіцієнти.

Розглянемо тепер умови, за якими повинні визначатись коефіцієнти кубічних поліномів довільного базового сплайну, що має одиничне значення в одному з внутрішніх вузлів мережі. В загальному випадку такий внутрішній базовий сплайн має вигляд

$$B_k\left(x\right) = \begin{cases} a_{k,-2} + b_{k,-2}x + c_{k,-2}x^2 + d_{k,-2}x^3; x_{k-2} \leqslant x \leqslant x_{k-1} \\ a_{k,-1} + b_{k,-1}x + c_{k,-1}x^2 + d_{k,-1}x^3; x_{k-1} \leqslant x \leqslant x_k \\ a_{k,0} + b_{k,0}x + c_{k,0}x^2 + d_{k,0}x^3; x_k \leqslant x \leqslant x_{k+1} \\ a_{k,1} + b_{k,1}x + c_{k,1}x^2 + d_{k,1}x^3; x_{k+1} \leqslant x \leqslant x_{k+2} \\ 0; x \leqslant x_{k-2} \cup x \geqslant x_{k+2} \end{cases}, \tag{2.34}$$

в якому є 4*4=16 коефіцієнти до 4 поліномів третього степеню. Ці поліноми визначають властивості конкретного базового сплайну на чотирьох відрізках, що прилягають до заданого вузла (по два з кожного боку). За межами цих чотирьох проміжків базовий сплайн дорівнює нулю. Отже, задача знаходження кожного конкретного з $N+3$ базових сплайнів має сталу складність n_B, що не залежить

від кількості вузлів, а задача знаходження всієї системи базових сплайнів потребує кількості дій, що дорівнює добуткові складності обчислення одного сплайну на кількість сплайнів, тобто має лінійну складність.

Базові сплайни для кінцевих вузлів, так само як і для додаткових уявних вузлів, можна побудувати, якщо доповнити мережу ще трьома уявними вузлами на кожному з кінців, а саме x_{-3} , x_{-2} , x_{-1}, x_{N+1}, x_{N+2}, x_{N+3}. Тоді кожен з вузлів, для якого потрібно побудувати базовий сплайн, стає внутрішнім. Звичайно, властивості побудованого в такий спосіб сплайну врешті-решт не залежатимуть, як саме обирались уявні вузли, тому їх можна обирати довільно. Найпростішим вибором буде використання відстані між першим та нульовим вузлами для побудови трьох лівих вузлів і так само – на другому кінці області інтерполяції.

2.3.4 Базові сплайни для рівномірної гратки

Задача побудови базових сплайнів, хоча й не потребує глибокого аналізу, є досить складною в обчислювальному плані, тому ми розглянемо спочатку суттєво простіший варіант сплайнів на рівномірній гратці, тобто такій, що відстані між кожними двома вузлами є однаковими і дорівнюють деякій величині h, що її називають кроком гратки. Вважатимемо, що перший вузол співпадає з початком числової вісі, тоді координата вузла k є просто $x_k = kh$.

Перше спрощення полягає в тому, що замість послідовності базових сплайнів, окремих для кожного вузла, маємо один спільний базовий сплайн, тільки що зсунутий відносно початку відліку на потрібну кількість кроків так, що базовий сплайн для к-того вузла є $B_k(x) = B(x - x_k)$. Дійсно, для базового сплайну з центром в к-тому вузлі маємо такий вираз

$$B_k(x) = \begin{cases} \alpha_k (x - (k-2)h)^3 ; (k-2)h \leqslant x \leqslant (k-1)h \\ 1 + b_k (x - kh) + c_k (x - kh)^2 + d_k (x - kh)^3 ; (k-1)h \leqslant x \leqslant kh \\ 1 + b_k (x - kh) + c_k (x - kh)^2 + e_k (x - kh)^3 ; kh \leqslant x \leqslant (k+1)h \\ \beta_k ((k+2)h - x)^3 ; (k+1)h \leqslant x \leqslant (k+2)h \end{cases} \tag{2.35}$$

Якщо тепер зробити заміну $x = kh + z$, цей вираз перетворюється на такий

$$B_k(z + kh) = \begin{cases} \alpha_k (z + 2h)^3 ; -2h \leqslant z \leqslant -h \\ 1 + b_k z + c_k z^2 + d_k z^3 ; -h \leqslant z \leqslant 0 \\ 1 + b_k z + c_k z^2 + e_k z^3 ; 0 \leqslant z \leqslant h \\ \beta_k (2h - z)^3 ; h \leqslant z \leqslant 2h \end{cases}, \tag{2.36}$$

що в ньому тільки номери коефіцієнтів нагадують про номер вузла.

Отже, позначимо тією ж літерою $B(x)$ тільки без індексу загальний базовий сплайн рівномірної гратки

$$B(x) = \begin{cases} \alpha (2h + x)^3 , -2h \leqslant x \leqslant -h \\ 1 + bx + cx^2 + dx^3 , -h \leqslant x \leqslant 0 \\ 1 + bx + cx^2 + ex^3 , 0 \leqslant x \leqslant h \\ \beta (2h - x)^3 , h \leqslant x \leqslant 2h \end{cases}, \tag{2.37}$$

тоді для сплайну з центром в вузлі k маємо такий вираз

$$B_k(x) = B(x - kh). \tag{2.38}$$

Симетричність базового сплайну відносно центрального вузла підказує, що розв'язок має бути теж симетричним, тому частина коефіцієнтів співпадатиме і можна одразу спростити вираз

$$B(x)=\begin{cases}\alpha(2h+x)^3;-2h\leqslant x\leqslant -h\\ 1+cx^2+dx^3;-h\leqslant x\leqslant 0\\ 1+cx^2-dx^3;0\leqslant x\leqslant h\\ \alpha(2h-x)^3;h\leqslant x\leqslant 2h\end{cases} \tag{2.39}$$

Для такого виразу всі рівняння в центральному вузлі задовольняються автоматично, а в двох сусідніх вузлах рівняння неперервності є однаковими

$$\begin{cases}\alpha_k h^3=1+c_k h^2-d_k h^3\\ -3\alpha_k h^2=2c_k h-3d_k h^2\\ 6\alpha_k h=2c_k-6d_k h\end{cases} \tag{2.40}$$

Розв'язавши цю систему, отримуємо такий вираз для базових сплайнів на рівномірній гратці

$$B(x)=\begin{cases}\frac{1}{4h^3}(x+2h)^3;-2h\leqslant x\leqslant -h\\ 1-\frac{3}{2h^2}x^2-\frac{3}{4h^3}x^3;-h\leqslant x\leqslant 0\\ 1-\frac{3}{2h^2}x^2+\frac{3}{4h^3}x^3;0\leqslant x\leqslant h\\ \frac{1}{4h^3}(2h-x)^3;h\leqslant x\leqslant 2h\end{cases} \tag{2.41}$$

Доцільно також заміною $x=zh$ перейти до нормованого аргументу

$$B(t)=\begin{cases}\frac{1}{4}(t+2)^3;-2\leqslant t\leqslant -1\\ 1-\frac{3}{2}t^2-\frac{3}{4}t^3;-1\leqslant t\leqslant 0\\ 1-\frac{3}{2}t^2+\frac{3}{4}t^3;0\leqslant t\leqslant 1\\ \frac{1}{4}(2-t)^3;1\leqslant t\leqslant 2\end{cases}, \tag{2.42}$$

тоді для базового сплайну в вузлі k на гратці з кроком h маємо такий вираз $B_k(x)=B\left(\frac{x}{h}-k\right)$.

2.3.5 Задача інтерполяції для рівномірної гратки

Повертаючись до задачі інтерполяції, потрібно визначити попередньо, які саме значення приймає базовий сплайн в вузлах гратки. Безпосередньою підстановкою легко отримати, що $B(\pm 1)=\frac{1}{4}$,
тому масове рівняння має вигляд

$$\frac{1}{4}s_{k-1}+s_{k-1}+\frac{1}{4}s_{k+1}=y_k. \tag{2.43}$$

В цьому рівнянні коефіцієнти не залежать від номера рівняння.

Крім масового рівняння, сплайн-коефіцієнти повинні задовольняти ще додатковим умовам. Далі будуть розглядатись лише стандартні умови

$$s''(x_0)=0;s''(x_N)=0. \tag{2.44}$$

Використовуючи подання сплайну, що розраховується, сумою базових сплайнів

$$s(x)=\sum_{k=-1}^{N+1}s_k B_k(x), \tag{2.45}$$

легко бачити, що додаткові умови перетворюються на пару рівнянь

$$\begin{array}{l} s''(x_0) = \left(s_{-1}\frac{d^2B\left(\frac{x}{h}+1\right)}{dx^2} + s_0\frac{d^2B\left(\frac{x}{h}\right)}{dx^2} + s_1\frac{d^2B\left(\frac{x}{h}-1\right)}{dx^2} \right)\Big|_{x=0} = 0 \\ s''(x_N) = \left(s_{N-1}\frac{d^2B\left(\frac{x}{h}-N+1\right)}{dx^2} + s_N\frac{d^2B\left(\frac{x}{h}-N\right)}{dx^2} + s_{N+1}\frac{d^2B\left(\frac{x}{h}-1-N\right)}{dx^2} \right)\Big|_{x=Nh} = 0 \end{array}. \tag{2.46}$$

Для другої похідної базового сплайну в вузлах мережі маємо такі значення

$$B''(\pm 1) = \frac{3}{2}; B''(0) = -3, \tag{2.47}$$

тому додаткові умови є

$$\left\{ \begin{array}{l} s''(0) = \frac{3}{2h^2}s_{-1} - \frac{3}{h^2}s_0 + \frac{3}{2h^2}s_{+1} = 0 \\ s''(Nh) = \frac{3}{2h^2}s_{N-1} - \frac{3}{h^2}s_N + \frac{3}{2h^2}s_{N+1} = 0 \end{array} \right. . \tag{2.48}$$

Обидва ці рівняння можна скоротити, надавши їм вигляду рівнянь, що визначають додаткові сплайн-коефіцієнти по розв'язках масової задачі.

$$\left\{ \begin{array}{l} s_{-1} = 2s_0 - s_{+1} \\ s_{N-1} = 2s_N - s_{N+1} \end{array} \right. . \tag{2.49}$$

Залучаючи до розгляду перше рівняння масової системи, маємо пару рівнянь

$$\left\{ \begin{array}{c} s_{-1} = 2s_0 - s_1 \\ \frac{1}{4}s_{-1} + s_0 + \frac{1}{4}s_{+1} = y_0 \end{array} \right. , \tag{2.50}$$

звідки отримуємо вже нормальне кінцеве рівняння для тридіагональної матриці

$$s_0 = \frac{2}{3}f_0 \tag{2.51}$$

і подібне кінцеве рівняння в останньому вузлі

$$s_N = \frac{2}{3}f_N \tag{2.52}$$

Останні два рівняння і є потрібними для методу прогонки кінцевими рівняннями.

Отже, повна система рівнянь методу прогонки складається з масового рівняння та двох кінцевих

$$\begin{array}{l} s_0 = \frac{2}{3}y_0 \\ \frac{1}{4}s_{k-1} + s_{k-1} + \frac{1}{4}s_{k+1} = y_k;\ k = 1 \ldots N-1 \\ s_N = \frac{2}{3}y_N \end{array} \tag{2.53}$$

Отримавши розв'язок цієї системи, можна будувати власне сплайн. Розв'язок шукається методом прогонки. Відповідно до базової ідеї цього методу покладаємо

$$s_{k+1} = \alpha_k s_k + \beta_k \tag{2.54}$$

і підстановкою в масове рівняння знаходимо рівняння для коефіцієнтів рекурсії α_k, β_k

$$\begin{array}{l} \frac{1}{4} + \left(1 + \frac{1}{4}\alpha_k\right)\alpha_{k-1} = 0;\ k = 1 \ldots N-1 \\ \left(1 + \frac{1}{4}\alpha_k\right)\beta_{k-1} + \frac{1}{4}\beta_k = f_k \end{array} . \tag{2.55}$$

Права кінцева умова має задовольнятись для довільного значення коефіцієнтів. Співвідношення $s_N = \alpha_{N-1} s_{N-1} + \beta_{N-1}$ разом з кінцевою умовою $s_N = \frac{2}{3} f_N$ дають вирази

$$\beta_{N-1} = \frac{2}{3} f_N; \alpha_{N-1} = 0 \tag{2.56}$$

для початку рекурсивного обчислення для коефіцієнтів рекурсії α_k, β_k за формулами

$$\begin{cases} \alpha_{k-1} = -\frac{1}{4+\alpha_k} \\ \beta_{k-1} = \frac{4f_k - \beta_k}{4+\alpha_k} \end{cases} ; k = N-1 \ldots 1. \tag{2.57}$$

Ліва кінцева умова дозволяє обчислити початкове значення $s_0 = \frac{2}{3} f_0$ для рекурсивного розрахунку коефіцієнтів – відповідей. По закінченні рекурсивних обчислень додаткові сплайн-коефіцієнти обчислюються з виразів

$$\begin{cases} s_{-1} = 2s_0 - s_{+1} \\ s_{N-1} = 2s_N - s_{N+1} \end{cases} \tag{2.58}$$

і на цьому розрахунок сплайн-коефіцієнтів закінчується.

2.3.6 Розрахунок сплайну

Останнім етапом обчислення сплайну є розрахунок коефіцієнтів сплайну, тобто коефіцієнтів $\{a_k, b_k, c_k, d_k; k = 0 \ldots N-1\}$ поліномів, з яких складається сплайн $s(x) = \left\{a_k + b_k x + c_k x^2 + d_k x^3;\ x_k \leqslant x \leqslant x_{k+1}\right\}$.

Цей розрахунок є масовою операцією, оскільки потрібно розрахувати по 4 коефіцієнти на кожен відрізок – загалом $4N$ коефіцієнтів. Всі ці коефіцієнти можна отримати, прирівнявши для певного проміжку поліном $a_m + b_m x + c_m x^2 + d_m x^3$ до того виразу, що його має подання цього ж сплайну сумою базових сплайнів

$$s(x) = \sum_{k=-1}^{N+1} s_k B_k(x). \tag{2.59}$$

В цій сумі більшість членів мають нульове значення, а саме ті, що центри відповідних базових сплайнів відстоять на два чи більше вузли від проміжку $[mh, (m+1)h)$. Тому в сумі залишаються тільки чотири члени

$$\begin{matrix} s(x) = s_{m-1} B_{m-1}(x) + s_m B_m(x) + s_{m+1} B_{m+1}(x) + s_{m+2} B_{m+2}(x); \\ x_m < x < x_{m+1} \end{matrix}, \tag{2.60}$$

до того ж для кожного з членів потрібно обчислювати тільки один з можливих чотирьох поліномів.

Маючи на увазі, що за виразом для кожного з поліномів надалі буде здійснюватись обчислення, доцільно трохи змінити вигляд цих поліномів.

По-перше, якщо для кожного поліному зсунути початок відліку на початок цього відрізку, то модифікований в такий спосіб поліном

$$y_m + b_m (x - x_m) + c_m (x - x_m)^2 + d_m (x - x_m)^3 \tag{2.61}$$

не вимагатиме вже обчислення степенів великих значень аргументу, а його вільний член співпадатиме зі значенням функції в вузлі.

По-друге, для малого кроку коефіцієнти при старших степенях повинні бути непропорційно великими, аби забезпечити потрібні зміни значень функції на відрізку. Якщо ж заміною $x = mh + zh$ перейти до нормалізованої координати, що змінюватиметься в межах від нуля (початок відрізку) до одиниці (кінець відрізку), значення нормалізованих коефіцієнтів будуть співвимірними до варіації функції на проміжку. Це також зменшуватиме вплив похибки округлення. Отже, потрібним виразом для поліному на кожному відрізку є

$$y_m + b_m z + c_m z^2 + d_m z^3; z = \frac{x}{h} - m \tag{2.62}$$

З огляду на це підставимо в суму базових сплайнів замість координати вираз $x = mh + zh$ і розглянемо, який вираз будуть мати базові сплайни

Першим членом суми є $B_{m-1}(mh + zh)$. Відповідно до закону перетворення базового сплайну на сплайн з вершиною в певному вузлі $B_k(x) = B\left(\frac{x}{h} - k\right)$ маємо для нього вираз

$$B_{m-1}(mh + zh) = B\left(\frac{mh + zh}{h} - (m-1)\right) = B(z+1). \tag{2.63}$$

Використаємо тепер загальний вираз для базового сплайну

$$B(z+1) = \begin{cases} \frac{1}{4}(z+1+2)^3; -2 \leqslant z+1 \leqslant -1 \\ 1 - \frac{3}{2}(z+1)^2 - \frac{3}{4}(z+1)^3; -1 \leqslant (z+1) \leqslant 0 \\ 1 - \frac{3}{2}(z+1)^2 + \frac{3}{4}(z+1)^3; 0 \leqslant (z+1) \leqslant 1 \\ \frac{1}{4}(2-(z+1))^3; 1 \leqslant (z+1) \leqslant 2 \end{cases}. \tag{2.64}$$

Умові $0 \leqslant z < 1$ задовольняє лише останній рядок цього виразу, отже, маємо

$$B_{m-1}(mh + zh) = \frac{1}{4}(2-(z+1))^3 = \frac{1}{4}(1-z)^3. \tag{2.65}$$

Подібно до цього маємо такі вирази для інших членів суми

$$\begin{array}{l} B_m(mh + zh) = 1 - \frac{3}{2}z^2 + \frac{3}{4}z^3; \\ B_{m+1}(mh + zh) = 1 - \frac{3}{2}(1-z)^2 + \frac{3}{4}(1-z)^3; \\ B_{m+2}(mh + zh) = \frac{1}{4}z^3. \end{array} \tag{2.66}$$

Таким чином, для поліному на відрізку маємо вираз

$$\begin{array}{l} s(mh + zh) = s_{m-1}\frac{1}{4}(1-z)^3 + s_m\left(1 - \frac{3}{2}z^2 + \frac{3}{4}z^3\right) \\ +s_{m+1}\left(1 - \frac{3}{2}(1-z)^2 + \frac{3}{4}(1-z)^3\right) + s_{m+2}\left(\frac{1}{4}z^3\right) \end{array}, \tag{2.67}$$

в якому ще потрібно привести подібні, що дасть

$$\begin{array}{l} s(mh + zh) = \left(\frac{s_{m-1}}{4} + s_m + \frac{s_{m+1}}{4}\right) + \left(-\frac{3}{4}s_{m-1} + \frac{3}{4}s_{m+1}\right)z \\ + \left(\frac{3}{4}s_{m-1} - \frac{3}{2}s_m + \frac{3}{4}s_{m+1}\right)z^2 + \left(-\frac{1}{4}s_{m-1} + \frac{3}{4}s_m - \frac{3}{4}s_{m+1} + \frac{1}{4}s_{m+2}\right)z^3 \end{array}. \tag{2.68}$$

Тут в першій дужці легко впізнати масове рівняння, а загалом отримуємо такі значення для коефіцієнтів сплайну

$$\begin{array}{l} y_m = \frac{s_{m-1}}{4} + s_m + \frac{s_{m+1}}{4} \\ b_m = -\frac{3}{4}s_{m-1} + \frac{3}{4}s_{m+1} \\ c_m = \frac{3}{4}s_{m-1} - \frac{3}{2}s_m + \frac{3}{4}s_{m+1} \\ d_m = -\frac{1}{4}s_{m-1} + \frac{3}{4}s_m - \frac{3}{4}s_{m+1} + \frac{1}{4}s_{m+2} \end{array}. \tag{2.69}$$

Здійснивши розрахунок коефіцієнтів сплайну, ми отримуємо вирази, що є досить зручними для практичного використання. Перший член в кожному з цих виразів співпадає із значенням функції в вузлі (масове рівняння), а всі інші визначають відхилення в значенні функції на відрізку.

2.3.7 Розрахунок значення сплайну в заданій точці

Задача обчислення значень сплайну в заданих точках звичайно є більш масовою, ніж навіть задача обчислення коефіцієнтів сплайну, адже головною метою використання сплайну є обчислення значень між вузлами. Якщо проміжних точок небагато, або якщо відхилення не є досить помітними, використання сплайн-інтерполяції взагалі недоцільне, тож звичайно кількість точок, в яких потрібно здійснювати обчислення, в десятки разів перевищує кількість вузлів інтерполяції.

Перед використанням конкретного виразу для поліному на конкретному відрізку потрібно ще визначити, до якого саме відрізку належить значення аргументу. Для рівномірної гратки це визначається досить просто, а саме, результатом ділення аргументу на крок x/h буде дійсне число, що має дробову $z = \{x/h\}$ та цілу $m = [x/h]$ частини. Легко зрозуміти, що значення аргументу знаходиться в межах $[x/h]\, h \leqslant x < ([x/h] + 1)\, h$, тож ціла частина m і є номером проміжку, а дробова має бути використаною аргументом відповідного полінома.

