

Міністерство освіти і науки України
Львівський національний університет
імені Івана Франка
Кафедра обчислювальної математики

Курсова робота
На тему
Ієрархічні матриці

Студентки 3-го курсу групи ПмП-31
спеціальності
6.040301 "Прикладна математика"
Солук О.О.

Керівник
Вавричук В.Г.

Національна шкала _____
Кількість балів _____ Оцінка ECTS _____

Члени комісії

_____	_____
_____	_____
_____	_____

Львів - 2017

Зміст

1	1. Вступ.	2
2	2. Ключова концепція.	3
2.1	Постановка проблеми.	3
2.2	Структура \mathcal{H} -матриці.	4
3	3. Дерево, Cluster Tree і Block Cluster Tree.	5
3.1	Означення дерева.	5
3.2	Означення Cluster Tree.	6
3.3	Означення Block Cluster Tree	7
3.4	Умова допустимості	8
3.4.1	Стандартна допустимість	9
3.4.2	Слабка допустимість	10
3.5	Означення \mathcal{H} -матриці	10
3.6	Приклад побудови block cluster tree	11
4	4. Метод граничних елементів (ВЕМ)	13
4.1	Модельна задача	13
4.2	Розклад ядра в ряд Тейлора	14
4.3	Наближення низького рангу блоків матриці	15
5	5. Збірка, пам'ять і множення матриці на вектор	18
5.1	Побудова матриці	18
5.1.1	Недопустимі листки	18
5.1.2	Допустимі листки	18
5.1.3	Репрезентація ієрархічної матриці	19
5.2	Метод спряжених градієнтів	20
5.3	Програмна реалізація	21
6	Висновок	22
7	Додатки	23

1. Вступ.

Ієрархічна матриця (\mathcal{H} -матриця) використовується для апроксимації розрідженими даними щільних матриць. Ці матриці застосовують коли намагаються розв'язати систему лінійних рівнянь

$$Ax = b \quad A \in \mathbb{R}^{n \times n}, \quad x \in \mathbb{R}^n$$

з майже лінійною складністю $O(n \log(n))$.

Вперше концепцію ієрахічних матриць запропонував Вольфганг Хакбуш в 1998 році. Він розширив ідею panel clustering methods, зробивши її застосовною до загальних алгебраїчних операцій над матрицями, оберненими матрицями тощо.

В цій роботі розглянуто базові означення та процес побудови \mathcal{H} -матриць, їх застосування на прикладі одновимірної модельної задачі, для розв'язування якої використовують метод граничних елементів (BEM - boundary element method).

Текст цієї роботи написано на основі матеріалів [1],[2]. Також, деякі графічні ілюстрації взяті з роботи [1].

Автору даної курсової роботи належить власна програмна реалізація на мові C# таких описаних у роботі алгоритмів:

- побудова cluster tree для $n = 2^p$.
- побудова block cluster tree.
- множення \mathcal{H} -матриці на вектор.
- реалізація методу спряжених градієнтів.

2. Ключова концепція.

2.1 Постановка проблеми.

В загальному випадку, вартість обчислень при роботі зі щільною матрицею A - $O(n^2)$. При великому n з цією складністю не можуть впоратися навіть сучасні комп'ютери. Враховуючи наявну пам'ять комп'ютера і швидкість обчислень, намагаються уникати операцій з великими, повністю заповненими матрицями. Натомість намагаються звести всі алгоритми до роботи з розрідженими матрицями. Таким чином постає питання чи можливо замінити матрицю A на матрицю \tilde{A} , таку що вартість обчислень стане майже лінійною ($n \log(n)$). Для цього використовують \mathcal{H} -матриці, операції з якими дозволяють наближення. Тим не менше, похибки наближення є допустимими так, як великі матриці отримують шляхом дискретизації, завдяки чому буде присутня похибка дискретизації.

Операції з \mathcal{H} -матрицями:

1. Додавання матриць
2. Множення матриць
3. Множення \mathcal{H} -матриці на вектор
4. Знаходження оберненої матриці
5. LU-розклад
6. Розклад Холецкого

Наведені твердження правдиві не для всіх матриць, але вони виконуються для важливого класу матриць, які походять від стандартної дискретизації еліптичних диференціальних рівнянь і відповідних інтегральних рівнянь.