Отже алгоритм обчислення значення складається з обчислення номера проміжку $m = [x/h]$, нормалізованої величини зсуву від початку проміжку $z = \{x/h\}$ та розрахунку значення полінома за стандартною схемою Горнера

$$s(z) = |m = [x/h]\,; z = \{x/h\}| = y_m + z\,(b_m + z\,(c_m + z d_m))\,. \tag{2.70}$$

На цьому задачу побудови алгоритму сплайн-інтерполяції на рівномірній гратці закінчено.

2.3.8 Програмна реалізація

Метод сплайн-інтерполяції потребує, порівняно з поліноміальною інтерполяцією, значної кількості дій, що передують власне обчисленню значень. З одного боку, на кожному відрізку інтерполяції значення сплайна розраховуються досить просто, як значення відповідного полінома третього степеню. З іншого ж боку, коефіцієнти кожного полінома можна вирахувати тільки в сукупності з коефіцієнтами усіх інших, адже рівняння на коефіцієнти полінома для кожного конкретного відрізку інтерполяції переплутані з рівняннями на коефіцієнти поліномів для сусідніх відрізків. Тому процес ініціалізації розрахункової структури доцільно відокремлювати від процесу розрахунку потрібних значень. Ефективним засобом такого відокремлення є використання спеціальних типів даних, що поєднують і місце в пам'яті для збереження параметрів, і методи використання цих параметрів і методи ініціалізації потрібних значень параметрів. Мова Pascal для такого об'єднання даних та методів їх обробки використовує уявлення про об'єкти, мова C++ - структури та класи. Мова Maple не відрізняє взагалі окремо типів для даних та методів – і змінні і програми цією мовою створюються динамічно в процесі виконання і в звичайному списку можна поєднувати і дані і процедури.

Отже, задача про побудову програми сплайн-інтерполяції є задачею про побудову програми перетворення вхідної таблиці значень функції дискретної змінної на таблицю результатів розрахунку значень в деякій кількості точок, що знаходяться між вузлами таблиці.

Вхідними даними задачі є значення координат вузлів $\{x_k, y_k; k = 0 \ldots N\}$, а результатом – значення координат точок графіку інтерполяційного сплайну $\{x_p, y_p; p = 0 \ldots M >> N\}$. Ця задача природним чином ділиться на дві частини – побудову програми підготовки даних для розрахунку результатів та програми обчислення результатів за проміжними даними.

Проміжним результатом першої частини програми є набір коефіцієнтів інтерполяційного сплайну – коефіцієнтів кубічних поліномів для кожного проміжку інтерполяції $\{x_k, a_k, b_k, c_k, d_k; k = 0 \ldots N-1\}$, або в більш придатній для обчислення результату формі $\{x_k, h_k, y_k, d_k, dd_k, t_k; k = 0 \ldots N-1\}$, що дозволяє більш ефективно обчислювати кінцеві результати відповідно до формули

$$s(x) : x_k \leqslant x < x_k + h_k := \left| z = \frac{x - x_k}{h_k} \right| = y_k + z \cdot (d_k + z \cdot (dd_k + z \cdot t_k)) \qquad (2.71)$$

Власне, розрахунок додаткових параметрів $\{h_k, d_k, dd_k, t_k; k = 0 \ldots N-1\}$, що відсутні в наборі вхідних даних, і є задачею першої частини програми – процесу ініціалізації.

Доцільно зауважити, що до другої частини програми потрібно ще формулювати додаткові вхідні дані про розташування послідовності потрібних точок $\{x_p; p = 0 \ldots M\}$, для яких будуть розраховуватись значення, але від цієї частини постановки задачі залежатимуть тільки окремі деталі програми, тому її можна розглядати окремо.

Для рівномірної гратки набір вихідних даних дещо скорочується, оскільки, по-перше, крок гратки є однаковим для всіх вузлів і його можна сберігати за межами масиву параметрів поліномів, по-друге, координата початку проміжку однозначно визначається номером і її теж можна вилучити з переліку параметрів для конкретного відрізку. Після такого скорочення набір даних, що їх потрібно сберігати для кожного вузла, повинен вміщувати тільки чотири коефіцієнти $\{y, d, dd, t\}$, а метод обробки цих даних, що дозволятиме отримувати необхідне значення функції, буде мати вигляд функції

$$s(z) := y + z \cdot (d + z \cdot (dd + z \cdot t)). \qquad (2.72)$$

Реалізація цієї конструкції мовою Pascal здійснюється за допомогою оголошень

```
Type
    PieceT=object
            y,d,dd,t :Double;
            function value(z:Double):Double;
            end;

function PieceT.value(z:Double):Double;
 begin
     value:=y+z*(d+z*(dd+z*t));
 end;
```

Перше з них є оголошенням типу даних `PieceT`, що будуть пізніше використані для побудови сплайну, друге – реалізацією методу
`value(z:Double):Double;`

для цього типу.

Мовою C++ доцільніше використати не клас, що накладатиме додаткові вимоги на розподіл параметрів на загальнодоступні та приватні, а просто структуру

```
struct piece{
    double y,d,dd,t;
  double value(double z){return y+z*(d+z*(dd+z*t));}
}
```

Вже в цьому прикладі легко порівняти переваги та недоліки цих двох найпопулярніших мов програмування. Якщо C++ підтримує набагато коротший код, то Pascal надає більшу прозорість окремим розділам коду.

Означена структура є набором даних тільки для одного проміжку, тоді як сплайн повинен містити подібні набори даних для кожного з проміжків. Відповідною структурою даних має бути масив, що його довжина залежить від того, скільки саме вузлів інтерполяції буде в тому чи іншому конкретному наборі даних для інтерполяції.

Невизначеність довжини масиву не дозволяє жодній з мов оголосити потрібну структуру просто у вигляді масиву даних, адже розмір області пам'яті, що виділяється під дані, визначається ще на етапі трансляції програми і не може бути змінений в процесі виконання. Використовувати ж просто найбільший можливий розмір пам'яті для цього масиву просто недоцільно. В 16 – розрядних ОС, як DOS, цей розмір занадто малий навіть для пересічної задачі (на кожен проміжок потрібно 4*8=32 байти, то 1000 проміжків вже вичерпають весь блок пам'яті під статичні дані), а в 32 – розрядних програма може використати навіть гігабайт пам'яті під дані, але якщо стільки місця в пам'яті виділити під статичні дані, вона буде вимагати від комп'ютера реального гігабайту оперативної пам'яті.

Ефективним механізмом збереження даних, кількість котрих невідома до початку роботи програми, є динамічний розподіл пам'яті, коли програма запрошує у виконавчої системи стільки пам'яті, скільки потрібно саме на обробку отриманих даних.

Мовою Pascal динамічний розподіл пам'яті можна здійснити в такий спосіб:

Спочатку оголошується тип даних, що є масивом найбільшої можливої довжини

```
 const
    maxlen=2000 -1;{for DOS, or 1024*1024 -1 (Win32,Unix)}
 Type
  SplineP=^SplineT;
  SplineT=Array[0..maxlen] of PieceT;
 Var
  sp :SplineP;
procedure Init;
 ...
 begin
 ...
 sp:=New(Number*SizeOf(PieceT));
 ...
 end;
```

Додатково також оголошується тип, вказівний на тип масиву, а також змінна вказівного на масив типу, що, власне, і буде вказувати на область, де зберігатимуться дані для сплайну. Потім в процедурі ініціалізації вже після визначення необхідної довжини реального масиву коефіцієнтів сплайну (змінна Number) оператором New(Number*SizeOf(PieceT)) виділяється необхідне місце для збереження всіх потрібних коефіцієнтів.

Мовою C++ ці ж дії виконуються трохи коротше

```
#define maxlen 1024*1024-1
  piece *sp;
void Init(){
 ...
 sp=new piece[Number];
 ...
};
```

Оскільки в цій мові змінна типу масив є одночасно вказівною на тип елементів масиву, виділення пам'яті для вказівної змінної еквівалентно створенню масиву.

Процедура розрахунку значення інтерполюючого сплайну для заданого аргументу може бути тепер записана, як функція, що отримує координату потрібної точки, обчислює номер потрібного проміжку і викликає метод value для відповідного примірника об'єкта

```
PASCAL
function Value(x:Double):Double;
 var
   k:Integer;

 begin
   k:=Trunc(((x-x0)/h);
   Value:=sp\ [k].value((x-x0)/h -k);
 end;

C++
double value(double x){
  int k= (int)floor(((x-x0)/h);
  return sp[k].value(((x-x0)/h -k);
}
```

Змінні x0, h повинні бути оголошеними, як глобальні.

Власне на побудові цієї функції другу частину програми – обчислення результатів інтерполяції, слід вважати закінченою, оскільки набір точок, для яких будуть розраховуватись значення, повинен визначатись окремо.

Перша частина програми полягає в тому, що потрібно з якихось джерел отримати значення таблички функції дискретної змінної і розрахувати за цими значеннями коефіцієнти сплайну.

Вважатимемо, що таблиця значень функції, що підлягає інтерполяції, записана в файл вхідних даних з назвою data.txt в форматі

```
double double \n,
```

тобто в кожному рядку знаходиться пара значень xy чергового вузла заданої функції, а кількість вузлів потрібно вирахувати відповідно до кількості рядків (подвійним символом $\backslash n$ стандартно позначається кінець рядка).

Процедура ініціалізації починається із зачитування даних

```
procedure Init;
 const
    dataname='data.txt';
 var
    datafile  :Text;
    x,y       :Array[0..maxlen] of Double;

...
 begin
     assign(datafile,dataname);reset(datafile);
     Number:=-1;
     While not Eof(datafile)
     do begin
        Number:=Number+1;
        ReadLn(datafile,x[Number],y[Number]);
        end;
 end;
```

В C++ простіше використати процедуру зачитування із стандартного вхідного потоку, запускаючи програму з відповідним аргументом spline.exe $<$ data.txt. Тому початок процедури ініціалізації є трохи компактнішим

```
void Init(){
  double x[maxlen];double y[maxlen];
  for(number=0;cin>>x[number]>>y[number];number++);number--;
}
```

Тут оператор зачитування із вхідного потоку одночасно формує умову нормального зачитування, як тільки потік (тобто файл!) закінчується, умова порушується і цикл буде закінчено.

Звичайно ж, в обох варіантах потрібно було б реалізовувати програмно динамічне виділення пам'яті під тимчасові масиви, але це призведе тільки до зайвого ускладнення тексту програми.

В обох варіантах по закінченні циклу зачитування змінна number має своїм значенням кількість проміжків.

Наступним етапом процесу ініціалізації є розрахунок сплайн-коефіцієнтів методом прогонки. Цей розрахунок вимагає створення двох масивів для проміжних коефіцієнтів та масиву власне сплайн-коефіцієнтів, до того ж алгоритм вимагає, аби масив сплайн-коефіцієнтів мав найменшим значенням індексу -1. Тут вже спрацьовує добре відомий недолік мови C++ - вона не дозволяє оголошувати масиви з першим індексом, що відрізнявся б від нуля. Взагалі кажучи, можна працювати і так, пам'ятаючи просто, що індекс масиву сплайн-коефіцієнтів зсунутий на одиничку відносно теоретично потрібного, але за такими дрібницями досить часто потім ховаються помилки в алгоритмі, що їх до того ж досить складно локалізувати. Позбутися цієї неприємності можна за допомогою спеціальної структури

```
struct coeff{
```

```
      double *v;
      int m;
   coeff(int min,int max){m:=min;v:=new double[max+min];}
   double& operator[](int k){return v[m+k];}
}
```

Вона дозволяє використовувати масив, оголошений як
`coeff *s=new coeff(-1,number+1);`
з цілком нормальною нумерацією елементів.

Подальші розрахунки за методом прогонки не відрізняються від того, що було в програмі реалізації цього методу, а по закінченні обчислення сплайн-коефіцієнтів мають бути обчислені вже коефіцієнти самого сплайну.

Слід звернути увагу, що внутрішнім змінним процедури ініціалізації пам'ять виділяється тільки під час її виконання і звільняється по її закінченні. Але динамічно розподілена пам'ять залишається розподіленою і по її закінченні, якщо не застосувати операцій звільнення пам'яті. Саме тому робота з динамічними змінними вимагає більшої уваги від програміста.

Повний текст відповідних програм наведено нижче. Також наведено і реалізацію алгоритму сплайн-інтерполяції в системі Maple. Аналіз особливостей цього прикладу залишається на самостійну роботу.

Реалізація мовою Pascal

```
program splineq;
Type
  poly=record y,d,dd,t:Double;end;
  mass=Array[0..1000] of double;
  smas=Array[-1..1001] of double;
var
 spline  :Array[0..1000] of poly;
 N       :Integer;
 yn,h    :Double;

function value(x:Double):Double;
 var
    z :Double;
    k :Integer;
 begin
     if x<0 then begin Writeln('negative argument');Exit;end;
     if x>N*h then begin Writeln('large argument');Exit;end;
     if x=N*h
       then value:=yn
       else begin
            k:=Trunc(x/h);z:=x/h-k;
            value:=spline[k].y+z*(spline[k].d+z*(spline[k].dd+z*spline[k].t));
            end;
  end;

Procedure Start;
 var
```

```
  alpha,beta : ^mass;
  s          : ^smas;
  k          :Integer;
  datafile   :Text;
 const
  dataname='data.txt';
 begin
     N:=-1; new(s);new(alpha);new(beta);
     assign(datafile,dataname);reset(datafile);
     ReadLn(datafile,h);
     while not Eof(datafile)
     do begin N:=N+1; ReadLn(datafile,spline[N].y);end;
     close(datafile);
     yn:=spline[N].y;
     alpha^[N-1]:=0;beta^[N-1]:=2*yn/3;
     for k:=N-1 downto 1
     do begin
        alpha^[k-1]:=-1/(4+alpha^[k]);
        beta^[k-1]:=(4*spline[k].y-beta^[k])/(4+alpha^[k]);
        end;
     s^[0]:=2*spline[0].y/3;
     for k:=0 to N-1
     do s^[k+1]:=alpha^[k]*s^[k]+beta^[k];
     s^[-1]:=2*s^[0]-s^[1];s^[N+1]:=2*s^[N]-s^[N-1];
     for k:=0 to N-1
     do begin
        spline[k].d :=3*(s^[k+1]-s^[k-1])/4;
        spline[k].dd :=3*(s^[k+1]-2*s^[k]+s^[k-1])/4;
        spline[k].t :=(s^[k+2]-3*s^[k+1]+3*s^[k]-s^[k-1])/4;
        end;
     dispose(s);dispose(alpha);dispose(beta);
end;

Procedure Res;
  var
     x,step  :Double;
     resfile :Text;
 const
     resname='res.txt';
 begin
   assign(resfile,resname);rewrite(resfile);
   x:=0;step:=h/5;
   repeat
       WriteLn(resfile,x,' ',Value(x));
       x:=x+step
       until x>h*N;
   close(resfile);
 end;

BEGIN
```

```
     Start;
     Res;
END.
```

Реалізація мовою C++

```
file main.c

#include "spline.h"
#include <stdio.h>

int main(){
  spline s=spline();
  double step=s.h/5;
  for(double x=0;x<=(s.h*s.len+step/2);x+=step){
      cout<<x<<"\t"<<s.val(x)<<"\n";};
  return 0;
}

file spline.h

#include <iostream.h>
#define maxdim 100000

struct piece{
    double y,d,dd,t;
};

struct coeff{
     double *v;
     int len;
  coeff(int n){v=new double[n+3];len=n+3;}
  double& operator[](int k){return v[k+1];}
};

struct spline{
       piece *v;
       double h,yn;
       int len;
   spline();
   double val(double);
};




file spline.c
```

```
#include "spline.h"
#include <stdio.h>
#include <math.h>

 double spline::val(double arg){
      if (arg<0) {return(v[0].y);};
      if (arg>=h*len) {return(yn);};
      int k=int (floor(arg/h));
      double z=arg/h-k;
      return(v[k].y+z*(v[k].d+z*(v[k].dd+z*v[k].t)));
};

spline::spline(){
  double *buf=new double[maxdim];
  cin>>h;
  for(len=0;(len<maxdim){\&}{\&}(cin>>buf[len]);len++){\{}{\}};
  len--;
  yn=buf[len];
  v=new piece[len];
  for(int k=0;k<len;k++){\{}v[k].y=buf[k];{\}};
  delete(buf);
  double alpha[len];
  double beta[len];
  coeff s=coeff(len);
  alpha[len-1]=0;beta[len-1]=2*yn/3;
  for(int k=len-1;k>0;k--){
    alpha[k-1]=-1/(4+alpha[k]);
    beta[k-1]=(4*v[k].y-beta[k])/(4+alpha[k]);
    }
  s[0]=2*v[0].y/3;
  for(int k=0;k<len;k++){s[k+1]=alpha[k]*s[k]+beta[k];};
  s[-1]=2*s[0]-s[1];
  s[len+1]=2*s[len]-s[len-1];
  for(int k=0;k<len;k++){
    v[k].d=3*(s[k+1]-s[k-1])/4;
    v[k].dd=3*(s[k+1]+s[k-1]-2*s[k])/4;
    v[k].t=(s[k+2]-3*s[k+1]+3*s[k]-s[k-1])/4;
    };
};
```

```
> restart:with(plots): path:="path to
> maple\\spline\\": dataname:="data.txt":resname:="res.txt":
> source:=cat(path,dataname):target:=cat(path,resname):
> dat:=readdata(dataname,1);nmax:=nops(dat)-2;
> sp:=array[0..nmax-1][yk,dk,ddk,tk];h:=dat[1];
> for k from 0 to nmax do sp[k][yk]:=dat[k+2] od;
```

$$dat := [.1,\ 0.,\ .25,\ 1.,\ .25,\ 0.]$$

$$nmax := 4$$

$$sp := array_{0..3\,yk,\ dk,\ ddk,\ tk}$$

$$h := .1$$

$$sp_{0\,yk} := 0.$$

$$sp_{1\,yk} := .25$$

$$sp_{2\,yk} := 1.$$

$$sp_{3\,yk} := .25$$

$$sp_{4\,yk} := 0.$$

```
> alpha:=array[0..nmax-1];
```

$$\alpha := array_{0..3}$$

```
> beta:=array[0..nmax-1];
```

$$\beta := array_{0..3}$$

```
> alpha[nmax-1]:=0;beta[nmax-1]:=sp[nmax][yk];
```

$$\alpha_3 := 0$$

$$\beta_3 := 0.$$

```
> for k from nmax-1 to 1 by -1 do
> alpha[k-1]:=-1/(4+alpha[k]);beta[k-1]:=(4*sp[k][yk]-beta[k])/(4+alpha[
> k]);od;
```

$$\alpha_2 := \frac{-1}{4}$$

$$\beta_2 := .2500000000$$

$$\alpha_1 := \frac{-4}{15}$$

$$\beta_1 := 1.000000000$$

$$\alpha_0 := \frac{-15}{56}$$

$$\beta_0 := 0.$$

```
> s:=array[-1..nmax+1];
```

$$s := array_{-1..5}$$

```
> s[0]:=2*sp[0][yk]/3;
```

$$s_0 := 0.$$

```
> for k from 0 to nmax-1 do s[k+1]:=alpha[k]*s[k]+beta[k];od;
```

$$s_1 := 0.$$

$$s_2 := 1.000000000$$

$$s_3 := 0.$$

$$s_4 := 0.$$

```
> s[-1]:=2*s[0]-s[1];s[nmax+1]:=2*s[nmax]-s[nmax-1];
```

$$s_{-1} := 0.$$

$$s_5 := 0.$$

```
> for k from 0 to nmax-1 do
> sp[k][dk]:=3/4*(s[k+1]-s[k-1]);
> sp[k][ddk]:=3/4*(s[k-1]+s[k+1]-2*s[k]);
> sp[k][tk]:=1/4*(-s[k-1]+3*s[k]-3*s[k+1]+s[k+2]);
> od;
```

$$sp_{0\,dk} := 0.$$

$$sp_{0\,ddk} := 0.$$

$$sp_{0\,tk} := .2500000000$$
$$sp_{1\,dk} := .7500000000$$
$$sp_{1\,ddk} := .7500000000$$
$$sp_{1\,tk} := -.7500000000$$
$$sp_{2\,dk} := 0.$$
$$sp_{2\,ddk} := -1.500000000$$
$$sp_{2\,tk} := .7500000000$$
$$sp_{3\,dk} := -.7500000000$$
$$sp_{3\,ddk} := .7500000000$$
$$sp_{3\,tk} := -.2500000000$$

```
> val:=proc(x) local k,z;global sp,h;
> k:=floor(x/h);z:=frac(x/h);
> RETURN(sp[k][yk]+z*(sp[k][dk]+z*(sp[k][ddk]+z*sp[k][tk])));
> end;
```

$val := \mathbf{proc}(x)$
local k, z;
global sp, h;
$k := \mathrm{floor}(x/h)$;
$z := \mathrm{frac}(x/h)$;
$\mathrm{RETURN}(sp_{k\,yk} + z * (sp_{k\,dk} + z * (sp_{k\,ddk} + z * sp_{k\,tk})))$
end proc

```
> val(0.1);
```

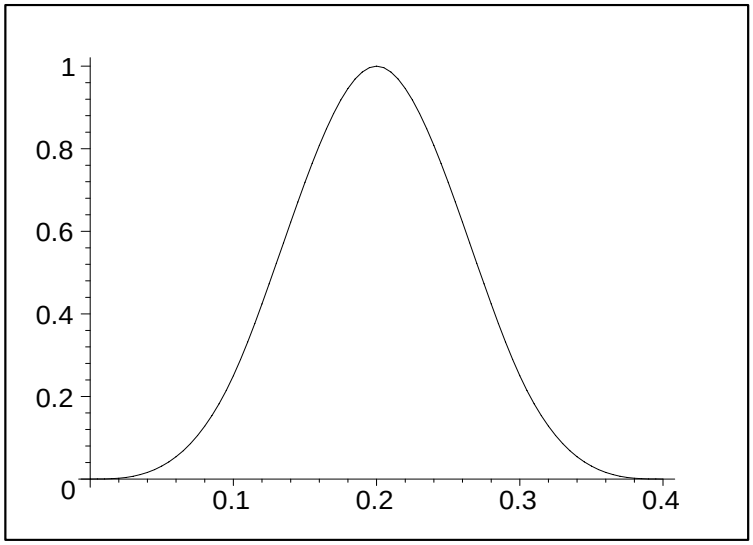
.25

```
> np:=5:step:=h/np:res:=[seq([k*step,val(k*step)],k=0..nmax*np)];
```

$res :=$ [[0., 0.], [.02000000000, .002000000000], [.04000000000, .01600000000], [.06000000000, .05400000000], [.08000000000, .1280000000], [.1000000000, .25], [.1200000000, .4240000000], [.1400000000, .6220000000], [.1600000000, .8080000000], [.1800000000, .9460000000], [.2000000000, 1.], [.2200000000, .9460000000], [.2400000000, .8080000000], [.2600000000, .6220000000], [.2800000000, .4240000000], [.3000000000, .25], [.3200000000, .1280000000], [.3400000000, .0540000000], [.3600000000, .0160000000], [.3800000000, .0020000000], [.4000000000, 0.]]

```
> np:=20:step:=h/np:PLOT(CURVES([seq([k*step,val(k*step)],k=0..nmax*np)
> ]));
```

Реалізація Maple


1
0.8
0.6
0.4
0.2
0
0.1
0.2
0.3
0.4

2.3.9 Кубічні сплайни на нерівномірній мережі.

Загальна постановка задачі про сплайн-інтерполяцію не залежить від рівномірності чи нерівномірності мережі, але алгоритм побудови розв'язків для нерівномірної мережі суттєво ускладнюється. Нерівномірній мережі не можна поставити у відповідність базовий сплайн, що відрізнявся б тільки номером вузла від подібних йому сплайнів в сусідніх вузлах.

Алгоритм розрахунку сплайнів може бути побудований, або як суцільний алгоритм прямого розрахунку всього сплайну, або як послідовність дій, подібна до таких, що використовувались для рівномірної гратки - побудови системи базових сплайнів, розв'язування рівнянь інтерполяції і розрахунку коефіцієнтів сплайну. Варіант прямого розрахунку працюватиме трохи швидше, але він набагато складніший для відлагодження, тому тут буде розглянуто метод базових сплайнів.

У разі нерівномірної мережі для кожного вузла потрібно розраховувати свій варіант базового сплайну, що його явний вигляд суттєво залежатиме від співвідношення між кроками тих проміжків, для яких цей сплайн відрізняється від нуля.

Рівняння інтерполяції

$$s(x_k) = f_k;\ k = 0 \dots N \tag{2.73}$$

разом з умовами неперервності

$$\begin{cases} s(x_k + 0) = s(x_k - 0) \\ s'(x_k + 0) = s'(x_k - 0) \\ s''(x_k + 0) = s''(x_k - 0) \end{cases};\ k = 1 \dots N - 1 \tag{2.74}$$

та кінцевими умовами

$$\begin{cases} s''(x_0) = 0 \\ s''(x_N) = 0 \end{cases} \tag{2.75}$$

повністю визначають сплайн

$$\begin{cases} s''(x_0) = 0 \\ s''(x_N) = 0 \end{cases} \tag{2.76}$$

на нерівномірній мережі, так само як і на рівномірній.

За методом базових сплайнів в повній системі умови неперервності виділяються в окрему підсистему, що визначає саме базові сплайни з такими властивостями:

Для кожного вузла означено базовий сплайн з вершиною в цьому вузлі, що відрізняється від нуля на чотирьох найближчих до вузла проміжках, по два з кожного боку, а в центральному вузлі дорівнює одиниці.

$$B_k(x) = \begin{cases} a_{k,-2} + b_{k,-2}x + c_{k,-2}x^2 + d_{k,-2}x^3; x_{k-2} \leqslant x \leqslant x_{k-1} \\ a_{k,-1} + b_{k,-1}x + c_{k,-1}x^2 + d_{k,-1}x^3; x_{k-1} \leqslant x \leqslant x_k \\ a_{k,0} + b_{k,0}x + c_{k,0}x^2 + d_{k,0}x^3; x_k \leqslant x \leqslant x_{k+1} \\ a_{k,1} + b_{k,1}x + c_{k,1}x^2 + d_{k,1}x^3; x_{k+1} \leqslant x \leqslant x_{k+2} \\ 0; x \leqslant x_{k-2} \cup x \geqslant x_{k+2} \end{cases} \tag{2.77}$$

Розшукуваний сплайн є сумою по всіх вузлах

$$s(x) = \sum_{k=-1}^{N+1} s_k B_k(x) \tag{2.78}$$

базових сплайнів з відповідними множниками – сплайн-коефіцієнтами $\{s_k; k = -1 \dots N+1\}$.

В кожному вузлі з усієї суми відмінними від нуля можуть бути лише три члени

$$s(x_m) = B_{m-1}(x_m) s_{m-1} + s_m + B_{m+1}(x_m) s_{m+1}, \tag{2.79}$$

тому умови інтерполяції утворюють тридіагональну систему

$$B_{m-1}(x_m) s_{m-1} + s_m + B_{m+1}(x_m) s_{m+1} = y_m, \tag{2.80}$$

що має бути доповнена кінцевими умовами

$$\begin{cases} B''_{-1}(x_0) s_{-1} + B''_0(x_0) s_0 + B''_{+1}(x_0) s_{+1} = 0 \\ B''_{N-1}(x_N) s_{N-1} + B''_N(x_N) s_N + B''_{N+1}(x_N) s_{N+1} = 0 \end{cases} \tag{2.81}$$

Побудова базових сплайнів

Розглянемо тепер умови, за якими повинні визначатись коефіцієнти кубічних поліномів довільного базового сплайну, що має одиничне значення в одному з внутрішніх вузлів мережі. В загальному випадку такий внутрішній базовий сплайн має вигляд

$$B_k(x) = \begin{cases} a_{k,-2} + b_{k,-2}x + c_{k,-2}x^2 + d_{k,-2}x^3; & x_{k-2} \leqslant x \leqslant x_{k-1} \\ a_{k,-1} + b_{k,-1}x + c_{k,-1}x^2 + d_{k,-1}x^3; & x_{k-1} \leqslant x \leqslant x_k \\ a_{k,0} + b_{k,0}x + c_{k,0}x^2 + d_{k,0}x^3; & x_k \leqslant x \leqslant x_{k+1} \\ a_{k,1} + b_{k,1}x + c_{k,1}x^2 + d_{k,1}x^3; & x_{k+1} \leqslant x \leqslant x_{k+2} \\ 0; & x \leqslant x_{k-2} \cup x \geqslant x_{k+2} \end{cases}. \tag{2.82}$$

В ньому присутні по 4 коефіцієнти до кожного 4 поліномів третього степеню – загалом 16 параметрів базового сплайну, що їх потрібно обчислити. Ці поліноми визначають властивості конкретного базового сплайну на чотирьох відрізках, що прилягають до заданого вузла (по два з кожного боку). За межами цих чотирьох проміжків базовий сплайн дорівнює нулю.

Базові сплайни для кінцевих вузлів, так само як і для додаткових уявних вузлів, можна побудувати, якщо доповнити мережу ще трьома уявними вузлами на кожному з кінців, а саме $x_{-3}, x_{-2}, x_{-1}, x_{N+1}, x_{N+2}, x_{N+3}$. Тоді кожен з вузлів, для якого потрібно побудувати базовий сплайн, стає внутрішнім. Звичайно, властивості побудованого в такий спосіб сплайну врешті-решт не залежатимуть від того, як саме обирались уявні вузли, тому їх можна обирати довільно. Найпростішим вибором буде розташування трьох додаткових вузлів ліворуч на тій же відстані, що є між першим та нульовим вузлами і так само – на другому кінці області інтерполяції.

Отже, задача про побудову системи базових сплайнів є задачею про побудову $N+3$ сплайнів на множині

$$\{x_k; k=-3\ldots N+3\}=$$
$$\left\{\begin{array}{lll} x_{-3}\,(=x_{-2}-h_0)\,, & x_{-2}\,(=x_{-1}-h_0)\,, & x_{-1}\,(=x_0-h_0)\,, \\ x_0, & x_1\,(=x_0+h_0)\,,*\ldots, & \\ x_{N-1}\,(=x_N-h_{N-1})\,, & x_N, & \\ x_{N+1}\,(=x_N+h_{N-1})\,, & x_{N+2}\,(=x_{N+1}+h_{N-1})\,, & x_{N+3}\,(=x_{N+2}+h_{N-1}) \end{array}\right\} \tag{2.83}$$

Кожен з базових сплайнів повинен задовольняти всі умови неперервності, крім того, він має в одному вузлі (власному центрі) одиничне значення і обертається на нуль разом з двома похідними в вузлах, що на два кроки відстоять від центру.

Умови неперервності доцільно розглядати по частинах, одразу спрощуючи загальний вигляд базового сплайну.

В першу чергу звернімо увагу на кінцеві точки відрізку нетривіальності - x_{k-2}, x_{k+2}. В кожній з цих точок сплайн має обертатись на нуль разом з двома похідними. Поліном третього степеню, що в деякій точці дорівнює нулю разом з двома похідними, має загальний вигляд $A\,(x-x_r)^3$, тому загальний вигляд для базового сплайну можна одразу спростити до такого

$$B_k\,(x)=\left\{\begin{array}{l} f_k\,(x-x_{k-2})^3\,; x_{k-2}\leqslant x\leqslant x_{k-1} \\ a_{k,-1}+b_{k,-1}x+c_{k,-1}x^2+d_{k,-1}x^3; x_{k-1}\leqslant x\leqslant x_k \\ a_{k,0}+b_{k,0}x+c_{k,0}x^2+d_{k,0}x^3; x_k\leqslant x\leqslant x_{k+1} \\ g_k\,(x_{k+2}-x)^3\,; x_{k+1}\leqslant x\leqslant x_{k+2} \end{array}\right., \tag{2.84}$$

що має вже на шість вільних параметрів менше.

Для двох центральних проміжків замінимо поліноми по степенях координати на поліноми по степенях відхилення від центрального вузла $x-x_k$. Тоді загальний вираз для базового сплайну

$$B_k\,(x)=\left\{\begin{array}{l} f_k\,(x-x_{k-2})^3\,; x_{k-2}\leqslant x\leqslant x_{k-1} \\ 1+a_k\,(x-x_k)+b_k\,(x-x_k)^2+c_k\,(x-x_k)^3\,; x_{k-1}\leqslant x\leqslant x_k \\ 1+a_k\,(x-x_k)+b_k\,(x-x_k)^2+d_k\,(x-x_k)^3\,; x_k\leqslant x\leqslant x_{k+1} \\ g_k\,(x_{k+2}-x)^3\,; x_{k+1}\leqslant x\leqslant x_{k+2} \end{array}\right. \tag{2.85}$$

буде одночасно враховувати і потрібне значення сплайну в центральному вузлі і умови неперервності разом з двома похідними в цьому вузлі і за рахунок цього кількість вільних параметрів зменшиться ще на чотири. Як наслідок, залишаються невизначеними 6 коефіцієнтів, що для них маємо по три рівняння неперервності в двох проміжних вузлах x_{k-1}, x_{k+1}. Оскільки кількість рівнянь співпадає з кількістю невідомих, слід чекати, що розв'язок можна буде знайти, звичайно, якщо ці рівняння сумісні.

Запишемо систему рівнянь, що залишилися нерозв'язаними

$$\left\{\begin{array}{ll} B\,(x_{k-1})\,, & f_k\,(x_{k-1}-x_{k-2})^3=1+a_k\,(x_{k-1}-x_k)+b_k\,(x_{k-1}-x_k)^2+c_k\,(x_{k-1}-x_k)^3 \\ B'\,(x_{k-1})\,, & 3f_k\,(x_{k-1}-x_{k-2})^2=a_k+2b_k\,(x_{k-1}-x_k)+3c_k\,(x_{k-1}-x_k)^2 \\ B''\,(x_{k-1})\,, & 6f_k\,(x_{k-1}-x_{k-2})^2=2b_k+6c_k\,(x_{k-1}-x_k) \\ B\,(x_{k+1})\,, & g_k\,(x_{k+2}-x_{k+1})^3=1+a_k\,(x_{k+1}-x_k)+b_k\,(x_{k+1}-x_k)^2+d_k\,(x_{k+1}-x_k)^3 \\ B'\,(x_{k+1})\,, & -3g_k\,(x_{k+2}-x_{k+1})^2=a_k+2b_k\,(x_{k+1}-x_k)+3d_k\,(x_{k+1}-x_k)^2 \\ B''\,(x_{k+1})\,, & 6g_k\,(x_{k+2}-x_{k+1})=2b_k+6d_k\,(x_{k+1}-x_k) \end{array}\right. \tag{2.86}$$

Визначник цієї системи є

$$\Delta = (x_{k+1} - x_k)(x_{k+2} - x_k)(x_{k+2} + x_{k+1} - 2x_k) + (x_k - x_{k-1})(x_k - x_{k-2})(2x_k - x_{k-1} - x_{k-2}). \tag{2.87}$$

Він не обертається на нуль, якщо тільки не співпадають значення координат якої-небудь пари сусідніх вузлів. Дійсно, якщо вважати відстані між сусідніми вузлами $h_k = x_{k+1} - x_k > 0$ додатними, визначник можна подати у вигляді

$$\Delta = h_k(h_{k+1} + h_k)(h_{k+1} + 2h_k) + h_{k-1}(h_{k-1} + h_{k-2})(2h_{k-1} + h_{k-2}), \tag{2.88}$$

що вміщує тільки суму додатних величин і тому не обертається на нуль.

Подальша побудова базових сплайнів вимагає розв'язування цієї системи рівнянь для кожного вузла. Це можна робити аналітично, з подальшою підстановкою числових значень в вирази для кожного коефіцієнта, але простіше навіть - програмно, для кожного набору чисел окремо.

Незалежно від способу реалізації процесу обчислень конкретні значення коефіцієнтів, що залишаються поки що невизначеними, залежатимуть від співвідношень між довжинами проміжків, тому результатом побудови системи базових сплайнів буде розрахунок цих коефіцієнтів для кожного базового сплайну.

Останнє міркування дозволяє визначити попередньо, що саме мають уявляти собою, з програмної точки зору, базові сплайни. Для кожного вузла мережі, в тому разі і для двох уявних вузлів з номерами $-1, N+1$, базовий сплайн можна повністю охарактеризувати шістьма коефіцієнтами $B \to \{a, b, c, d, f, g\}$. Відповідно набір базових сплайнів має бути масивом таких послідовностей.

Використання системи базових сплайнів

Припустимо тепер, що задачу розрахунку сплайну вже розв'язано, тобто відомі всі базові сплайни і всі сплайн-коефіцієнти і розглянемо, як саме мають розраховуватись значення сплайну в конкретній точці. Припустимо навіть, що нам вже відомо, до якого саме проміжку $x_m \leqslant x < x_{m+1}$ належить значення аргументу, що саме для нього потрібно розрахувати значення сплайну. Тоді розрахунок значення має відбуватись за виразом

$$s(x) = B_{m-1}(x)\, s_{m-1} + B_m(x)\, s_m + B_{m+1}(x)\, s_{m+1} + B_{m+2}(x)\, s_{m+2}, \tag{2.89}$$

в якому замість загальних позначень потрібно підставити конкретні вирази для поліномів. Наприклад, для першого члена маємо

$$B_{m-1}(x) = \begin{cases} f_{m-1}(x - x_{m-1-2})^3; & x_{m-1-2} \leqslant x \leqslant x_{m-1-1} \\ 1 + a_{m-1}(x - x_{m-1}) + b_{m-1}(x - x_{m-1})^2 + c_{m-1}(x - x_{m-1})^3; & x_{m-1-1} \leqslant x \leqslant x_{m-1} \\ 1 + a_{m-1}(x - x_{m-1}) + b_{m-1}(x - x_{m-1})^2 + d_{m-1}(x - x_{m-1})^3; & x_{m-1} \leqslant x \leqslant x_{m-1+1} \\ g_{m-1}(x_{m-1+2} - x)^3; & x_{m-1+1} \leqslant x \leqslant x_{m-1+2} \end{cases} \tag{2.90}$$

і потрібна умова виконується тільки для останнього рядка. Подібно до цього можна визначити, які саме конкретні вирази будуть використані і для всіх інших базових сплайнів, тож отримаємо такий вираз для значення сплайну

$$\begin{array}{l} s\left(x\right)=s_{m-1}g_{m-1}\left(x_{m+1}-x\right)^{3}+ \\ s_{m}\left(1+a_{m}\left(x-x_{m}\right)+b_{m}\left(x-x_{m}\right)^{2}+d_{m}\left(x-x_{m}\right)^{3}\right)+ \\ s_{m+1}\left(1+a_{m+1}\left(x-x_{m+1}\right)+b_{m+1}\left(x-x_{m+1}\right)^{2}+c_{m+1}\left(x-x_{m+1}\right)^{3}\right)+ \\ s_{m+2}f_{m+2}\left(x-x_{m}\right)^{3} \end{array} \tag{2.91}$$

В цьому виразі залишається багато подібних членів – різних степенів аргументу. Оскільки обчислення значень сплайну є звичайно більш масовою операцією, ніж розрахунок коефіцієнтів сплайну, доцільно перерахувати за значеннями базових сплайнів та сплайн-коефіцієнтів значення коефіцієнтів розраховуваного сплайну, надавши їм зручного для розрахунків вигляду.

Міркування про співвимірність коефіцієнтів, що мають використовуватись в обчисленнях (подібні до таких для сплайну на рівномірній гратці) призводять до необхідності заміни аргументу на відносну величину

$$z=\frac{x-x_{k}}{x_{k+1}-x_{k}}=\frac{x-x_{k}}{h_{k}}, \tag{2.92}$$

що знаходиться в межах від нуля до одиниці і для якої вираз для значення сплайну на відрізку має дорівнювати

$$s\left(x\right):x_{k}\leqslant x<x_{k+1}=\left|z=\frac{x-x_{k}}{h_{k}}\right|=y_{k}+z\cdot\left(d_{k}+z\cdot\left(dd_{k}+z\cdot t_{k}\right)\right) \tag{2.93}$$

Обчислення за цим виразом можуть здійснюватись, якщо для кожного відрізку зберігатиметься набір параметрів $\{x,y,h,d,dd,t\}$. Саме ці величини і потрібно обчислити за значеннями коефіцієнтів базових сплайнів та сплайн-коефіцієнтів.

Вирази для розрахунку параметрів сплайну отримуються, якщо в вираз для значення суми базових сплайнів підставити вираз для залежності значення аргументу від відносного аргументу і порівняти члени при однакових степенях

Підстановка дає

$$\begin{array}{l} x=x_{m}+zh_{m} \\ s\left(x\right)=s_{m-1}g_{m-1}h_{m}^{3}\left(1-z\right)^{3}+s_{m}\left(1+a_{m}h_{m}z+b_{m}h_{m}^{2}z^{2}+d_{m}h_{m}^{3}z^{3}\right)+ \\ s_{m+1}\left(1+a_{m+1}h_{m}\left(1-z\right)+b_{m+1}h_{m}^{2}\left(1-z\right)^{2}+c_{m+1}h_{m}^{3}\left(1-z\right)^{3}\right)+ \\ s_{m+2}f_{m+2}h_{m}^{3}z^{3} \end{array} \tag{2.94}$$

і для коефіцієнтів сплайну маємо розрахункові вирази

$$\begin{array}{l} d_{m}=-3s_{m-1}g_{m-1}h_{m}^{3}+s_{m}a_{m}h_{m}+s_{m+1}\left(-a_{m+1}h_{m}-2b_{m+1}h_{m}^{2}-3c_{m+1}h_{m}^{3}\right) \\ dd_{m}=3s_{m-1}g_{m-1}h_{m}^{3}+s_{m}b_{m}h_{m}^{2}+s_{m+1}\left(b_{m+1}h_{m}^{2}+3c_{m+1}h_{m}^{3}\right) \\ t_{m}=-s_{m-1}g_{m-1}h_{m}^{3}+s_{m}d_{m}h_{m}^{3}-s_{m+1}c_{m+1}h_{m}^{3}+s_{m+2}f_{m+2}h_{m}^{3} \end{array}. \tag{2.95}$$

Отже, кінцевою метою задачі побудови базових сплайнів є розрахунок за цими формулами коефіцієнтів інтерполяційного сплайну.