Багато проблем в природі можуть бути представлені за допомогою диференціальних або інтегральних рівнянь. Методи розв'язання цих рівнянь базуються на лінеаризації або дискретизації ядра. Після їх застосування отримуємо систему лінійних рівнянь вигляду

$$Ax = b \quad A \in \mathbb{R}^{n \times n}, \quad x \in \mathbb{R}^n$$

Основні методи:

1. FEM (finite element method) - метод скінченних елементів. A - розріджена матриця, A^{-1} - щільна.

2. ВЕМ (boundary element method) - метод граничних елементів. A - щільна матриця.

2.2 Структура \mathcal{H} -матриці.

Техніка побудови ієрархічних матриць (\mathcal{H} -матриць) використовує деревоподібну структуру з розрідженими даними для того, щоб зберігати заповнену матрицю таким чином, що листки дерева є заповненими або матрицями низького рангу.

Формат \mathcal{H} -матриці базується на ієрархічній деревоподібній структурі, яка називається block cluster tree. Цю структуру можна отримати ієрархічним поділом множини індексів на підблоки, які називають cluster tree.

Під час побудови cluster tree (\mathcal{H} -дерева) на кожному рівні поділу, ці підблоки повинні задовільняти умову допустимості (admissibility condition), яка вирішує чи вони є листком, чи підлягають подальшому поділу.

Означення \mathcal{H} -матриці в загальному базується на означенні дерева.

3. Дерево, Cluster Tree і Block Cluster Tree.

3.1 Означення дерева.

Перед тим, як дати означення дерева введемо такі означення:

Означення 3.1 Бінарним відношенням на множині A називається довільно підмножина декартового добутку $A \times A$.

Означення 3.2 Графом $G = (V, E)$ називається впорядкована пара множин V, E , де V - скінченна множина вершин, а E - мультимножина, яка містить ребра, що описуються парою вихідна і кінцева вершина.

Означення 3.3 Шляхом в неорієнтованому графі $\langle u, v \rangle$ довжиною r називають послідовність ребер

$$\langle u, v \rangle = (\{x_0, x_1\}, \{x_1, x_2\}, \dots, \{x_{r-1}, x_r\})$$

$$x_0 = u, x_r = v$$

Шлях, який не містить повторюваних ребер називається простим.

Неорієнтований граф називається зв'язним, якщо між довільною парою його вершин існує шлях.

Означення 3.4 (Дерево) Нехай V - непорожня множина і $E \subseteq V \times V$ є бінарним відношенням над V . Пара $\mathbb{T} = (V, E)$, де множина $V = V(\mathbb{T})$ є множиною вершин \mathbb{T} , а множина $E = E(\mathbb{T})$ - множина ребер \mathbb{T} , називається деревом, якщо виконуються такі умови:

- Унікальна вершина $v \in V$ називається коренем дерева і позначається $root(\mathbb{T})$
 $\Leftrightarrow \forall w \in V : w \neq v$ виконується $(w, v) \notin E$.
- Для будь-якої вершини $v \in V \setminus root(\mathbb{T})$ існує єдиний простий шлях з $root(\mathbb{T})$ до v .

Іншими словами, дерево - це неорієнтований зв'язний граф без простих циклів.

Введемо такі позначення:

- Для вершини $v \in V$ множина її синів визначається як

$$S(v) = \{w \in V | (v, w) \in E\}$$

- Множину всіх листків дерева \mathbb{T} визначають як $\mathcal{L}(\mathbb{T}) = \{v \in V | S(v) = \emptyset\}$
- Рівень дерева \mathbb{T} визначається рекурсивно як

$$\mathbb{T}^{(0)} = \text{root}(\mathbb{T})$$

$$\mathbb{T}^{(l)} = \{v \in V | \exists w \in \mathbb{T}^{(l-1)} : (w, v) \in E\}$$

- Висотою дерева $d(\mathbb{T})$ називається найдовший простий шлях від кореня до листка.

Означення 3.5 Повним бінарним деревом називається кореневе дерево, в якого кожна внутрішня вершина має точно двоє дітей.

Означення 3.6 Кореневе бінарне дерево називається збалансованим, якщо його листки знаходяться на рівні h або $h-1$.

3.2 Означення Cluster Tree.