Алгоритм обчислення базового сплайну

Оскільки в загальному випадку додаткового скорочення коефіцієнтів чекати даремно, кількість обчислювальних дій, потрібних для розрахунку коефіцієнтів базових сплайнів, практично однакова що для аналітичних виразів, що для методу Гауса. Єдина аналітична дія, що її доцільно ще зробити – підготовка системи до вигляду, зручного для використання методу Гауса. З цією метою позначимо $h_m = x_{m+1} - x_m$ довжину кожного відрізку і перепишемо систему рівнянь для знаходження параметрів базового сплайна у вигляді

$$\left\{\begin{array}{lcr} h_{k-1}a_k - h_{k-1}^2 b_k + h_{k-1}^3 c_k + f_k h_{k-2}^3 & = & 1 \\ a_k - 2h_{k-1}b_k + 3h_{k-1}^2 c_k - 3h_{k-2}^2 f_k & = & 0 \\ b_k - 3c_k h_{k-1} - 3f_k h_{k-2} & = & 0 \\ a_k h_k + b_k h_k^2 + d_k h_k^3 - g_k h_{k+1}^3 & = & -1 \\ a_k + 2b_k h_k + 3d_k h_k^2 + 3g_k h_{k+1}^2 & = & 0 \\ b_k + 3d_k h_k - 3g_k h_{k+1} & = & 0 \end{array}\right. . \tag{2.96}$$

Тепер можна визначити послідовність для коефіцієнтів, що розраховуватимуться за методом Гауса, наприклад, таку $(a_k, b_k, c_k, d_k, f_k, g_k)$ і побудувати матрицю рівнянь

$$\left(\begin{array}{cccccccc} h_{k-1} & -h_{k-1}^2 & h_{k-1}^3 & 0 & h_{k-2}^3 & 0 & 1 \\ 1 & -2h_{k-1} & 3h_{k-1}^2 & 0 & -3h_{k-2}^2 & 0 & 0 \\ 0 & 1 & -3h_{k-1} & 0 & -3h_{k-2} & 0 & 0 \\ h_k & h_k^2 & 0 & h_k^3 & 0 & -h_{k+1}^3 & -1 \\ 1 & 2h_k & 0 & 3h_k^2 & 0 & 3h_{k+1}^2 & 0 \\ 0 & 1 & 0 & 3h_k & 0 & -3h_{k+1} & 0 \end{array}\right) . \tag{2.97}$$

Вигляд цієї системи підказує, що процедурі, що розраховуватиме набір коефіцієнтів базових сплайнів, потрібно передавати даними не всю матрицю, а тільки параметри відрізків – довжини проміжків, що їх можна позначити, як h_{2l}, h_l, h_r, h_{2r} відповідно. Тоді номер проміжку зникає з переліку аргументів процедури і вона може використовуватись одна і та ж сама для базових сплайнів в усіх вузлах.

Визначення сплайн-коефіцієнтів

Розглянемо тепер систему рівнянь, за якими визначаються сплайн-коефіцієнти. Вона поділяється на систему рівнянь інтерполяції

$$s\left(x_k\right) = y_k;\ k = 0 \ldots N \tag{2.98}$$

та додаткові умови,

$$s''\left(x_0\right) = 0; s''\left(x_N\right) = 0 \tag{2.99}$$

Кожне з рівнянь інтерполяції є тричленним рекурентним співвідношенням

$$s\left(x_k\right) = s_{k-1}B_{k-1}\left(x_k\right) + s_k + s_{k+1}B_{k+1}\left(x_k\right) = y_k;\ k = 0 \ldots N, \tag{2.100}$$

а от додаткові умови теж містять по три члени, що заважає безпосередньо використовувати метод прогонки. Дійсно, вираз для другої похідної в вузлах вміщує внески від трьох сусідніх базових сплайнів

$$s''(x_m) = s_{m-1}B''_{m-1}(x_m) + s_m B''_m(x_m) + s_{m+1}B''_{m+1}(x_m); \tag{2.101}$$

Тому додатковими умовами є

$$s''(x_0) = s_{-1}B''_{-1}(x_0) + s_0 B''_0(x_0) + s_1 B''_1(x_0); \tag{2.102}$$

$$s''(x_N) = s_{N-1}B''_{N-1}(x_N) + s_N B''_N(x_N) + s_{N+1}B''_{N+1}(x_N); \tag{2.103}$$

Для методу прогонки потрібно, аби кожне з кінцевих рівнянь вміщувало тільки два члени (інакше матриця системи не буде тридіагональною). Якщо першу з додаткових умов доповнити першим з рівнянь інтерполяції, в системі

$$\begin{cases} s_{-1}B_{-1}(x_0) + s_0 + s_1 B_1(x_0) & = y_0 \\ s_{-1}B''_{-1}(x_0) + s_0 B''_0(x_0) + s_1 B''_1(x_0) & = 0 \end{cases} \tag{2.104}$$

можна методом підстановки вилучити з першого рівняння змінну s_{-1}

$$\begin{cases} s_0\left(1 - B''_0(x_0)\frac{B_{-1}(x_0)}{B''_{-1}(x_0)}\right) + s_1\left(B_{+1}(x_0) - B''_1(x_0)\frac{B_{-1}(x_0)}{B''_{-1}(x_0)}\right) = y_0 \\ s_{-1} = -\frac{s_0 B''_0(x_0) + s_1 B''_1(x_0)}{B''_{-1}(x_0)} \end{cases}, \tag{2.105}$$

тоді кінцевим рівнянням для методу прогонки буде

$$s_0\left(1 - B''_0(x_0)\frac{B_{-1}(x_0)}{B''_{-1}(x_0)}\right) + s_1\left(B_{+1}(x_0) - B''_1(x_0)\frac{B_{-1}(x_0)}{B''_{-1}(x_0)}\right) = y_0, \tag{2.106}$$

а рівняння

$$s_{-1} = -\frac{s_0 B''_0(x_0) + s_1 B''_1(x_0)}{B''_{-1}(x_0)} \tag{2.107}$$

визначатиме сплайн-коефіцієнт для додаткового вузла, що його можна буде обчислити вже після знаходження основної системи сплайн-коефіцієнтів.

Так само і на другому кінці з системи

$$\begin{cases} s_{N-1}B_{N-1}(x_N) + s_N + s_{N+1}B_{N+1}(x_N) = y_N \\ s_{-1}B''_{-1}(x_0) + s_0 B''_0(x_0) + s_1 B''_1(x_0) = 0 \end{cases} \tag{2.108}$$

підстановкою вилучаємо s_{N+1} і отримуємо

$$\begin{cases} s_N\left(1 - B''_N(x_N)\frac{B_{N-1}(x_N)}{B''_{N-1}(x_N)}\right) + s_{N+1}\left(B_{N+1}(x_N) - B''_{N+1}(x_N)\frac{B_{N-1}(x_N)}{B''_{N-1}(x_N)}\right) = y_N \\ s_{N-1} = -\frac{s_N B''_N(x_N) + s_{N+1}B''_{N+1}(x_N)}{B''_{N-1}(x_N)} \end{cases} \tag{2.109}$$

або окремо рівняння для визначення додаткового сплайн-коефіцієнту

$$s_{N+1} = -\frac{s_N B''_N(x_N) + s_{N+1}B''_{N+1}(x_N)}{B''_{N-1}(x_N)} \tag{2.110}$$

та кінцеве двочленне рівняння для метода прогонки

$$s_N\left(1-B''_N(x_N)\frac{B_{N-1}(x_N)}{B''_{N-1}(x_N)}\right)+s_{N+1}\left(B_{N+1}(x_N)-B''_{N+1}(x_N)\frac{B_{N-1}(x_N)}{B''_{N-1}(x_N)}\right)=y_N \tag{2.111}$$

Метод прогонки, що його потрібно застосовувати для обчислення значень сплайн-коефіцієнтів, є вже повністю стандартним.

Обчислення значень інтерполяційного сплайну.

В задачах з рівномірною граткою вибір відрізку, до якого відноситься значення аргументу, для котрого потрібно розрахувати значення сплайну, здійснюється досить просто, оскільки номер відрізку є цілою частиною результату ділення значення аргументу на крок мережі.

Нерівномірна мережа потребує набагато складніших методів пошуку відрізка. На жаль, найпростіші з цих методів не є найефективнішими.

Дійсно, найпростішим є метод послідовного перебору, за яким для кожного вузла послідовно перевіряються нерівності $x_k \leqslant x < x_{k+1}$, аж поки не буде знайдено потрібний проміжок. Залежно від значення аргументу може знадобитись від одного до N кроків до завершення задачі пошуку. Якщо потрібні значення розподіляються рівномірно, в середньому знадобиться $N/2$ дій на кожне обчислення і задача пошуку перетворюється на задачу лінійної складності.

Більш ефективним є метод бінарного пошуку. Він полягає в тому, що першим кандидатом на перевірку, незалежно від потрібного значення, обирається середній відрізок. Якщо потрібне значення менше за координату лівого кінця, тобто $x < x_c$, пошук продовжується ліворуч, і так само за кандидата на потрібний відрізок обирається середній відрізок лівої частини області, тобто $c_{new} = c/2$, якщо ж ця умова на справджується, здійснюється порівняння з правим кінцем відрізку і якщо потрібне значення аргументу більше $x > x_c + h_c$, пошук продовжується на середньому відрізку праворуч $c_{new} = (N-c)/2$. Цей алгоритм потребує, в найгіршому випадку, $\log_2 N$ дій, а в середньому трохи більше за половину від цього числа, тобто має замість лінійної логарифмічну складність. Наприклад, для гратки в $1000 \approx 2^{10}$ вузлів необхідно в середньому 5-6 кроків для знаходження необхідного проміжку.

Додаткового прискорення пошуку можна отримати, використовуючи уявлення про співвідношення між значенням аргументу та середнім кроком $h_a = \frac{x_N - x_0}{N}$. Якщо припустити, що мережа близька до рівномірної, гарною оцінкою для очікуваного номера проміжку буде ціла частина $\left[\frac{x-x_0}{h_a}\right]$. Якщо потрібне значення не належить до цього проміжку, величина $\left[\frac{x-x_k}{h_a}\right]$ може бути гарною оцінкою потрібного зсуву на мережі. Такий метод пошуку буде мати таку ж саму логарифмічну складність, що й бінарний, але з дещо більшим значенням основи логарифму. Для наведеного прикладу в 1000 проміжків його використання зменшуватиме середню потрібну кількість кроків пошуку до 2 або 3, тобто програма в цілому буде працювати в півтора – два рази швидше.

Програмна реалізація

Програма побудови сплайнів для нерівномірної мережі є достатньо складною, тому її реалізація наведена тільки мовою C++, оскільки інші мови програмування не надто зручні для складних програм.

Програма поділена на окремі модулі. Модуль hspline.h вміщує майже тільки оголошення потрібних структур, відповідні методи винесено в модуль spline.c . Цей останній транслюється окремо від головного модуля. Модуль base.h вміщує і оголошення і імплементації процедур, потрібних для побудови базових сплайнів. Під час трансляції spline.c всі процедури цього модулю монтуються в spline.o. Головний модуль main.c тільки викликає необхідні структури, що описані в hspline.h, тому він є достатньо компактним.

Трансляція програми може здійснюватись під Unix стандартним транслятором GCC, або під Windows системою Bloodshed Dev-C++ , що є імплементацією того ж транслятора. Програма розрахована на використання аргументів командного рядка – запуск

`b-spline < data.txt`

відтворює результат розрахунку на екрані, Його можна також спрямувати в файл результату, використавши командний рядок у вигляді

`b-spline < data.txt >res.txt`.

```
File main.c
#include "hspline.h"
int main(){
        spline s=spline();
 /* створення сплайну, підготовка циклу відтворення сплайну
    й виведення результату*/
        double step=(s.xn-s.x0)/s.len/20;  double res;
        for(double x=s.x0;x<(s.xn+step);x+=step){
           res=s.val(x);
           cout<<x<<"\t"<<res<<"\n";
        };
return(0); }

File hspline.h

#include <iostream> #define maxdim 1000

struct piece{
        double xk,yk,dk,ddk,tk,hk,xk1;
        /*початок відрізку, значення на початку,
        перший, другий та третій коефіцієнти
        крок на відрізку та кінцева координата*/
        piece *left, *right;
        /*вказівники на середину масиву ліворуч та праворуч*/
  piece(){left=right=0;}
  /*ініціалізація вичищає посилання*/
  void init(double x,double y,double d,double dd,double t,double h){
        xk=x;yk=y;dk=d;ddk=dd;tk=t;hk=h;xk1=x+h;
      /*задання всіх значень*/
        } ;
 double value(double x){
  double z=(x-xk)/hk;
  return(yk+z*(dk+z*(ddk+z*tk)));
```

```
  /*значення поліному*/
};
 double val(double x){
  if(x<xk)    {double res=(left==0)?yk:left->val(x);
  /*якщо потрібне менше значення аргументу і лівіше є відрізок -
    брати значення звідти, інакше значення на лівому кінці цього відрізку
  */
    return res; };
  if(x>=xk+hk){double res=(right==0)?value(xk1):right->val(x);
  /*якщо потрібне більше значення аргументу і правіше є відрізок -
    брати значення звідти, інакше значення на правому кінці цього відрізку
  */
   return res;};
     /*якщо потрібне значення аргументу з цього відрізку -
     розрахувати потрібне */
  return(value(x));
};; };

struct spline{
        piece *v; /* майбутній масив відрізків */
        piece *beam; /* майбутня середина дерева відрізків */
        double yn,xn,x0,y0; /*кінцеві точки області*/
        int len;   /*довжина області*/
        spline();  /* оголошення конструктора; реалізація - в окремому модулі*/
 double value(double x); /*розрахунок значення з пошуком методом перебору*/
 double val(double x){
  /*розрахунок значення з пошуком методом бінарного пошуку*/
     if(x<=x0){return y0;};
     if(x>=xn){return yn;};
  /*переадресація бінарного пошуку в процедуру пошуку структури piece*/
     return(beam->val(x));
 };
 void sort(int kmin,int kc,int kmax);
  /*процедура сортування для бінарного пошуку */
};

File base.h

#include <math.h> #define mind 1e-12 struct coeff{ //array with
indexes from min up to max
        double *v;
        int max,min;
        coeff(int mn, int mx){max=mx;min=mn;v=new double[max-min+1];};
 double& operator[](int k){return v[k-min];} //one need boudary control:)
};

struct gauss6{ /*має навчитись розв'язувати систему рівнянь*/
        double d[6][7];//equation coeff's
        double res[6]; //must turn be to roots
  void showm(){
```

```
   /*тільки для відлагодження; друкує матрицю*/
for(int n=0;n<6;n++){
        for(int nn=0;nn<7;nn++){
cout<<d[n][nn]<<"|";};cout<<"\n"; }cout<<"\n";
  } ;
  void showr(){
/*тільки для відлагодження; друкує результат*/ for(int
n=0;n<6;n++){cout<<res[n]<<"|";};
        cout<<"\n\n";
  };

  void forw(int k){
  //прямий хід методу Гауса
        double tmp=fabs(d[k][k]);int nt=k;
      for(int n=k;n<6;n++){
        if(fabs(d[n][k])>tmp){nt=n;tmp=fabs(d[n][k]);};
        }//leader row finding
      if(nt!=k){
        for(int nn=k;nn<7;nn++){
         tmp=d[k][nn];d[k][nn]=d[nt][nn];d[nt][nn]=tmp;
         };//row swap
      };//
      tmp=d[k][k];
      for(int nn=k;nn<7;nn++){d[k][nn]/=tmp; };//leader row normalization
      for(int n=k+1;n<6;n++){//for all nexted rows
         tmp=d[n][k]; d[n][k]=0;
         for(int nn=k+1;nn<7;nn++){d[n][nn]-=tmp*d[k][nn];};
                //linear combination with leader row
         } ;

      if(k<6){forw(k+1);} //to next minor matrix
   };//end forw


   double* solver(double xl2,double xl,double xr,double xr2){
//matrix build   повністю відтворює рівняння, що задають матрицю

     d[0][0]=xl;d[0][1]=-xl*xl;d[0][2]=xl*xl*xl;d[0][3]=0;
           d[0][4]=xl2*xl2*xl2;d[0][5]=0;d[0][6]=1;
     d[1][0]=1;d[1][1]=-2*xl;d[1][2]=3*xl*xl;d[1][3]=0;d[1][4]=-3*xl2*xl2;
        d[1][5]=0;d[1][6]=0;
     d[2][0]=0;d[2][1]=1;d[2][2]=-3*xl;d[2][3]=0;d[2][4]=-3*xl2;
        d[2][5]=0;d[2][6]=0;
     d[3][0]=xr;d[3][1]=xr*xr;d[3][2]=0;d[3][3]=xr*xr*xr;d[3][4]=0;
        d[3][5]=-xr2*xr2*xr2;d[3][6]=-1;
     d[4][0]=1;d[4][1]=2*xr;d[4][2]=0;d[4][3]=3*xr*xr; d[4][4]= 0;
        d[4][5]=3*xr2*xr2;d[4][6]=0;
     d[5][0]=0;d[5][1]=1;d[5][2]=0;d[5][3]=3*xr;d[5][4]=0;
        d[5][5]=-3*xr2;d[5][6]=0;
//turn to three-angle form
```

```
     forw(0);
//result generation зворотній хід методу Гауса
        for(int n=5;n>=0;n--){
        res[n]=d[n][6];
        for(int nn=n+1;nn<6;nn++){res[n]-=res[nn]*d[n][nn];};
       // res[n]=(floor(res[n]/mind)*mind);
        }
   };//end solver
};

struct base{
 double a,b,c,d,f,g; //piece coefficients
// g(x-xl2)^3  :xl2<=x<=xl
// 1+ax+bx^2+dx^3 :xl<=x<=0
// 1+ax+bx^2+cx^3 :0<=x<=xr
// f(xr2-x)^3  :xr<=x<=xr2
 double vl,vr,vddl,vddc,vddr;
// vl -function value, vddl - double derivative value for x=-xl;
// vr -function value, vddr - double derivative value for x=xr;
// vddc - double derivative value for x=0;
void solve(double* x){
   double norm=(x[2]-x[-2])/4;
//normalization needed for matrix balancing
  gauss6 gs;
/*виклик методу Гауса для поточних параметрів відрізку*/
 gs.solver((x[-1]-x[-2]),(x[0]-x[-1]),(x[1]-x[0]),(x[2]-x[1]));
 /*використання результату розрахунку за методом Гауса*/
 a=gs.res[0];b=gs.res[1];c=gs.res[2];d=gs.res[3];g=gs.res[4];f=gs.res[5];
/*розрахунок значень в вузлах - підготовка до матоду прогонки*/
  vl=g*(x[-1]-x[-2])*(x[-1]-x[-2])*(x[-1]-x[-2]);vddl=6*g*(x[-1]-x[-2]);
  vddc=2*b;
  vr=f*(x[2]-x[1])*(x[2]-x[1])*(x[2]-x[1]);vddr=6*f*(x[2]-x[1]);
 };
/*пустий конструктор потрібний для створення масиву*/
 base(){};
};

struct basis{
  base *bs;
  int len;
 basis(int n,coeff x){
/*створення масиву базових сплайнів*/
 len=n;
 bs=new base[len+3];
 };
/*цей оператор забезпечує нормальну індексацію елементів масиву*/
 base& operator[](int n){return(bs[n+1]);}
};

File spline.c
```

```
#include "hspline.h"
#include "base.h"

spline::spline(){
 coeff x=coeff(-3,maxdim+3);/*масив аргументів, звільняється по закінченні
  процедури */
 double *y=new double[maxdim];/*масив значень, теж звільняється */
 /*зачитування таблиці функції; закінчується, коли вичерпається місце
   або коли вичерпається таблиця */
  for(len=0;(len<maxdim)&&(cin>>x[len]>>y[len]);len++);
  /*останній len вказував на пустий рядок*/
  len--;
  /*тепер len дорівнює кількості проміжків*/
        //кінцеві значення
        x0=x[0];y0=y[0];xn=x[len];yn=y[len];
        //додаткові вузли для базових сплайнів
  double h=x[1]-x[0];x[-1]=x[0]-h;x[-2]=x[-1]-h;x[-3]=x[-2]-h;
  h=x[len]-x[len-1];
  x[len+1]=x[len]+h;x[len+2]=x[len+1]+h;x[len+3]=x[len+2]+h;

  basis b=*(new basis(len,x)); // створення масиву базових сплайнів
  for(int k=-1;k<=len+1;k++){   //basis components calculation
        b[k].solve(&x[k]);  //виклик процедури з модулю base.h
        };

    // sweep method:
// recursion coefficients
  double *alpha= new double[len];double *beta= new double[len];
  double tmp=1-b[len].vddc*b[len+1].vl/b[len+1].vddl;
  alpha[len-1]=(b[len-1].vr-b[len-1].vddr*b[len+1].vl/b[len+1].vddl)/tmp;
  beta[len-1]=y[len]/tmp;
  for(int k=len-1;k>0;k--){tmp=1+b[k+1].vl*alpha[k];
        alpha[k-1]=-b[k-1].vr/tmp;beta[k-1]=(y[k]-b[k+1].vl*beta[k])/tmp;
        };
//coefficients of spline seria
  coeff s=coeff(-1,len+2);
   tmp=1-b[0].vddc*b[-1].vr/b[-1].vddr+
               alpha[0]*(b[1].vl-b[1].vddl*b[-1].vr/b[-1].vddr);

  s[0]=(y[0]-beta[0]*(b[1].vl-b[1].vddl*b[-1].vr/b[-1].vddr))/tmp;
  for(int k=0;k<len;k++){
        s[k+1]=alpha[k]*s[k]+beta[k];};
//additional coeff's
  s[-1]=-(s[0]*b[0].vddc+s[1]*b[1].vddl)/b[-1].vddr;
  s[len+1]=-(s[len]*b[len].vddc+s[len-1]*b[len-1].vddr)/b[len+1].vddl;
  v=new piece[len];
  /*створення масиву коефіцієнтів сплайну та побудова бінарного дерева*/
//coeff's of spline pieces
  int kc=len/2;beam=&v[kc];sort(0,kc,len-1);
```

```
  double d,dd,t,h2,h3;
//recalculation from basis to piece coeff's
     //recalculation from basis to piece coeff's
   /*для кожного відрізку розраховуються його коефіцієнти*/
  for(int k=0;k<len;k++){
        h=x[k+1]-x[k];h2=h*h;h3=h2*h;
        d=-3*s[k-1]*b[k-1].f*h3+s[k]*b[k].a*h+
                s[k+1]*(b[k+1].a*h-2*b[k+1].b*h2+3*b[k+1].c*h3);
        dd=3*s[k-1]*b[k-1].f*h3+s[k]*b[k].b*h2+
                s[k+1]*(b[k+1].b*h2-3*b[k+1].c*h3);
        t=-s[k-1]*b[k-1].f*h3+s[k]*b[k].d*h3+
                s[k+1]*b[k+1].c*h3+s[k+2]*b[k+2].g*h3;
        v[k].init(x[k],y[k],d,dd,t,h);
                                                  };
}; //spline


 double spline::value(double x){
  if(x<x0){return y0;};
  if(x>=xn){return yn;};
  int kc=len/2;
  /*починається пошук з середнього проміжку*/
  for(;((kc>=0)&&(v[kc].xk>x));kc--);
  /*поки потрібне значення аргументу менше за початок відрізку,
  переходити на сусідній відрізок ліворуч
  по закінченні гарантовано, що v[kc].xk<=x
  */
  for(;((kc<len-1)&&(v[kc].xk1<x));kc++);
  /*поки потрібне значення аргументу більше за кінець відрізку,
  переходити на сусідній відрізок праворуч
  по закінченні гарантовано, що v[kc].xk1>=x
*/
       return(v[kc].value(x));
};

void spline::sort(int min,int k,int max){
  /*процедура сортування для бінарного пошуку */
  if(k>min){  /*тобто якщо існують нерозподілені вузли ліворуч
  призначити лівим вузлом середину відрізку від min до k-1*/
  int kl=(k+min)/2;v[k].left=&v[kl];sort(min,kl,k-1);};
  if(k<max){  /*тобто якщо існують нерозподілені вузли праворуч
  призначити правим вузлом середину відрізку від k+1 до max*/
  int kr=(k+max+1)/2;v[k].right=&v[kr];sort(k+1,kr,max);};
};
```

Тестування програми повинно перевірити її працездатність для типових даних. Попереднє тестування має перевірити працездатність для рівномірної гратки, оскільки в для неї можна створити набори даних, для яких досить просто вигля-

дають проміжні результати – розрахунок сплайн-коефіцієнтів. Типовим набором таких даних може бути табличка

0	0
0.1	0
0.2	0.25
0.3	1
0.4	0.25
0.5	0.0
0.6	0.25
0.7	1
0.8	0.25
0.9	0
1.0	0

Табл. 2.3: Тестові значення: пара базових сплайнів

В ній відтворені властивості двох базових сплайнів, тож перевірити працездатність програми можна, забезпечивши друк сплайн-коефіцієнтів.

Результуючий файл можна експортувати в будь-яку програму наукової графіки, наприклад, Origin. Результат такого експортування наведено нижче.

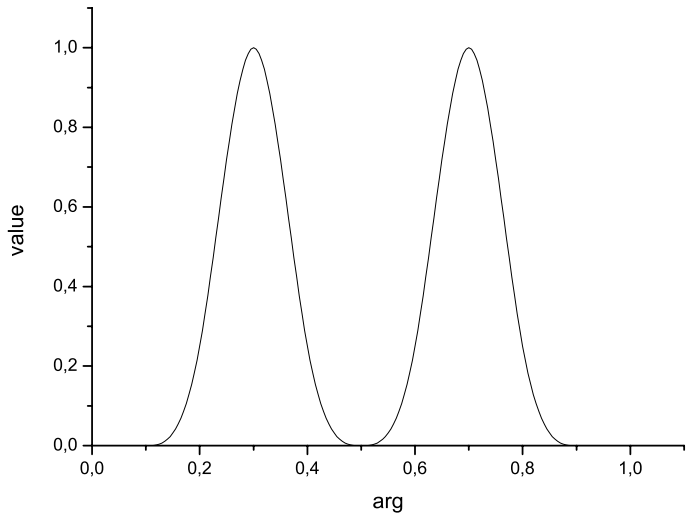


Рис. 2.3: Результат інтерполяції тестових значень.

2.4 Задача апроксимації

Задача апроксимації, подібно до задачі інтерполяції, є задачею побудови такої функції неперервної змінної, що в певний спосіб відбивала б властивості заданої функції дискретної змінної. Вона відрізняється від задачі інтерполяції в першу чергу тим, що клас функцій неперервної змінної занадто вузький, аби в ньому існував розв'язок задачі інтерполяції.

Найпростішим прикладом задачі апроксимації є побудова графіка залежності висоти від часу за результатами вимірювання послідовності положень тіла, що вільно падало. З теоретичних міркувань зрозуміло, що шукана залежність має бути поліномом другого степеню, разом з тим для послідовності більше ніж трьох вимірювань годі чекати, що виміряні значення будуть точно потрапляти на якусь параболу, оскільки вимірювання завжди здійснюються з певною похибкою. З іншого боку, шукати поліном, що точно проходив би через усі виміряні точки, нема ніякого сенсу, оскільки відхилення виміряних точок від параболи обумовлені не законом руху тіла, а розподілом випадкових величин – похибок вимірювання.

Важливою частиною задачі апроксимації є вибір критерію якості апроксимації – величини, значеннями якої оцінюватиметься, яка саме з множини функцій заданого класу краще за інші апроксимує задану послідовність значень. Інтуїтивний вибір критерію часто призводить до проблем в побудові апроксимуючої функції. За приклад такого неефективного критерію можна навести критерій мінімуму найбільшого відхилення, що задає досить складну для розв'язування систему нерівностей. Найчастіше критерій якості апроксимації узгоджується з фізичним змістом можливої похибки апроксимації.

Задача апроксимації найчастіше є частиною процесу обробки результатів вимірювань і полягає в побудові такої функції неперервної змінної, що, з одного боку, відповідає теоретичним уявленням про вимірюваний процес, а з другого боку, в найкращій спосіб наближає результати вимірювань.

Математична постановка задачі апроксимації містить такі вхідні дані:

- Функція дискретної змінної (таблиця результатів вимірювань);
- Клас функцій, що відповідає теоретичній моделі;
- Критерій якості апроксимації.

Функція дискретної змінної визначається своєю табличкою, або послідовністю пар значень „аргумент – функція" $\{x_k, y_k; k = 0 \dots N\}$ і досить часто додатково характеризується величиною похибки вимірювань. Найчастіше значення однієї з двох вимірюваних величин для кожного окремого вимірювання контролюються і всі є різними, навіть з урахуванням можливої похибки вимірювань – саме ця величина і обирається за аргумент функції дискретної змінної. Якщо ж в процесі вимірювань значення обох величин можуть співпадати (в межах похибки), то ставиться вже більш складна задача про кореляцію результатів вимірювань двох (або декількох) величин.

Клас функцій $f\,(x; a_0 \dots a_K)$ обирається за теоретичною моделлю досліджуваного явища і може бути яким завгодно, але з міркувань результативності задачі апроксимації його, можливо навіть шляхом відповідного перетворення змінних, намагаються звести до поліноміального. В усякому разі кількість параметрів $K+1$ у функцій цього класу завжди суттєво менша за кількість $N+1$ вузлів функції дискретної змінної. Якщо це не так, задача апроксимації втрачає зміст, з одного боку, а вимірювання не можна вважати за відповідні досліджуваному явищу, з другого.

2.4.1 Метод найменших квадратів

За критерій якості найчастіше обирається довжина вектору
$\{\varepsilon_m = f(x_m; a_0 \dots a_K) - y_m; m = 0 \dots N\}$ різниць значень теоретичної функції та результату вимірювань (метод найменших квадратів).

Величина

$$\Phi\left[f(x; a_0 \dots a_K), \{x_m, y_m\}\right] = \sum_{n=0}^{N} \varepsilon_n^2 = \sum_{n=0}^{N} \left(f(x_n; a_0 \dots a_K) - y_n\right)^2 \qquad (2.112)$$

має назву функціонал похибки апроксимації. Взагалі терміном „функціонал" позначають величини, що є відображенням на числову вісь не одного числа чи набору чисел, а математичного об'єкту більш складної природи – вектора, або, як в даному разі – функції. Точніше кажучи, цей термін використовується саме тоді, коли множина значень є частиною простору дійсних (чи комплексних) чисел, а область означення має більш складну природу, ніж такий же простір. Найчастіше цей термін використовують для відображень на числову вісь елементів векторного простору або простору функцій.

В задачах апроксимації клас функцій, в якому шукається розв'язок, має обмежену кількість параметрів, тому функціонал похибки перетворюється на функцію декількох змінних – параметрів, що відрізняють між собою примірники функцій розглядуваного класу. Вигляд цієї функції суттєво залежить від того, які саме величини обрано за параметри. Найпростіший приклад – клас функцій, що відповідають лінійній залежності однієї величини від іншої. Такі функції можна подавати у вигляді поліномів першого степеню $y = ax + b$, тоді параметрами будуть коефіцієнти полінома a, b, а можна використовувати канонічну форму прямої $\frac{x}{p} + \frac{y}{q} = 1$. Явний вигляд залежності функціонала похибки від параметрів буде суттєво відрізнятись, але і одна і інша функції двох змінних будуть відповідати одному й тому ж функціоналу.

Власне задача полягає в знаходженні такого набору параметрів апроксимуючої функції, для якого функціонал похибки має мінімум.

Функціонал похибки є невід'ємною величиною, тому завжди має мінімум, але в загальному випадку задача пошуку мінімуму може бути зовсім не тривіальною, адже множина можливих значень параметрів зовсім не обов'язково повинна бути, наприклад, зв'язною.

Якщо не брати до уваги особливості, що можуть виникати внаслідок обмеження на припустимі значення параметрів, мінімум функціоналу похибки досягатиметься на розв'язках системи

$$\left\{ \frac{\partial \Phi}{\partial a_m} = 0 \right., \qquad (2.113)$$

що має саме стільки рівнянь, скільки є параметрів апроксимуючої функції.

В необхідних випадках фізики завжди вдаються до різних штучних спотворень множини параметрів. Наприклад, при експериментальній перевірці закону Кулона в перелік параметрів, що характеризують силу, крім відстані між тілами додається степінь залежності, але в такому разі вважається, що степінь є не цілим, а дійсним числом. Тоді з загального виразу для сили

$$F(r; k, d) = \frac{k}{r^d} \qquad (2.114)$$

можна утворити шляхом заміни $r = r_0 e^z$ поліном

$$f(z; a, d) = \ln(F) = -dz + a \tag{2.115}$$

і за результатами апроксимації оцінювати значення величини d. Саме в такий спосіб і відбувається обробка результатів для прямої перевірки закону обернених квадратів.

Коли вже відомий розв'язок задачі апроксимації $f(x) = f(x; \tilde{a}_0 \dots \tilde{a}_K)$, значення функціоналу похибки на цьому розв'язкові

$$\tilde{\Phi} = \Phi\left[f\left(x; \tilde{a}_0 \dots \tilde{a}_K\right), \{x_m, y_m\}\right] \tag{2.116}$$

дає оцінку якості апроксимації. Загальна величина похибки апроксимації складається з власне похибки в побудові апроксимуючої функції та неусувної похибки вимірювань. Зрозуміло, що підвищенням степеню апроксимуючого полінома можна зменшити похибку апроксимації до нуля – це коли апроксимація перетворюється на інтерполяцію. Так само легко зрозуміти, що коли значення функціоналу похибки зменшується до суми квадратів вимірювальних похибок

$$\tilde{\Phi}_{\text{exp}} = \sum_{n=0}^{N} \sigma_n^2, \tag{2.117}$$

подальше покращення апроксимації не може бути змістовним. Звичайно вимірювальні похибки однакові для всіх вузлів, тому якість апроксимації можна оцінювати, як відношення середньої (на один вузол) похибки апроксимації до дисперсії вимірювань

$$\delta = \frac{\Phi\left[f\left(x; \tilde{a}_0 \dots \tilde{a}_K\right), \{x_m, y_m\}\right]}{(N+1)\,\sigma^2}. \tag{2.118}$$

Саме ця величина є мірою узгодженості теоретичної моделі з експериментом.

Інколи навіть узгодженість отримується не за рахунок покращення теоретичної моделі, а за рахунок необґрунтованого збільшення кількості параметрів апроксимуючої функції.

2.4.2 Поліноміальна апроксимація

Припустимо, що класом апроксимуючих функцій є поліноми заданого степеню

$$f(x; a_0 \dots a_K) \equiv P_K(x) = a_0 + a_1 x + \dots + a_K x^K = \sum_{k=0}^{K} a_k x^k. \tag{2.119}$$

Параметрами цих поліномів є їх коефіцієнти $\{a_0 \dots a_K\}$, тож залежність апроксимуючої функції від кожного з параметрів є лінійною.

Завдяки лінійній залежності від параметрів загальні рівняння апроксимації значно спрощуються. Дійсно, в функціоналі похибки

$$\begin{aligned} \Phi\left[f\left(x; a_0 \dots a_K\right), \{x_m, y_m\}\right] &= \sum_{n=0}^{N} \left(P_K(x_n) - y_n\right)^2 \\ &= \sum_{n=0}^{N} \left(a_0 + a_1 x_n + \dots + a_K x_n^K - y_n\right)^2 \end{aligned} \tag{2.120}$$

кожен з членів суми є квадратичною формою від параметрів, а сума квадратичних форм також є квадратичною формою від параметрів, а її похідні

$$\begin{array}{l}\frac{\partial}{\partial a_k}\Phi\left[f\left(x;a_0\ldots a_K\right),\{x_m,y_m\}\right]=\\ 2\sum\limits_{n=0}^{N}\left(a_0+a_1x_n+\ldots+a_Kx_n^K-y_n\right)\frac{\partial}{\partial a_k}\left(a_0+a_1x_n+\ldots+a_Kx_n^K-y_n\right);\\ k=0\ldots K\end{array}\tag{2.121}$$

є лінійними формами.

Отже, рівняння поліноміальної апроксимації

$$\begin{array}{l}\frac{\partial}{\partial a_k}\Phi\left[f\left(x;a_0\ldots a_K\right),\{x_m,y_m\}\right]=\\ 2\sum\limits_{n=0}^{N}\left(a_0+a_1x_n+\ldots+a_Kx_n^K-y_n\right)x_n^k=0;\\ k=0\ldots K\end{array}\tag{2.122}$$

є лінійною системою рівнянь.

Перепишемо ці рівняння, використовуючи перегрупування порядку обчислення суми

$$a_0\sum_{n=0}^{N}x_n^k+a_1\sum_{n=0}^{N}x_n^{k+1}+\ldots+a_K\sum_{n=0}^{N}x_n^{k+K}=\sum_{n=0}^{N}x_n^ky_n;k=0\ldots K\tag{2.123}$$

Позначимо як

$$\begin{array}{l}\left\langle x^m\right\rangle=\sum\limits_{n=0}^{N}x_n^m;m=0\ldots2K\\ \left\langle x^my\right\rangle=\sum\limits_{n=0}^{N}x_n^my_n;m=0\ldots K\end{array}\tag{2.124}$$

середні по всіх вимірюваннях значення m -того степеня координати та добутку функції з m -тим степенем координати і задамо матрицю

$$A=\begin{pmatrix}\left\langle x^0\right\rangle & \left\langle x^1\right\rangle & \ldots & \left\langle x^K\right\rangle\\ \left\langle x^1\right\rangle & \left\langle x^2\right\rangle & \ldots & \left\langle x^{K+1}\right\rangle\\ \vdots & \ddots & \ddots & \vdots\\ \left\langle x^K\right\rangle & \left\langle x^{K+1}\right\rangle & \ldots & \left\langle x^{2K}\right\rangle\end{pmatrix}\tag{2.125}$$

та вектор – стовпчики

$$Y=\begin{pmatrix}\left\langle x^0y\right\rangle\\ \left\langle x^1y\right\rangle\\ \vdots\\ \left\langle x^Ky\right\rangle\end{pmatrix};a=\begin{pmatrix}a_0\\ a_1\\ \vdots\\ a_K\end{pmatrix}.\tag{2.126}$$

В цих позначеннях система рівнянь для коефіцієнтів апроксимуючого полінома має стандартний вигляд

$$Aa=Y.\tag{2.127}$$

Отже, задача поліноміальної апроксимації за методом найменших квадратів розв'язується, як задача обчислення коефіцієнтів матриці та знаходження розв'язку системи лінійних рівнянь.

Недоліком метода поліноміальної апроксимації є складність обчислення похибки кожного з коефіцієнтів апроксимуючого поліному. Матриця рівняння апроксимації є повною, тому кожен з коефіцієнтів залежить від кожної з пар значень „аргумент – функція". Більш того, розв'язки скороченої матриці з меншою кількістю параметрів не є розв'язками повнішої матриці. Тому в тих випадках, коли потрібно оцінити окремо точність апроксимації поліномами різних степенів, використовуються спеціалізовані методи.

Алгоритм поліноміальної апроксимації полягає в побудові матриці рівняння, що потребує обчислення $2K+2$ сум – коефіцієнтів матриці рівняння апроксимації. Розв'язок рівняння апроксимації має складність, що не залежить від кількості вузлів. За умови поступового накопичення набору послідовних степенів аргументу в кожному вузлі обчислення кожного елементу матриці потребує 4 дій на кожен вузол (добуток аргументу та його степеня, добуток степеня та функції, додавання отриманих добутків до суми), загалом $8KN$ дій.

2.4.3 Програмна реалізація

Залишаючи поза увагою інтерфейсну частину програми, вважатимемо, що два масиви - x[dim], y[dim] містять координати послідовних точок, тобто таблицю функції дискретної змінної, що для неї потрібно побудувати апроксимуючий поліном. Припустимо також, що в змінній K збережено значення степеню апроксимуючого полінома. Головною, масовою частиною задачі є обчислення всіх потрібних компонентів матриці A,Y рівняння для коефіцієнтів полінома. Елементами матриці є суми послідовних степенів x та добутків цих степенів на відповідні значення y. Відповідно крім масивів даних потрібно буде мати ще один, тимчасовий, масив степенів координати xp[dim]. На початок алгоритму цей масив можна заповнити просто одиничками і покласти $a\,[0]\,[0] \Leftarrow \mathrm{dim}$. Далі потрібно вираховувати, по-перше, суму добутків поточного степеню координати та функції $a\,[cur]\,[\mathrm{dim}] \Leftarrow a\,[cur]\,[\mathrm{dim}] + sp[k] \cdot y[k]$, а також масив наступного степеню координати $sp\,[k] \Leftarrow sp\,[k] \cdot x\,[k]$ та його суму $tmp \Leftarrow tmp + sp[k]$. Зауважимо, що в цьому обчисленні послідовність дій є принципово важливою, оскільки по обчисленні добутку чергова комірка $sp\,[k]$ вміщує вже значення наступного степеню координати. По закінченні циклу перебору всіх точок апроксимованої послідовності елемент матриці $a\,[cur]\,[\mathrm{dim}]$ має значенням черговий член правої частини матриці рівняння, елементи масиву $sp\,[k]$ наступний степінь координати, а допоміжна змінна tmp суму цих самих чергових степенів.