Як $I = 0, 1 \dots n - 1$ позначимо скінченну множину індексів з потужністю $|I| = n$. В майбутньому, в ролі I використовуватимемо індекси базових функцій, отриманих для дискретизації з ВЕМ.

Концепція cluster tree базується на концепції дерева.

Означення 3.7 (Cluster Tree) Дерево \mathbb{T}_I називається cluster tree над множиною індексів I з $\text{root}(\mathbb{T}_I) = I$, якщо наступні умови виконуються:

- $I \in V$ є коренем \mathbb{T}_I і $\forall v \in V, v \neq \emptyset \Rightarrow v \subseteq I$.
- Якщо $v \in V$ не є листком ($S(v) \neq \emptyset$), то він рівний об'єднанню своїх синів, тобто $v = \bigcup_{w \in S(v)} w$.

$v \in V$ називають кластером.

В одновимірному випадку cluster tree - збалансоване бінарне дерево.

Приклад. Одновимірний випадок.

Як корінь дерева \mathbb{T}_I беремо множину індексів $I_0^{(0)} = \{0, \dots, n - 1\}$. Для легкості презентації припустимо, що кількість базисних функцій n є степенем 2:

$$n = 2^p$$

Розглядаємо випадок, коли $p = 3$. Починаючи з кореня, де тільки один кластер, це дерево конструюється шляхом поділу кожної множини індексів $I_i^{(j)}$ на два нащадки $I_{2i}^{(j+1)}$ і $I_{2i+1}^{(j+1)}$ при $0 \leq i, j \leq p$. Нарешті на рівні p всі кластери (вузли) є листками, наприклад $\mathcal{L}(\mathbb{T}) = \{I_i^{(3)}\}_{i=0}^7$. Кожен вузол (окрім листків) отриманого дерева матиме рівно два нащадки:

Два нащадки $I_0^{(0)}$: $I_0^{(1)} = \{0, \dots, \frac{n}{2} - 1\}$ і $I_1^{(1)} = \{\frac{n}{2}, \dots, n - 1\}$.

Два нащадки $I_0^{(1)}$: $I_0^{(2)} = \{0, \dots, \frac{n}{4} - 1\}$ і $I_1^{(2)} = \{\frac{n}{4}, \dots, \frac{n}{2} - 1\}$.

Два нащадки $I_1^{(1)}$: $I_2^{(2)} = \{\frac{n}{2}, \dots, \frac{3n}{4} - 1\}$ і $I_3^{(2)} = \{\frac{3n}{4}, \dots, n - 1\}$.

Зауваження. В попередньому прикладі, подальший поділ множини індексів зупиняється точно тоді, коли залишається тільки один елемент. Проте зазвичай на практиці фіксують найменший розмір кластера n_{min} . Завдяки цьому поділ зупиняється, тоді коли потужність вузла менша чи рівна за n_{min} . В обчислювальних експериментах, є ефективним брати n_{min} між 32 і 64.

З практичної точки зору, \mathbb{T}_I зазвичай є бінарним деревом. Висота бінарного дерева $\approx \log(n)$, а отже складність побудови cluster tree - $O(n \log(n))$.