Версія C++

```
//#include <stdio.h>
#include <iostream.h> #define maxdim 1024*1024    //оголошення
розміру буфера для даних #define mdim 20           //оголошення
найбільшого степеня полінома
        double x[maxdim]; double y[maxdim]; //буфери даних
        double a[mdim][mdim+1];  //матриця рівняння для коефіцієнтів
        double res[mdim];        //коефіцієнти полінома
        int dim;                 //дійсний розмір масивів даних
```

```
        int km;                  //дійсний степінь полінома

void showa(){   //для відлагодження: друк матриці рівняння
      for(int k=0;k<=km;k++){
        cout<<k<<" ";      //номер рядка
        for(int kk=0;kk<=km;kk++){cout<<a[k][kk]<<"\t";};
        cout<<a[k][km+1]<<"\n";
      };
      cout<<"\n";
};

void reader(){ //читає дані та розраховує матрицю рівняння
    cin>>km;   //степінь апроксимації
    for(dim=0;(dim<maxdim)&&(cin>>x[dim]>>y[dim]);dim++);
              //масив даних:
              //dim містить кількість зачитаних рядків
    a[0][0]=dim;
    double xp[dim]; // степені аргументу
    double tmp;
    for(int k=0;k<dim;k++){xp[k]=1;a[0][km+1]+=y[k];};
            //сума одиниць та сума y - в першому рядку матриці
    for(int k=1;k<=km;k++){ //послідовний розрахунок степенів
      tmp=0;
      for (int kd=0;kd<dim;kd++){
         xp[kd]*=x[kd];          // наступний степінь аргументу
         a[k][km+1]+=xp[kd]*y[kd];//добутки степеня на значення функції
         tmp+=xp[kd];  //накопичення суми степенів
         };
         //розкладання суми степенів по зворотніх діагоналях матриці
      for(int kk=0;kk<=k;kk++){a[kk][k-kk]=tmp;}
      };
//  showa();
    for(int k=1;k<=km;k++){
      tmp=0;
      for (int kd=0;kd<dim;kd++){
         xp[kd]*=x[kd];
         tmp+=xp[kd];
         }; //спеціальний вигляд зворотньої діагоналі після головної
      for(int kk=km;kk>=k;kk--){a[kk][k+km-kk]=tmp;} ;
      };
//  showa();
};

void forw(int n){      //прямий хід метода Гауса
    double tmp=fabs(a[n][n]);int kt=n;
    for(int kr=n+1;kr<=km;kr++){ //пошук найбільшого провідного елемента
       if (fabs(a[kr][n])>tmp){kt=kr;tmp=fabs(a[kr][n]);};
       };
    if(kt>n){  //якщо є більший, рядки міняються місцями
        for(int kc=n;kc<=km+1;kc++)
```

```
        {tmp=a[kt][kc];a[kt][kc]=a[n][kc];a[n][kc]=tmp;};
        };
    tmp=a[n][n];    //нормалізація матриці a[n][n]<=1
    for(int kc=n;kc<=km+1;kc++){a[n][kc]/=tmp;};
    // тут тільки і починається зворотній хід
    for(int kr=n+1;kr<=km;kr++)
       {tmp=a[kr][n];
        for(int kc=n;kc<=km+1;kc++)
           {a[kr][kc]-=a[n][kc]*tmp;}
       }; //а тут вже й закінчується
// showa();
    if(n<km){forw(n+1);};
}; void ret(){        //обчислення результату
   for(int kr=km;kr>=0;kr--){
      res[kr]=a[kr][km+1];     //внесок від діагонального елемента
      for(int kc=kr+1;kc<=km;kc++){res[kr]+=-res[kc]*a[kr][kc];};
       //враховування внесків від усіх попередньо розрахованих
      };
};

void showr(){   //відтворення результату

for(int k=0;k<=km;k++){cout<<k<<" "<<res[k]<<"\n";};
};

int main(int argc, char *argv[]) {
   reader();   //зачитування даних
//   showa();
   forw(0);    //прямий
   ret();      //зворотній хід метода Гауса
   showr();     //відтворення  результату
  return 0;
}
```

Версія Pascal

```
program approx;
   const
        maxm        =1024;
        maxlen      =1024*1024;
        dataname    ='data.txt';
   Type
        data=array[0..maxlen] of double;
        matr=array[0..maxm,0..maxm+1] of double;
   var
        x,y       :data;
        a         :matr;
        res       :array[0..maxm] of double;
        dim,num   :Integer;
```

```
procedure showa;
  var kr,kc :Integer;
  begin
    for kr:=0 to dim
    do begin
       for kc:=0 to dim do Write(a[kr,kc]:8,' ');
       Writeln(a[kr,dim+1]:8);
       end;
  Writeln;
  end;
procedure showres;
  var k :Integer;
  begin
     for k:=0 to dim do Write(res[k]:10:5,' ');
     Writeln;
  end;
procedure Reader;
  var
     k,kc,kr   :Integer;
     infile    :Text;
     xpower    :data;
     tmp       :double;
  begin
     assign(infile,dataname);reset(infile);
     ReadLn(infile,dim);
     num:=-1;
     while not Eof(infile)
     do begin
        Inc(num);
        ReadLn(infile,x[num],y[num]);
        end;
     a[0,0]:=num+1;a[0,dim+1]:=0;
     for k:=0 to num
     do begin xpower[k]:=1;a[0,dim+1]:=a[0,dim+1]+y[k] end;
{showa;}
     for kr:=1 to dim
     do begin
        tmp:=0;a[k,dim+1]:=0;
        for k:=0 to num
        do begin
           xpower[k]:=xpower[k]*x[k];
           a[kr,dim+1]:=a[kr,dim+1]+xpower[k]*y[k];
           tmp:=tmp+xpower[k];
           end;
        for kc:=0 to kr do a[kr-kc,kc]:=tmp;
{showa;}
        end;
{showa;}
     for kr:=1 to dim
     do begin
```

```
        tmp:=0;
        for k:=1 to num
        do begin
           xpower[k]:=xpower[k]*x[k];
           tmp:=tmp+xpower[k];
           end;
        for kc:=dim downto kr do a[kc,kr+dim-kc]:=tmp;
        end;
showa;
  end; {Reader}
procedure forw(n:Integer);
  var kr,kc,kt :Integer; tmp:double;
  begin
     tmp:=abs(a[n,n]);  kt:=n;
     for kr:=n+1 to dim
     do if (abs(a[kr,n])>tmp)
        then begin tmp:=abs(a[kr,n]);kt:=kr;end;
     if kt>n
     then for kc:=n to dim+1
          do begin
             tmp:=a[n,kc];a[n,kc]:=a[kt,kc];a[kt,kc]:=tmp;
             end;
     tmp:=a[n,n];
     for kc:=n to dim+1 do a[n,kc]:=a[n,kc]/tmp;
     for kr:=n+1 to dim
     do begin
        tmp:=a[kr,n];
        for kc:=n to dim+1 do a[kr,kc]:=a[kr,kc]-tmp*a[n,kc];
        end;
showa;
     if n<dim then forw(n+1);
  end;

procedure ret;
  var kr,kc :Integer;
  begin
     for kr:=dim downto 0
     do begin
        res[kr]:=a[kr,dim+1];
        for kc:=kr+1 to dim do res[kr]:=res[kr]-res[kc]*a[kr,kc];
        end;
  end;
begin
  Reader;
  forw(0);
  ret;
  showres;
end.
```

```
> restart:with(linalg):Digits:=6:with(plots):
> path:="C:\\Documents and Settings\\const.US\\My
> Documents\\work\\tutor\\samples\\maple\\approx\\":
> dataname:="data.txt":
> source:=cat(path,dataname):target:=cat(path,resname):

Warning, the protected names norm and trace have been redefined
and unprotected

Warning, the name changecoords has been redefined
```

Версія Maple

підготовка даних про файл з параметрами та файл результатів

```
> dat:=readdata(source,2):nmax:=nops(dat)-1;
> dim:=round(dat[1][1]); x:=[seq(dat[k+1][1],k=1..nmax)];
> y:=[seq(dat[k+1][2],k=1..nmax)]; xp:=[seq(1,k=1..nmax)];
```

$$nmax := 5$$

$$dim := 2$$

$$x := [0., 1., 2., 4., 8.]$$

$$y := [0., .01, .03, .05, .06]$$

$$xp := [1, 1, 1, 1, 1]$$

Зачитування даних з файлу. Перший рядок містить степінь апроксимації, кожен наступний рядок містить дві координати чергового вузла.

```
> powx:=unapply(sum((x[k])^(ka+kb-2),k=1..nmax),kb,ka):
> powy:=unapply(sum(y[k]*x[k]^(ky-1),k=1..nmax),ky):
```

Функції розрахунку суми степенів аргументу та суми добутків степеня а ргумента на значення функції

побудова матриці рівняння

```
> a:=matrix(dim+1,dim+1,powx);
```

$$a := \begin{bmatrix} 5. & 15. & 85. \\ 15. & 85. & 585. \\ 85. & 585. & 4369. \end{bmatrix}$$

побудова правої частини рівняння

```
> b:=vector(dim+1,powy);
```

$$b := [.15, .75, 4.77]$$

знаходження розв'язку системи

```
> res:=linsolve(a,b);
```

$$res := [-.00230891, .0179415, -.00126563]$$

```
> Sum(X[k]^(kr+kc-2),k=1..len);
> Sum(Y[k]*X[k]^(ky-1),k=1..len);
```

$$\sum_{k=1}^{len} X_k{}^{(kr+kc-2)}$$

$$\sum_{k=1}^{len} Y_k \, {X_k}^{(ky-1)}$$

Побудова апроксимуючого полінома у формі функції

```
> fres:=unapply(sum(res[k+1]*z^k,k=0..dim),z);
```

$$fres := z \to -.00230891 + .0179415\, z - .00126563\, z^2$$

```
> appr:=plot(fres(z),z=x[1]..x[nmax]):
```

підготовка графічних зображень апроксимуючого полінома та набору даних для апроксимації

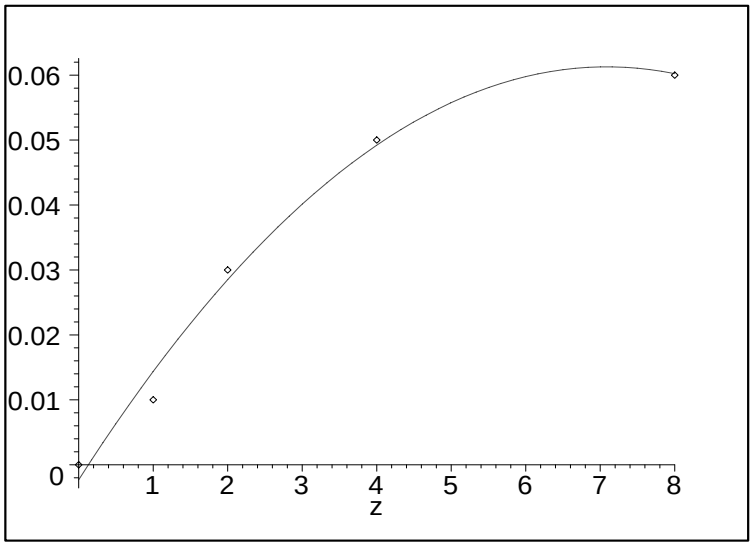
```
> pnt:=seq([x[k],y[k]],k=1..nmax);
```

$$pnt := [0., 0.], [1., .01], [2., .03], [4., .05], [8., .06]$$

```
> pn:=PLOT(POINTS(pnt,SYMBOL(DIAMOND))):
```

Результат в графічному зображенні

```
> display(appr,pn);
```



2.4.4 Метод ортогональних поліномів

Метою методу є таке переформулювання задачі поліноміальної апроксимації, за яким кожен з параметрів розраховувався би окремо від усіх інших.

З лінійної алгебри відомо, що квадратичну форму завжди можна звести до діагонального вигляду, тобто можна підібрати таке лінійне перетворення вектора аргументів квадратичної форми, що форма матиме вигляд суми квадратів.

Припустимо, що ми шукаємо розв'язок задачі апроксимації в класі поліномів степеню K, тоді параметрами апроксимації є коефіцієнти полінома $\{a_0, a_1, \ldots a_K\}$. Лінійне перетворення коефіцієнтів

$$\begin{cases} a_0 = M_{0,0}t_0 + M_{0,1}t_1 + \ldots + M_{0,K}t_K \\ a_1 = M_{1,0}t_0 + M_{1,1}t_1 + \ldots + M_{1,K}t_K \\ \\ a_K = M_{K,0}t_0 + M_{K,1}t_1 + \ldots + M_{K,K}t_K \end{cases} \tag{2.128}$$

відповідає тому, що замість апроксимуючого полінома $P_K(x) = a_0 + a_1 x + \ldots + a_K x^K$ розглядається вираз

$$\begin{array}{l} P_K(x) = (M_{0,0}t_0 + M_{0,1}t_1 + \ldots + M_{0,K}t_K) + \\ +(M_{1,0}t_0 + M_{1,1}t_1 + \ldots + M_{1,K}t_K)\,x + \ldots \\ +(M_{K,0}t_0 + M_{K,1}t_1 + \ldots + M_{K,K}t_K)\,x^K \end{array}, \tag{2.129}$$

що його можна перегрупувати, зібравши разом члени зі спільним новим параметром

$$\begin{array}{l} P_K(x) = t_0\left(M_{0,0} + M_{1,0}x + + \ldots + M_{K,0}x^K\right) + \\ +t_1\left(M_{0,1} + M_{1,1}x + \ldots + M_{K,1}x^K\right) + \ldots \\ +t_K\left(M_{0,K} + M_{1,K}x + \ldots + M_{K,K}x^K\right) \end{array}. \tag{2.130}$$

В цьому виразі кожен член в дужках є певним (відомим) поліномом, тож цей вираз означає апроксимаційний поліном, як суму деяких попередньо означених поліномів

$$P_K(x) = t_0 M_0(x) + t_1 M_1(x) + \ldots + t_K M_K(x) = \sum_{k=0}^{K} t_k M_k(x) \tag{2.131}$$

Підставимо цей вираз в функціонал похибок

$$\begin{array}{l} \Phi\left[f(x; a_0 \ldots a_K)\right] = \\ \sum\limits_{n=0}^{N} (t_0 M_0(x_n) + t_1 M_1(x_n) + \ldots + t_K M_K(x_n) - y_n) \\ (t_0 M_0(x_n) + t_1 M_1(x_n) + \ldots + t_K M_K(x_n) - y_n), \end{array} \tag{2.132}$$

або

$$\Phi\left[f(x; a_0 \ldots a_K)\right] = \sum_{n=0}^{N} \left(\sum_{k=0}^{K} t_k M_k(x_n) - y_n\right)\left(\sum_{p=0}^{K} t_p M_p(x_n) - y_n\right). \tag{2.133}$$

Переставимо тепер місцями порядок обчислення сум

$$\begin{array}{l} \Phi\left[f(x; a_0 \ldots a_K)\right] = \\ \sum\limits_{k,p=0}^{K} t_k t_p \sum\limits_{n=0}^{N} M_p(x_n) M_k(x_n) - 2\sum\limits_{k=0}^{K} t_k \sum\limits_{n=0}^{N} M_k(x_n)\, y_n + \sum\limits_{n=0}^{N} y_n^2 \end{array} \tag{2.134}$$

В цій потрійній сумі останній член взагалі не залежить від параметрів і ним можна нехтувати, якщо тільки не розраховується остаточна похибка апроксимації. Другий член є сумою незалежних внесків кожного з параметрів і тільки в першому внески параметрів з різними номерами переплутані.

Користуючись можливістю вільного чи майже вільного означення поліномів, спробуємо перетворити на нуль всі члени, в яких присутні параметри з різними номерами. Це буде мати місце, якщо кожна з сум

$$\sum_{n=0}^{N} M_p(x_n) M_k(x_n) = 0; \forall p \neq k \tag{2.135}$$

обертається на нуль, як тільки в ній присутні поліноми з різними номерами.

Використаємо позначення

$$\begin{aligned} \langle M_p M_k \rangle &= \sum_{n=0}^{N} M_p(x_n) M_k(x_n) \\ \left\langle M_p^2 \right\rangle &= \sum_{n=0}^{N} M_p^2(x_n) \\ \langle M_p y \rangle &= \sum_{n=0}^{N} M_p(x_n) y_n \end{aligned} \tag{2.136}$$

Якщо вважати значення полінома в кожному вузлі за компоненти вектора, то ці вирази є скалярними добутками відповідних векторів, а умова

$$\sum_{n=0}^{N} M_p(x_n) M_k(x_n) = 0; \forall p \neq k \tag{2.137}$$

є умовою ортогональності векторів. Як добре відомо з лінійної алгебри, взаємно ортогональних векторів завжди існує стільки, яка розмірність простору. Поліноми степеню не більше ніж K утворюють простір розмірності $K+1$ тому ми завжди можемо обрати систему $K+1$ ортогональних поліномів за базис і розкласти довільний поліном по цьому базису. Коефіцієнти такого розкладення і будуть параметрами апроксимації, а вираз для функціонала похибки

$$\Phi\left[f(x; a_0 \dots a_K)\right] = \sum_{k=0}^{K} \left(t_k^2 \left\langle M_k^2 \right\rangle - 2 t_k \langle M_k y \rangle\right) + \sum_{n=0}^{N} y_n^2 \tag{2.138}$$

буде сумою квадратних двочленів відносно кожного параметру апроксимації окремо.

Рівняння апроксимації замість переплутаної системи лінійних рівнянь розщеплюються в систему незалежних рівнянь

$$t_k \left\langle M_k^2 \right\rangle = \langle M_k y \rangle; k = 0 \dots K, \tag{2.139}$$

які розв'язуються в одну дію кожне.

Само собою зрозуміло, що початкова складність задачі нікуди не зникає, вона просто перекладається із задачі побудови матриці рівнянь та розв'язування цієї матриці на задачу побудови системи ортогональних поліномів.

Починаючи побудову системи ортогональних поліномів, зауважимо в першу чергу, що така система не є єдиною – довільне ортогональне перетворення базису знов призводить до ортогонального базису. Відповідну степінь вільності можна використати з метою спрощення побудови системи поліномів.

З практичної точки зору буде на користь, якщо система ортогональних поліномів впорядкована в такий спосіб, що кожен наступний поліном має на одиничку більший степінь, а починаються вони з полінома нульового степеня. Позначимо $T_0(x) = 1$ – найпростіший за побудовою поліном нульового степеня і побудуємо ортогональний до нього поліном першого степеня у вигляді $T_1(x) = x - a$. Цей поліном, дійсно є лінійним, а вільного параметра достатньо, аби задовольнити одну умову ортогональності до полінома нульового степеню. Дійсно, умова ортогональності

$$\langle T_1 T_0 \rangle = \sum_{n=0}^{N} (x_n - a) = 0 \tag{2.140}$$

буде задовільненою, якщо покласти

$$a = \frac{\sum\limits_{n=0}^{N} x_n}{N+1} = \bar{x} \tag{2.141}$$

Припустимо тепер, що ми побудували вже $m+1$ взаємно ортогональних поліномів послідовних степенів до m включно. Тоді довільний поліном m -того степеню є лінійною комбінацією таких поліномів. Більш за те, оскільки m - тий базисний поліном є ортогональним до всіх попередніх, він також є ортогональним до всіх поліномів степеню, меншого за m.

Центральним місцем подальшої побудови є припущення про те, що існують такі числа a_m, b_m, що вираз

$$T_{m+1}(x) = (x - a_m) T_m(x) - b_m T_{m-1}(x) \tag{2.142}$$

є ортогональним до всіх поліномів степеню, не вищого за m.

В першу чергу доведемо дійсність цього припущення для поліномів степеню, меншого за $m-1$.

Розглянемо скалярний добуток

$$\langle T_{m+1}(x), T_k(x) \rangle = \langle (x - a_m) T_m(x), T_k(x) \rangle - b_m \langle T_{m-1}(x), T_k(x) \rangle ; k < m-1 \tag{2.143}$$

другий член праворуч дорівнює нулю, оскільки $T_{m-1}(x)$ ортогональний до всіх поліномів меншого степеню. В першому члені частина з a_m обертається на нуль з тієї ж причини. Розглянемо тепер вираз

$$\begin{array}{l} \langle x T_m(x), T_k(x) \rangle = \\ \sum\limits_{n=0}^{N} x_n T_m(x_n) T_k(x_n) = \\ \sum\limits_{n=0}^{N} T_m(x_n) x_n T_k(x_n) = \\ \langle T_m(x), x T_k(x) \rangle ; \end{array} \quad k < m-1 \tag{2.144}$$

Переставивши множник x до поліному $T_k(x)$, легко зрозуміти, що добуток поліному степеню k та x утворює поліном, степінь якого $k+1$ завдяки умові $k < m-1$ залишається меншим за m, тому цей поліном ортогональний до поліному $T_m(x)$. Доведено.

Поліном

$$T_{m+1}(x) = (x - a_m) T_m(x) - b_m T_{m-1}(x) \tag{2.145}$$

буде ортогональним до всіх попередніх поліномів, якщо підібрати a_m, b_m в такий спосіб, аби справджувались умови ортогональності до поліномів $T_m(x), T_{m-1}(x)$ - маємо два рівняння для двох параметрів. Запишемо ці рівняння та їх розв'язки

$$\langle T_{m+1}, T_{m-1} \rangle = \langle x T_m, T_{m-1} \rangle - b_m \langle T_{m-1}^2 \rangle \Rightarrow b_m = \frac{\langle x T_m, T_{m-1} \rangle}{\langle T_{m-1}^2 \rangle} \tag{2.146}$$

$$\langle T_{m+1}, T_m \rangle = \left\langle xT_m^2 \right\rangle - a_m \left\langle T_m^2 \right\rangle \Rightarrow a_m = \frac{\left\langle xT_m^2 \right\rangle}{\langle T_m^2 \rangle} \tag{2.147}$$

Обидва вирази для розв'язків є змістовними, оскільки знаменники, як довжини векторів, ніколи не обертаються на нуль.

Таким чином, побудований алгоритм означення системи ортогональних поліномів, що дозволяє послідовно обчислювати поліноми до необхідного степеню включно.

Побудовані поліноми часто зустрічаються в обчислювальних задачах і тому отримали власну назву – їх називають поліномами Чебишева дискретної змінної, на відміну від поліномів Чебишева неперервної змінної. Для останніх скалярний добуток, замість суми по вузлах, означається інтегралом по заданому відрізку.

Обчислювальний алгоритм для задачі апроксимації методом ортогональних поліномів полягає в здійсненні наступних кроків

Початковий етап: апроксимація поліномами нульового та першого степенів. Квадрат довжини полінома нульового степеню дорівнює просто $N + 1$, тому параметр апроксимації нульового степеню є просто

$$t_0 = \frac{\sum\limits_{n=0}^{N} y_n}{N+1} = \bar{y} \tag{2.148}$$

середнім значенням функції. Необхідна кількість дій - N для обчислення середнього значення та ще декілька.

Поліном першого степеню має один параметр –

$$\bar{x} = \frac{\sum\limits_{n=0}^{N} x_n}{N+1}, \tag{2.149}$$

його обчислення потребує N дій. Обчислення параметру апроксимації за формулою

$$t_1 = \frac{\langle T_1 y \rangle}{\langle T_1^2 \rangle} \tag{2.150}$$

потребує ще $2N$ дій для обчислення чисельника та стільки ж дій для обчислення знаменника, загалом маємо $5N$

Для кожного наступного параметра апроксимації перед обчисленням власне параметра за формулою

$$t_k = \frac{\langle T_k y \rangle}{\langle T_k^2 \rangle}, \tag{2.151}$$

що потребує $2N$ дій, потрібно обчислити параметри наступного полінома за формулами

$$b_k = \frac{\langle xT_k, T_{k-1} \rangle}{\left\langle T_{k-1}^2 \right\rangle}; a_k = \frac{\left\langle xT_k^2 \right\rangle}{\langle T_k^2 \rangle}, \tag{2.152}$$

що потребує ще $2 * 3 * N$ дій (обчислення квадрата значення полінома, добутку на аргумент, та додавання до суми по вузлах – 3 дії на вузол). Отже, загалом маємо $8N$ на кожен наступний степінь апроксимації.

Алгоритм можна будувати ефективнішим або за швидкодією, або за пам'яттю. В першому разі доцільно для кожного наступного поліному зберігати: всі значення в вузлах, коефіцієнти розкладення полінома по степенях аргументу, квадрат довжини полінома. Оцінки обчислювальної складності стосуються саме такої реалізації. За умови жорстких обмежень на використання пам'яті можна кожного разу обчислювати значення поліному в усіх вузлах. Це потребує додаткової кількості дій $2k$ на кожен вузол.

Таким чином, метод ортогональних поліномів за обчислювальною складністю еквівалентний до методу прямої апроксимації. Перевагою цього методу є лише незалежність параметрів апроксимації, а як наслідок, можливість кращої оцінки точності визначення кожного з параметрів.

Програмна реалізація

Оскільки алгоритм апроксимації поліномами Чебишева досить складний, програмну реалізацію доцільно здійснювати тільки мовою C++.

Основою програми є дві спеціалізовані структури – динамічний список поліномів Чебишева cheby та динамічний список апроксимуючих поліномів poly. Структура поліномів Чебишева зберігає значення кожного з поліномів в усіх вузлах заданої мережі. Такий спосіб побудови алгоритму вимагає зайвої пам'яті, але суттєво прискорює роботу програми, оскільки не вимагає періодичного обчислення значень чергового полінома в кожному з вузлів. Для кожного з поліномів Чебишева зберігається також масив коефіцієнтів цього полінома.

$$T_m(x) = \sum_{k=0}^{m} c_{m,k} x^k \tag{2.153}$$

Це дозволяє під час побудови апроксимаційного полінома вираховувати не тільки коефіцієнти розкладення цього полінома в ряд по поліномах Чебишева, але й одночасно вираховувати коефіцієнти апроксимаційного полінома.

Результатом роботи програми є послідовність коефіцієнтів апроксимуючого поліному. Доповнення програми засобами обчислення загальної похибки та похибки обчислення окремих коефіцієнтів ряду по поліномах Чебишева залишається завданням для самостійної роботи.

Для розрахунку коефіцієнтів поліномів Чебишева використано вирази

$$\begin{aligned} &T_m(x) = (x-a)\sum_{k=0}^{m-1} c_{m-1,k} x^k - b\sum_{k=0}^{m-2} c_{m-2,k} x^k = \\ &c_{m-1,m-1} x^m + c_{m-1,m-2} x^{m-1} - a c_{m-1,m-1} x^{m-1} - a c_{m-1,0} - b c_{m-2,0} + \\ &\sum_{k=1}^{m-2} \left(c_{m-1,k-1} - a c_{m-1,k} - b c_{m-2,k}\right) x^k \end{aligned} \tag{2.154}$$

,

що витікають з рекурентного означення цих поліномів

```
#include <iostream.h>
#define maxsize 1024*1024

struct cheby{ /*поліном Чебишева*/
```

```
    double *cf;    /*коефіцієнти полінома*/
    double *seria; /*послідовність значень в вузлах*/
    double *args;  /*масив координат */
    double norm;   /*квадрат :) норми полінома*/
    int pow,len;   /*степінь полінома, довжина послідовності*/
    cheby* pred;   /*лінк до попереднього поліному */
cheby(){pred=0;cf=new double[1];cf[0]=1;pow=0;};
/* поліном нульового степеню*/
cheby(int length,double *arg){ /*поліном першого степеню*/
  len=length;
  pred=new cheby();pred->len=len;
  pred->args=new double[len];
   /*координати вузлів - в нульовому поліномі*/
  pred->seria=new double[len];
  pred->norm=len;
  for(int k=0;k<len;k++){
    pred->args[k]=arg[k];
    pred->seria[k]=1;
    /* поліном нульового степеню має значеннями одиниці*/
    };
  args=pred->args;
  /* посилається на той же масив, що є у нульового плінома*/
  pow=1;
  seria=new double[len];cf=new double[2];cf[1]=1;
   /* T1=(x-<x>): обчислення середньої координати */
  cf[0]=0;for(int k=0;k<len;k++){cf[0]-=args[k];};
  cf[0]/=len;
  norm=0;
  for(int k=0;k<len;k++){
    /* нормовочний множник дорівнює сумі квадратів значень */
     seria[k]=args[k]+cf[0];norm+=seria[k]*seria[k];
     };
};
cheby(cheby *old){
  /* наступний поліном посилається на попередній*/
  pred=old;
  pow=pred->pow+1; len=pred->len;
  /* степінь на 1 більше, ніж попереднього */
  cf=new double[pow+1];cf[pow]=1;
  args=pred->args;
  /* посилається на той же масив, що є у нульового плінома*/
  double a=0;
  for(int k=0;k<len;k++){
  /* a=sum(x*Tm*Tm)/norm(Tm) */
     a+=pred->seria[k]*pred->seria[k]*args[k];};
  a/=pred->norm;
  double b=0;
  for(int k=0;k<len;k++){
  /* b=sum(x*Tm*T(m-1))/norm(T(m-1)) */
     b+=pred->seria[k]*pred->pred->seria[k]*args[k];};
```

```
     b/=pred->pred->norm;
     /* коефіцієнти поліному */
     cf[pow-1]=pred->cf[pow-2]-a;
     cf[0]=-a*pred->cf[0]-b*pred->pred->cf[0];
     for(int p=1;p<pow-1;p++){
        cf[p]=pred->cf[p-1]-a*pred->cf[p]-b*pred->pred->cf[p];};
     norm=0;
      /* значення поліному в вузлах та норма */
    for(int k=0;k<len;k++){
        seria[k]=(args[k]-a)*pred->seria[k]-b*pred->pred->seria[k];
        norm+=seria[k]*seria[k];
        };
   };

}//cheby ;

struct poly{
       double t;      /* коефіцієнт при черговому поліномі */
       double *c;     /* коефіцієнти поліному (повного) */
       double *y;     /* масив значень функції */
       int pow;        /* степінь апроксимації (поточний) */
       poly *pred;     /* вказівник на попередній поліном */
       cheby *ch;      /* вказівник на відповідний поліном Чебишева*/
   double value(double x){
   /* значення розраховуються за схемою Горнера */
      double res=c[pow];
      for(int k=pow-1;k>=0;k--){
        res*=x;res+=c[k];};
      return res;
     };//value
   poly(double *val,int length){
     /* ініціалізація поліному нульового степеня */
      pred=0;   /* меншого степеня нема:)*/
      y=new double[length];t=0;pow=0;
      for(int k=0;k<length;k++){y[k]=val[k];t+=y[k];};
      /* коефіцієнт дорівнює середньому значенню функції */
      t/=length;
      c=new double[1];c[0]=t;
     };
   poly(double *arg, double *val,int length){
      /* ініціалізація поліному першого степеня,
         насправді - стартова точка:
         ініціалізує поліноми Чебишева
         а за компанію - поліном нульового степеня для себе
         */
      ch=new cheby(length,arg);
      pred=new poly(val,length);
      pow=1;y=pred->y;
      /* масив значень функції один, а посилання на нього є всюди*/
      t=0;
```

```
      for(int k=0;k<ch->len;k++){
         t+=y[k]*ch->seria[k];};
      t/=ch->norm;
      /* t=sum(y*T)/norm(T)*/
      c=new double[2];c[1]=t;
      /* повний поліном - сума добутку t*T та попереднього поліному */
      c[0]=t*ch->cf[0]+pred->c[0];
      };//init
   poly(poly* old){
      pred=old;
      pow=pred->pow+1;
      ch=new cheby(pred->ch);
      y=pred->y;
      t=0;
      for(int k=0;k<ch->len;k++){
         t+=y[k]*ch->seria[k];};
      t/=ch->norm;
       /* t=sum(y*T)/norm(T)*/
      c=new double[pow+1];
      c[pow]=t;
      for(int p=0;p<pow;p++){
         c[p]=t*ch->cf[p]+pred->c[p];};
      /* повний поліном - сума добутку t*T та попереднього поліному */
      };//prolongation
};//poly


int main(int argc, char *argv[]) {
        /* при відлагодженні вхідні дані задаються в тексті */
  double *x;//={0,1,2,3,5};//,0.3,0.4};
  double *y;//={0,0,2,6,20};//,0.09,0.16};
  x=new double[maxsize];y=new double[maxsize];
  int size;
  int power;
  cin>>power;
  for(size=0;(size<maxsize)&&(cin>>x[size]>>y[size]);size++);
  poly *pch=new poly(x,y,size);
  /*ініціалізація поліномів нульового та першого степенів*/
   for(int pw=1;pw<power;pw++){pch=new poly(pch);  };
  /*ініціалізація поліномів до потрібного степеню включно*/
  for(int k=0;k<=pch->pow;k++){
     /*відтворення коефіцієнтів апроксимуючого полінома*/
  cout<<k<<"\t"<<pch->c[k]<<"\n";
  };
  return 0;
}
```

Файл даних для апроксимації в першому рядку вміщує степінь апроксимую-

чого поліному, далі рядок за рядком таблицю значень, що їх потрібно апроксимувати.

Приклад тестового файлу. Дані є точками прямої $y = 1 - x$, тому апроксимуючий поліном повинен мати коефіцієнтами $a_0 = 1, a_1 = -1, a_2 = 0$.

```
2
0 1.0
0.1 0.9
0.2 0.8
0.5 0.5
0.9 0.1
1.0 0
```

Запуск програми здійснюється з переадресацією стандартного вхідного потоку на файл. Результатом є послідовність коефіцієнтів.

```
|15:38:14>cheb.exe < data.txt
    0   1
    1   -1
    2   1.02779e-15

|15:38:15>
```

Отримане значення $a_2 = 1.02779e - 15$ відрізняється від нуля внаслідок похибок округлення, що можна побачити хоча б з порядку величини значення.

2.5 Інтегрування та диференціювання

Задача інтегрування за своїми властивостями (починаючи вже від означення) є оберненою задачею, з усіма властивими оберненим задачам недоліками. Мається на увазі, що для задачі інтегрування сама постановка задачі полягає, якщо мова йде про визначення первісної, в знаходженні такої функції, що обчислення похідної від результату має призвести до заданої (підінтегральної) функції.

Відсутність прямої постановки задачі означає, що далеко насправді відсутнім є алгоритм обчислення інтегралу. Дійсно, курс математичного аналізу переповнений різноманітними штучними методами обчислення інтегралів для спеціальних класів підінтегральних функцій, починаючи від сталої та поліному. В загальному ж випадку означення інтегралу зводиться до границі скінченних сум, але граничний перехід не є алгоритмом, оскільки він не може бути здійснений за скінченну кількість кроків.

Як наслідок, явні вирази для первісних існують тільки для обмеженої множини явно інтегровних функцій. Практично всі ці функції вже давно проінтегровані, їх занесено до відповідних таблиць інтегралів, а останнім часом – і до бібліотек комп'ютерних систем аналітичних обчислень. Більше за те, певна кількість інтегралів, що для них не існує аналітичних виразів, але вони часто зустрічаються в практичних обчисленнях, просто оголошені спеціальними функціями математичної фізики і для них побудовано відповідні таблиці значень, вирази у вигляді нескінченного ряду і тому подібні засоби отримання потрібних значень. Тим не менше в практиці досить часто зустрічаються задачі, що потребують обчислення значень таких інтегралів, що для них відсутня можливість побудови більш-менш аналітичного виразу. Саме для таких інтегралів і застосовуються обчислювальні методи інтегрування. Доцільно зауважити, що ці методи є складовою частиною практично всіх систем наукового програмування, хоча в розрахункових задачах досить часто замість якої-небудь стандартної процедури інтегрування використовується саморобна процедура, що звичайно зроблена нашвидкоруч, рахує не надто швидко, може не враховувати особливостей поведінки конкретних функцій, що саме для них обчислюються інтеграли ... Коротше, в практичних обчисленнях досить часто єдиним критерієм вибору алгоритму обчислення інтегралу є критерій відомої оцінки для похибки обчислень, адже саме цей критерій дає можливість здійснити оцінку точності всього обчислення.

2.5.1 обчислення інтеграла

Обчислювальні методи в застосуванні до задачі інтегрування полягають в заміні обчислення інтеграла обчисленням скінченної суми, що її значення певною мірою наближається до значення інтеграла. Ця заміна здійснюється шляхом заміни підінтегральної функції - функції неперервної змінної, деякою функцією дискретної змінної, такою, що її значення в вузлах мережі співпадають зі значеннями підінтегральної функції.

$$f(x) : x = a \dots b \Rightarrow \{x_k, f_k = f(x_k); k = 0 \dots N, x_0 = a, x_N = b\} \qquad (2.155)$$

Має місце певна сваволя в побудові такої заміни. По-перше, можна в різні способи обирати мережу для функції дискретної змінної, як рівномірну, так і нерівномірну. По-друге, можна в різні способи обирати вагу того чи іншого вузла в сумі, що наближає значення інтеграла.

$$\int_a^b f(x)\,dx \Rightarrow \sum_{k=0}^{N} f_k \mu_k \tag{2.156}$$

Звичайно, якщо не пощастить і в обчисленні, припустимо, значення інтегралу від такої простої функції, як $\sin(x)+1$ всі вузли випадуть в точки $x_k = (2k - 1/2)\,\pi$, можна скільки завгодно змінювати вагу окремих вузлів – результат не змінюватиметься. Але найчастіше підінтегральні функції не мають такої впорядкованої поведінки і тому значення інтегралу доволі суттєво залежать від міри, що надається тим чи іншим вузлам мережі.

Найголовнішою властивістю скінченних сум, що обираються для обчислення інтегралу, є їх залежність від кількості вузлів.

Звичайно ж, в загальному випадку нічого конкретного про властивості границі суми

$$\lim_{N\to\infty} \sum_{k=0}^{N} f_k \mu_k \tag{2.157}$$

стверджувати не виходить, адже за невдалого вибору мережі можна отримати найрізноманітніші результати.

Разом з тим за досить нормальних умов існує багато різних можливостей так визначити вагу вузлів мережі, що послідовність скінченних сум зі зростанням кількості вузлів буде прямувати до значення інтегралу від підінтегральної функції. Залишається тільки питанням, скільки саме вузлів потрібно використати, аби значення скінченної суми знехтовно мало відрізнялось від значення інтегралу. Різні методи обчислення інтегралів дають різні оцінки для потрібної кількості вузлів за рахунок використання різних значень для ваги вузлів, але завжди слід мати на увазі, що всі ці оцінки є саме оцінками, тобто для деяких функцій вони майже співпадають із справжньою похибкою, а в окремих випадках можуть суттєво відрізнятись.

Найчастіше для оцінки точності методу використовується найбільший степінь полінома, що значення інтегралу від нього даний метод обчислює точно. Відповідно найточніші з цієї точки зору методи ніколи не дадуть високої точності при інтегруванні функції, що в деяких точках має стрибки.

Мистецтво обчислення інтегралів і полягає в правильному виборі методу, саме того, що є адекватним розв'язуваній задачі, має достатню точність і не потребує зайвих обчислень.

Оскільки всі методи відрізняються вибором ваги, що надається точкам мережі, критерії оцінки методів криються в тому, в який саме спосіб скінченна сума може бути витлумачена, як інтеграл.

Таке тлумачення ґрунтується на припущенні про те, що після дискретизації функції неперервної змінної здійснюється інтерполяція отриманої функції неперервної змінної, тобто відбувається подвійне перетворення

$$f(x) \Rightarrow \{x_k, f_k = f(x_k)\} \Rightarrow \tilde{f}(x) \tag{2.158}$$

і результатом другого етапу перетворення є така функція, що, з одного боку, її можна інтегрувати аналітично, з другого боку, значення інтегралу від неї знехтовно мало відрізняються від значення інтегралу початкової функції.

Інтерполяція в усіх методах інтегрування здійснюється за допомогою кусково-означених функцій, тобто є сплайн-інтерполяцією в деякому узагальненому тлумаченні. В найпростіших методах вона є лінійною, але можуть використовуватись і більш складні типи сплайнів.

Початкове ознайомлення з методами інтегрування доречно обмежити саме найпростішими методами. Перше і найпринциповіше обмеження – рівномірність гратки. Використання нерівномірної гратки є одним з найефективніших засобів прискорення обчислень, але вимагає досить глибокого дослідження функції, що інтегрується, тому може бути доцільним тільки в спеціальних задачах. Найчастіше в практиці обчислень гратка обирається рівномірною, для неї за характеристику детальності зображення інтегровної функції функцією дискретної змінної обирається крок – відстань між вузлами

$$h = \frac{b-a}{N} \Rightarrow x_k = a + kh; x_N = b \tag{2.159}$$

Вага кожного вузла повинна бути пропорційною величині кроку, найчастіше вона для пересічного вузла просто дорівнює величині кроку, змінюючись тільки для кінцевих вузлів.

Метод прямокутників полягає в виборі ваги для кожного вузла однаковою і такою, що дорівнює кроку.

За наївними міркуваннями слід було б чекати, що за методом прямокутників слід враховувати значення функції саме в усіх вузлах, використовуючи за наближене значення інтегралу суму

$$I_{bad} = \sum_{k=0}^{N} f\left(a + kh\right) h. \tag{2.160}$$

Виявляється, що цей метод має похибку навіть для сталої функції. Дійсно, припустимо, що функція, яку ми інтегруємо, має на всьому проміжку інтегрування стале значення $f\left(x\right) \equiv f_0; a \leqslant x \leqslant b$. Тоді інтеграл від неї дорівнює просто добутку цього сталого значення на довжину проміжку

$$I_{exact} = f_0 \left(b - a\right), \tag{2.161}$$

а з наївного виразу маємо

$$I_{bad} = \sum_{k=0}^{N} f_0 h = f_0 h \left(N + 1\right) = f_0 \left(Nh + h\right) = f_0 \left(b - a + h\right). \tag{2.162}$$

Певного покращення результату можна досягнути вилученням одного з вузлів, все одно якого. Геометричне тлумачення підказує, що це міг би бути один з кінцевих вузлів.

Метод трапецій

Цілком зрозуміло, що вилучення з переліку значень того чи іншого вузла не є найкращим методом підвищення точності. Єдиним шляхом покращення наївного виразу для інтегральної суми може бути перехід до сприйняття кінцевих вузлів за особливі, але однаково важливі для обчислення. Найпростіший спосіб такої корекції суми – надання кінцевим вузлам половинної ваги, тобто заміна наївного виразу наступним

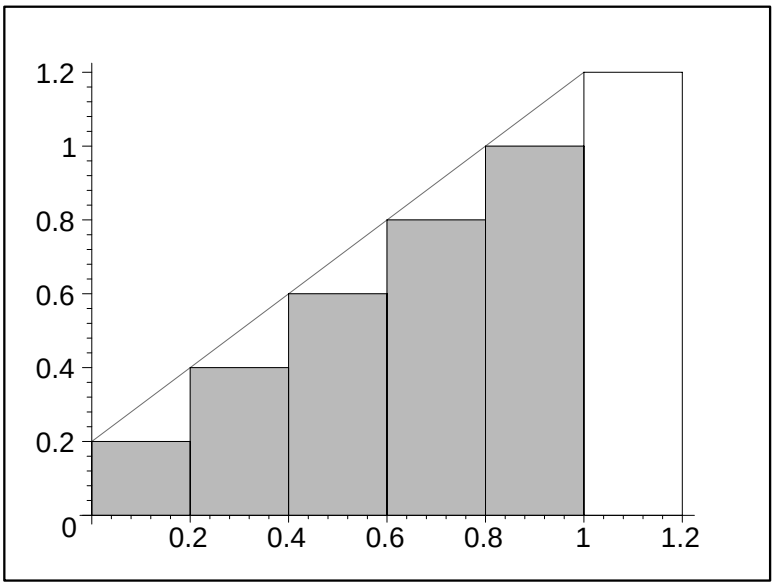


Рис. 2.4: Інтегральна сума дорівнює площі прямокутників. Значенню функції в останньому вузлі відповідає прямокутник, що виходить за межі області інтегрування.

$$I_{trap} = \sum_{k=1}^{N-1} f\left(a+kh\right)h + \frac{1}{2}\left(f\left(a\right)h + f\left(b\right)h\right). \tag{2.163}$$

Виявляється, що такому покращенню відповідає вже краща точність, ніж очікувалось. По-перше, звернімо увагу на ту обставину, що цю суму можна тепер подати у вигляді

$$I_{trap} = \sum_{k=0}^{N-1} \frac{1}{2}\left(f\left(a+kh\right)h + f\left(a+kh+h\right)h\right). \tag{2.164}$$

Пара доданків в кожному члені суми відповідають площі трапеції, що обмежена двома сусідніми вузлами, кожна з трапецій, зрозуміло ж, краще наближає площу під кривою. Власне, такий метод обчислення інтегралів і називають методом трапецій.

По-друге, з геометричним тлумаченням пов'язане і аналітичне. Методу трапецій відповідає використання інтерполяції лінійними сплайнами – відрізками прямих, що з'єднують сусідні точки.

Виявляється, що метод трапецій є точним не лише для сталої функції, а й для лінійної. Це легко зрозуміти з геометричного тлумачення, але доцільно розглянути й аналітичний вираз. Отже, припустимо, що підінтегральна функція є лінійною, тобто $f\left(x\right) = f_a + p\left(x-a\right); a \leqslant x \leqslant b$. Аналітичне обчислення дає таке значення інтегралу

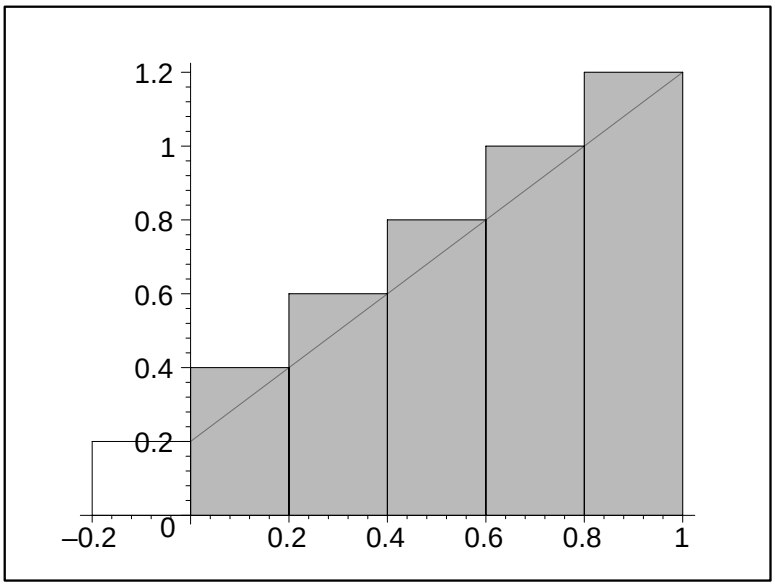


Рис. 2.5: Інтегральна сума дорівнює площі прямокутників. Значенню функції в першому вузлі відповідає прямокутник, що виходить за межі області інтегрування.

$$\int_a^b f(x)\,dx = \int_a^b (f_a + p(x-a))\,dx = f_a(b-a) + \frac{1}{2}p(b-a)^2 \tag{2.165}$$

Обчислимо відповідну суму методу трапецій. Для значень функції в вузлах гратки маємо

$$f(x_k) = f(a+kh) = f_a + pkh \tag{2.166}$$

тому

$$I_{trap} = \sum_{k=1}^{N-1} (f_a + pkh)\,h + \frac{1}{2}f_a h + \frac{1}{2}(f_a h + pNh) \tag{2.167}$$

або

$$I_{trap} = f_a h \sum_{k=1}^{N-1} 1 + ph^2 \sum_{k=1}^{N-1} k + f_a h + \frac{1}{2}pNh. \tag{2.168}$$

Обидві суми обчислюються аналітично. Перша є просто $I_1 = f_a h(N-1)$, а друга - $I_2 = \frac{1}{2}ph^2 N(N-1)$, тому значення інтегральної суми є

$$I_{trap} = f_a hN + \tfrac{1}{2}ph^2 N^2 \tag{2.169}$$

Неважко зрозуміти, що це значення повністю дорівнює аналітичному виразові.

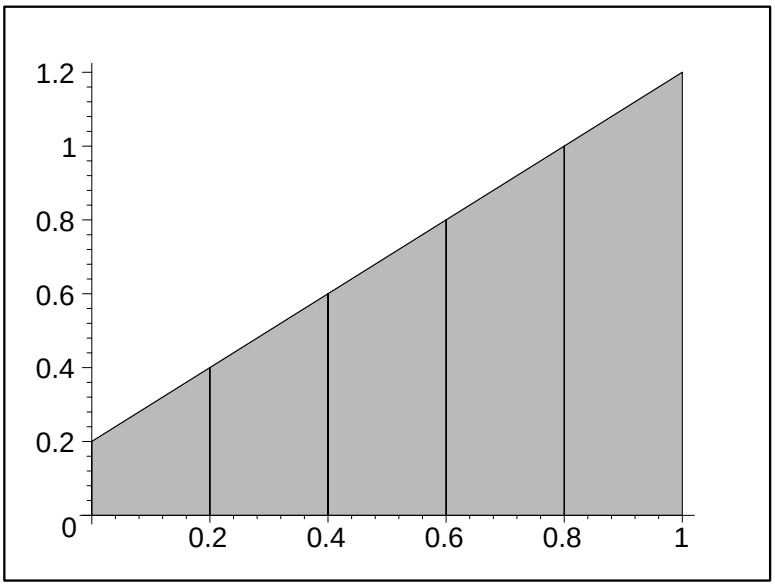


Рис. 2.6: Інтегральна сума дорівнює сумі площ трапецій.

2.5.2 Метод Симпсона

Метод Симпсона використовує квадратичну інтерполяцію. Замість властивого сплайн-інтерполяції узгодження похідних в кожному вузлі за методом Симпсона гратка поділяється на пари відрізків і на кожній парі використовується поліноміальна інтерполяція, не узгоджена з поліномами сусідніх пар відрізків. Вираз для інтегральної суми отримаємо, побудувавши інтерполяційний поліном для деякої пари відрізків, що мають середнім вузол з номером k.

Маємо вираз для інтерполяційного поліному

$$P_2(x) = f_k + p(x - kh) + q(x - kh)^2 \tag{2.170}$$

та рівняння інтерполяції

$$\begin{cases} f_k + p(kh - h - kh) + q(kh - h - kh)^2 = f_{k-1} \\ f_k + p(kh + h - kh) + q(kh + h - kh)^2 = f_{k+1} \end{cases}, \tag{2.171}$$

або

$$\begin{cases} f_k - ph + qh^2 = f_{k-1} \\ f_k + ph + qh^2 = f_{k+1} \end{cases}, \tag{2.172}$$

звідки

$$p = \frac{f_{k+1} - f_{k-1}}{2h}, q = \frac{f_{k+1} + f_{k-1} - 2f_k}{2h^2} \tag{2.173}$$

Інтеграл від кожного інтерполяційного поліному обчислюється на відповідній парі проміжків, тобто для розглядуваного поліному

$$I_{simp,k} = \int_{kh-h}^{kh+h} \left(f_k + p\,(x-kh) + q\,(x-kh)^2\right) dx = f_k 2h + q\frac{2}{3}h^3, \tag{2.174}$$

або

$$I_{simp,k} = f_k 2h + \frac{2}{3}h^3\left(\frac{f_{k+1}+f_{k-1}-2f_k}{2h^2}\right) = \frac{h}{3}\left(f_{k+1}+4f_k+f_{k-1}\right) \tag{2.175}$$

Побудуємо тепер вираз для всієї інтегральної суми. Це потребує обчислення суми подібних виразів по всіх парах відрізків (кількість відрізків обов'язково є парною). Центрами пар відрізків будуть вузли з непарними номерами, тож отримуємо

$$I_{simp} = \sum_{m=0}^{(N-1)/2} \frac{h}{3}\left(f_{2m}+4f_{2m+1}+f_{2m+2}\right) \tag{2.176}$$

Цей вираз доцільно розділити на три суми

$$I_{simp} = \frac{h}{3}\sum_{m=0}^{(N-1)/2} 4f_{2m+1} + \frac{h}{3}\sum_{m=0}^{(N-1)/2} f_{2m} + \frac{h}{3}\sum_{m=0}^{(N-1)/2} f_{2m+2} \tag{2.177}$$

В першій сумі присутні значення тільки в непарних вузлах, тоді як в другій та третій – в парних і тому їх можна об'єднати, вилучивши перший та останній члени. Отримуємо

$$\begin{array}{l} I_{simp} = \frac{h}{3}f_0 + \frac{h}{3}\sum\limits_{m=0}^{N/2-1} 4f_{2m+1} + \frac{h}{3}\sum\limits_{m=1}^{N/2-1} 2f_{2m} + \frac{h}{3}f_N \\ = \frac{h}{3}f_0 + \frac{4h}{3}f_1 + \frac{2h}{3}f_2 + \ldots + \frac{2h}{3}f_{N-2} + \frac{4h}{3}f_{N-1} + \frac{h}{3}f_N \end{array}. \tag{2.178}$$

Оцінку точності цієї схеми можна розпочати з того, що для поліномів другого степеню вона буде точною. Розглянемо поліном третього степеню в найпростішому вигляді $f(x) = x^3$. Для нього значення в вузлах є $f_k = h^3k^3$, тому для інтегральної суми маємо, позначивши $n = {}^N\!/_2$

$$\begin{array}{l} I_{simp} = \frac{4h^4}{3}\sum\limits_{m=0}^{n-1}(2m+1)^3 + \frac{16h^4}{3}\sum\limits_{m=1}^{n-1} m^3 + \frac{8h^4}{3}n^3 = \\ \frac{4h^4}{3}\left(2n^4-n^2\right) + \frac{16h^4}{3}\left(\frac{1}{4}n^4 - \frac{1}{2}n^3 + \frac{1}{4}n^2\right) + \frac{8h^4}{3}n^3 = 4h^4n^4 = \frac{1}{4}(hN)^4 \end{array} \tag{2.179}$$

Таким чином, метод Симпсона виявляється точним для поліномів до третього степеню включно.

Подібні ж обчислення для поліному четвертого степеню дають значення інтегральної суми

$$I_{simp}\left(x^4\right) = \frac{32}{5}h^5\left(n^5 - \frac{1}{3}n^3\right) = \frac{1}{5}b^5\left(1 - \frac{4h^2}{3b^2}\right), \tag{2.180}$$

що відрізняється від точного на величину $\Delta = -\frac{4}{15}b^3h^2$, тому похибку методу можна оцінити, як квадратичну.

Суттєвою особливістю похибки є зростання її абсолютної величини із зростанням довжини проміжку інтегрування. Це явище відоме, як накопичення похибки і властиве всім методам обчислювального інтегрування.

2.5.3 Обчислювальне диференціювання

На відміну від інтегрування, диференціювання є прямою операцією, тобто для кожної функції, що її можна записати явно, існує скінченна послідовність дій, результатом якої буде вираз для значень похідної. Тому задача обчислювального диференціювання має сенс тільки в тому разі, якщо функція задана не аналітично, а табличкою значень, тобто є функцією дискретної змінної.

Взагалі для функції дискретної змінної уявлення про похідну не означене, як таке, оскільки відсутня принципова можливість для здійснення граничного переходу, вкрай потрібного для обчислення похідної. Певні припущення про властивості похідної можна було б робити, якщо уявити, що значення в вузлах є відбиттям деякої функції неперервної змінної. Тоді для відношення різниць

$$\frac{\Delta f}{\Delta x} = \frac{f_{k+1} - f_k}{x_{k+1} - x_k} \tag{2.181}$$

можна було б шукати тлумачення, як значення похідної. Певні підстави для такого тлумачення дає стандартна теорема про приріст функції

$$\Delta f = f'\left(x + \vartheta \Delta x\right) \Delta x, \tag{2.182}$$

яка стверджує, що існує таке $0 < \vartheta < 1$, що визначає точку на проміжку, що значення похідної в цій точці задає приріст функції. На жаль, про величину ϑ більше нічого із загальних міркувань вигадати не виходить. Як наслідок, про відношення скінченних різниць можна тільки сказати, що воно існує, але не можна визначити, до якого саме кінця відрізку можна віднести це значення, аби вважати його за значення похідної в певному вузлі.

Більше за те, скінченні різниці та їх відношення взагалі не можуть тлумачитись а ні диференціалами, а ні похідними, оскільки не задовольняють стандартним властивостям цих операцій, таким, наприклад, як властивості диференціалу добутку функцій. Дійсно, для диференціалу добутку маємо стандартно, що

$$d\left(f\left(x\right) g\left(x\right)\right) = \left(df\left(x\right)\right) g\left(x\right) + f\left(x\right) \left(dg\left(x\right)\right). \tag{2.183}$$

Розглянемо відповідну скінченну різницю

$$\begin{array}{rcl} \Delta\left(fg\right) = & f_{k+1}g_{k+1} - f_k g_k & \\ = & \left(f_{k+1}g_{k+1} - f_k g_{k+1}\right) + \left(f_k g_{k+1} - f_k g_k\right) & \\ = & \left(f_{k+1} - f_k\right) g_{k+1} + f_k \left(g_{k+1} - g_k\right) & \end{array} \tag{2.184}$$

В першому члені значення функції g необхідно обчислювати в іншому вузлі, ніж в другому, а спроба звести їх до одного вузла призводить до іншого виразу

$$\begin{array}{rcl} \Delta\left(fg\right) = & \left(f_{k+1} - f_k\right)\left(g_{k+1} - g_k\right) + \left(f_{k+1} - f_k\right) g_k + f_k \left(g_{k+1} - g_k\right) & \\ = & \left(\Delta f\right) g + f\left(\Delta g\right) + \left(\Delta f\right)\left(\Delta g\right) & \end{array} \tag{2.185}$$

Для функцій неперервної змінної останній член є малою другого порядку малості і після граничного переходу їм можна нехтувати. Якщо ж розглядаються функції дискретної змінної, знехтувати останнім членом не виходить.

Як висновок, можна стверджувати, що означення похідної для функції, що її задано таблицею значень, можливе тільки після побудови відповідної функції неперервної змінної, а методи, що їх можна застосовувати до обчислення значень похідної для функції, заданої табличкою значень, суттєво залежать від походження функції.

Для функцій, що виникають в результаті теоретичних розрахунків і їх значення можна вважати точними, ефективною є інтерполяція функції з наступним обчисленням похідних. Зрозуміло, що результат інтерполяції суттєво залежить від класу функцій, в яком відшуковується розв'язок. Означення похідної можливе тільки в тому разі, коли клас функцій є диференційовним хоча б один раз. Саме ця вимога і заважає використанню відношення скінченних різниць в якості похідної. Взагалі кажучи, відношення

$$\frac{\Delta f}{\Delta x} = \frac{f_{k+1} - f_k}{x_{k+1} - x_k} \tag{2.186}$$

досить добре характеризує властивості інтерполюючої функції, але тільки у випадку інтерполяції лінійними сплайнами. Про ці сплайни добре відомо, що вони є неперервними всюди, але вони є диференційовними не всюди, а майже всюди, за виключенням саме тих точок, що в них необхідно визначати величину похідних – вузлів мережі.

З практичної точки зору обчислення похідної досить добре можна здійснювати за допомогою інтерполяції кубічними сплайнами, але слід мати на увазі, що результатом однократного диференціювання кубічного сплайну є вже квадратичний сплайн, а двократного - лінійний.

Якщо таблиця функції виникла внаслідок обробки результатів вимірювань, її значення завжди мають певну похибку і використання інтерполяційних методів є недоцільним. Але в таких задачах завжди присутня певна теоретична модель, що за нею відбувається апроксимація результатів вимірювань. Результатом апроксимації є вже добре означена функція неперервної змінної, що її можна диференціювати аналітично.

Застосування скінченних різниць до результатів вимірювань є серйозною помилкою, оскільки свідчить про відсутність уявлення про похибки вимірювань. Спробуємо оцінити вплив похибки на величину, що її можна було б обрати за оцінку похідної.

Припустимо, що в результаті вимірювань отримано по два досить близьких значення аргументу та функції $\{f_0 + \varepsilon_0, x_0 + \delta_0; f_1 + \varepsilon_1, x_1 + \delta_1\}$. Тут до кожної величини додано можливу похибку її вимірювання.

Тоді відношення різниць цих значень є

$$\frac{f_1 + \varepsilon_1 - f_0 + \varepsilon_0}{x_1 + \delta_1 - x_0 + \delta_0} = \frac{\Delta f + \varepsilon_1 + \varepsilon_0}{\Delta x + \delta_1 + \delta_0} \tag{2.187}$$

Тут враховано, що випадкові похибки, що виникають в процесі вимірювання, ніколи не скорочуються, а здатні тільки додаватись одна до одної, або просто як сума, або (і це більш вірогідно) за правилом додавання дисперсій незалежних випадкових величин, тобто $\varepsilon_{1+2} = \sqrt{\varepsilon_1^2 + \varepsilon_2^2}$, $\quad \delta_{1+2} = \sqrt{\delta_1^2 + \delta_2^2}$. Оскільки самі середні значення під час обчислення скінченних різниць майже скорочуються, цілком вірогідно отримати такі значення, що похибка і в чисельнику і в знаменнику буде домінувати і результатом обчислення буде випадкова величина.