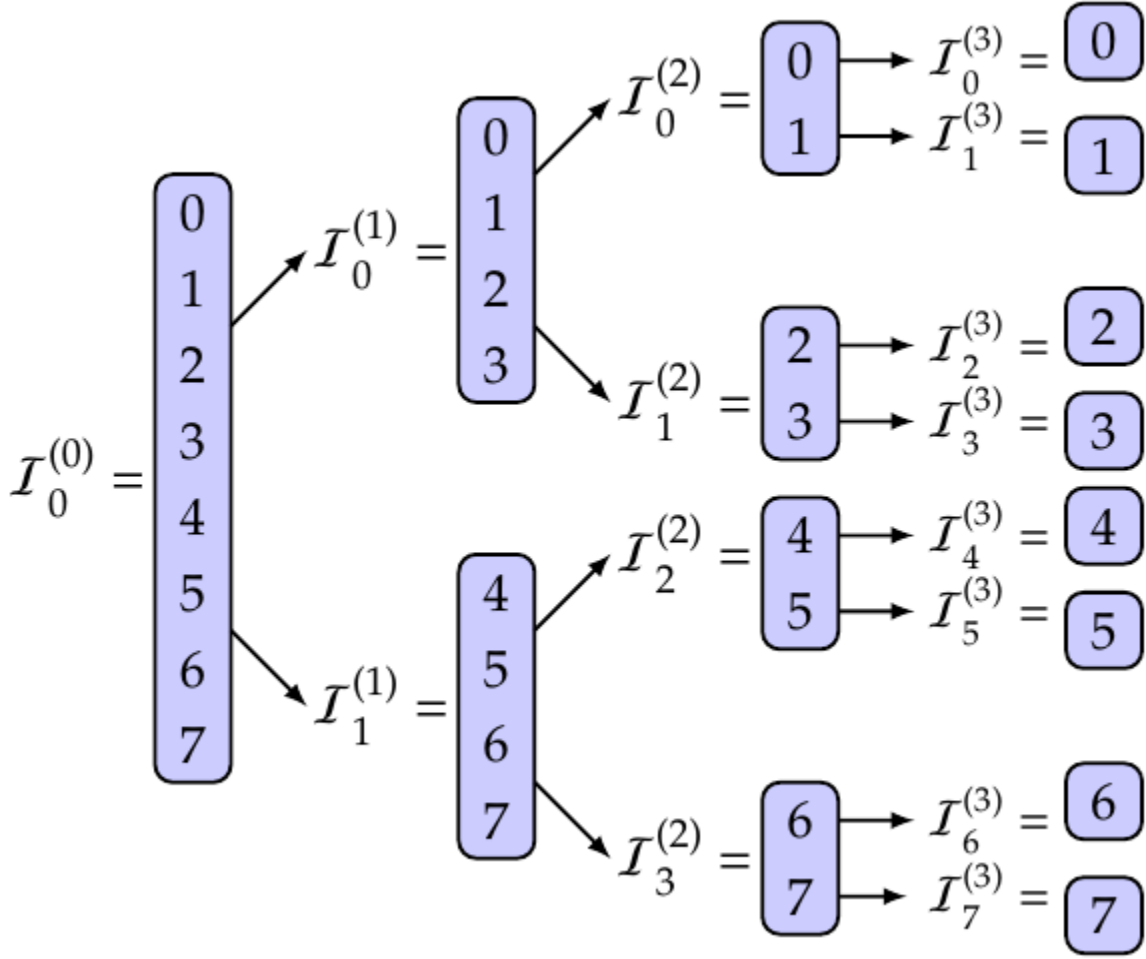


Рис. 3.1: Cluster Tree при $p=3$.

3.3 Означення Block Cluster Tree

Block cluster tree є cluster tree над множиною індексів $I \times I$ замість I . В загальному, для неквадратних матриць, що належать до $\mathbb{R}^{I \times J}$, потрібно два різні cluster trees \mathbb{T}_I та \mathbb{T}_J , тому ми розглядаємо інше cluster tree \mathbb{T}_J , яке базується на множині індексів J потужності $|J| = m$.

Означення 3.8 Нехай \mathbb{T}_I і \mathbb{T}_J - cluster trees над множинами індексів I та J відповідно. Cluster tree $\mathbb{T}_{I \times J} = \mathbb{T}_{\mathbb{T}_I \times \mathbb{T}_J} = (V, E)$ називається block cluster tree над добутком множини індексів $I \times J$, якщо $\forall v \in V$ виконуються наступні умови:

- $\mathbb{T}_{I \times J}^{(0)} = I \times J$
- Якщо $v \in \mathbb{T}_{I \times J}^{(l)}$, то існують $\tau \in \mathbb{T}_I^{(l)}$ і $\sigma \in \mathbb{T}_J^{(l)}$ такі, що $v = \tau \times \sigma$.
- Для синів $v = \tau \times \sigma$, де $\tau \in \mathbb{T}_I$ і $\sigma \in \mathbb{T}_J$ виконується
$$S(v) = \begin{cases} \emptyset, \text{якщо } S(\tau) = \emptyset \text{ або } S(\sigma) = \emptyset \\ \{\tau' \times \sigma' : \tau' \in S(\tau), \sigma' \in S(\sigma)\}, \text{інакше} \end{cases}$$

Властивості block cluster tree $\mathbb{T}_{I \times J}$:

- Якщо обое cluster trees \mathbb{T}_I і \mathbb{T}_J є бінарними деревами, то отримане block cluster tree є quad-деревом, тобто кожний внутрішній вузол має точно чотири нащадки.
- $|\mathcal{L}(\mathbb{T}_{I \times J})| \leq |\mathcal{L}(\mathbb{T}_I)| \cdot |\mathcal{L}(\mathbb{T}_J)|$
- $|\mathbb{T}_{I \times J}^{(l)}| \leq |\mathbb{T}_I^{(l)}| \cdot |\mathbb{T}_J^{(l)}|$
- $d(\mathbb{T}_{I \times J}) = \min\{d(\mathbb{T}_I), d(\mathbb{T}_J)\}$

Кількість можливих блоків $t \times s$ з вузлами t, s , що належать дереву \mathbb{T}_I становить $(\#\mathbb{T}_I)^2 = (2n-1)^2 = \mathcal{O}(n^2)$. Оскільки ми не можемо тестувати всі можливі комбінації, нашою метою є зменшення квадратичної збіжності для збірки матриці. Можливим варіантом є тестування блоків рівень за рівнем починаючи від кореня I дерева \mathbb{T}_I і в подальшому заглиблюючись в дерево. Блоки, що тестуються, зберігають в block cluster tree $\mathbb{T}_{I \times I}$, листки якого утворюють поділ множини індексів $I \times I$.

Алгоритм побудови block cluster tree(викликати з параметрами $\text{BuildBlockClusterTree}(I, I)$):

Algorithm 1 Побудова block cluster tree $\mathbb{T}_{I \times I}$

```

procedure BuildBlockClusterTree(cluster t,s)
if (t, s) is admissible then
     $S(t \times s) := \emptyset$ ;
else
     $S(t \times s) := \{t' \times s' | t' \in S(t) \text{ and } s' \in S(s)\}$ ;
    for  $t' \in S(t)$  and  $s' \in S(s)$  do
        BuildBlockClusterTree( $t', s'$ );
    end for
end if

```

3.4 Умова допустимості

Під час побудови block cluster tree $\mathbb{T}_{I \times J}$ потрібна допоміжна умова, яка перевіряє чи блок $b = \tau \times \sigma \in \mathbb{T}_{I \times J}$ є відповідного розміру і в особливості чи він може бути наближений розрідженою матрицею. Це робиться за допомогою умови допустимості,

яка в певному сенсі залежить від геометрії основної проблеми. Умова допустимості певним чином збалансовує точність апроксимації та вимоги до пам'яті \mathcal{H} -матриць.

Означення 3.9 Умова допустимості є булівською функцією

$$Adm : \mathbb{T}_{I \times J} \rightarrow \{true, false\}$$

для якої виконується умова

$$Adm(b) \Rightarrow Adm(b'), \quad \text{для всіх синів } b' \subseteq b \in \mathbb{T}_{I \times J}$$

і властивість

$$Adm(b) = true, \quad \text{для всіх листків } b \in \mathbb{T}_{I \times J}$$

Означення 3.10 Поділ \mathcal{P} називається допустимим (\mathcal{P}_{Adm}), якщо всі $b = (\tau \times \sigma) \in \mathcal{P}$ є допустимими або малими в порівнянні з n_{min} .

В літературі виділяють два типи умов допустимості:

1. Стандартна допустимість.
2. Слабка допустимість.

3.4.1 Стандартна допустимість

Стандартна умова допустимості була вперше описана в класичній побудові \mathcal{H} -матриць, де вона застосовується для розподілу, здебільшого для проблем, що вирішуються за допомогою ВЕМ.

Означення 3.11 Нехай $\eta > 0$ - фіксований параметр. Кажуть, що блок $b = \tau \times \sigma$ задовольняє стандартну умову допустимості, якщо

$$Adm_\eta(b) = true \Leftrightarrow \min(diam(\Omega_\tau), diam(\Omega_\sigma)) \leq \eta \cdot dist(\Omega_\tau, \Omega_\sigma)$$

де Ω_τ і Ω_σ визначаються як

$$\Omega_\tau := \bigcup_{i \in \tau} supp(\varphi_i)$$

$$\Omega_\sigma := \bigcup_{i \in \sigma} supp(\varphi_i)$$

В попередніх означеннях "diam" і "dist" позначають Евклідовий діаметр і відстань між Ω_τ та Ω_σ . Вони визначаються наступним чином:

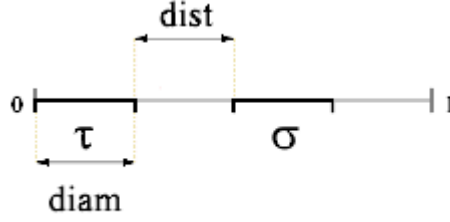
$$diam(\Omega_\tau) := \max_{x_i, x_j \in \Omega_\tau} \|x_i - x_j\|$$

$$dist(\Omega_\tau, \Omega_\sigma) := \min_{x_i \in \Omega_\tau, x_j \in \Omega_\sigma} \|x_i - x_j\|$$

Якщо в означенні 4.3 замінити "min" на "max" то отримуємо сильну умову допустимості:

$$Adm_\eta(b) = true \Leftrightarrow \max(diam(\Omega_\tau), diam(\Omega_\sigma)) \leq \eta \cdot dist(\Omega_\tau, \Omega_\sigma)$$

В подальшому для одновимірної проблеми ми будемо використовувати стандартну умову допустимості в такому вигляді



$$\text{diam}(\tau) \leq \text{dist}(\tau, \sigma)$$

Означення 3.12 Блок індексів $t \times s \subset I \times I$ називається допустимим, якщо відповідна область $\tau \times \sigma$ з $\tau := \bigcup_{i \in t} \text{supp} \varphi_i$, $\sigma := \bigcup_{i \in s} \text{supp} \varphi_i$ задовільняє умову

$$\text{diam}(\tau) \leq \text{dist}(\tau, \sigma)$$

3.4.2 Слабка допустимість

Інший критерій допустимості дозволяє всім блокам $b = \tau \times \sigma$ з різними τ і σ бути допустимими. Ця слабка умова допустимості застосовується для одновимірних проблем, бо для багатовимірних вона в загальному не є дійсною.

Означення 3.13 Кажуть, що блок $b = \tau \times \sigma$ задовольняє слабку умову допустимості, якщо

$$\text{Adm}_W(b) = \text{true} \Leftrightarrow b \text{ є листком або } \tau \neq \sigma$$

Слід зазначити, що якщо для блоку b виконується сильна умова допустимості, то слабка умова виконується автоматично, тобто

$$\text{Adm}_\eta = \text{true} \Rightarrow \text{Adm}_W = \text{true}$$

3.5 Означення \mathcal{H} -матриці

На допустимих блоках апроксимуємо за допомогою структури `gkmatrix`. На недопустимих листках використовуємо структуру `fullmatrix`. Детальніше про ці структури в 6 розділі.

n_{\min} - розмірність найменшого листка.

Означення 3.14 Нехай $M \in \mathbb{R}^{I \times J}$ - матриця над множиною індексів $I \times J$. Підматриця $(M_{i,j})_{(i,j) \in I' \times J'}$ для підмножини $I' \times J'$ множини $I \times J$ позначається як $M|_{I' \times J'}$.

Означення 3.15 (\mathcal{H} -матриця) Нехай $k, n_{\min} \in \mathbb{N}_0$. Множина \mathcal{H} -матриць, що базується на допустимому поділі \mathcal{P}_{Adm} над block cluster tree $\mathbb{T} := \mathbb{T}_{I \times J}$ визначається як

$$\mathcal{H}(\mathbb{T}, k) := \{M \in \mathbb{R}^{I \times J} \mid \forall \tau \times \sigma \in \mathcal{P}_{\text{Adm}} : \text{rank}(M|_{\tau \times \sigma}) \leq k \text{ або } \min(|\tau|, |\sigma|) \leq n_{\min}\}$$

Спрощене означення \mathcal{H} -матриці виглядає наступним чином

Означення 3.16 Нехай $\mathbb{T}_{I \times I}$ - block cluster tree над множиною індексів I . Означаємо множину \mathcal{H} -матриць як

$$\mathcal{H}(\mathbb{T}_{I \times I}, k) := \{M \in \mathbb{R}^{I \times I} | \text{rank}(M|_{t \times s}) \leq k \text{ для всіх допустимих листків } t \times s \text{ дерева } \mathbb{T}_{I \times I}\}$$

Графічна репрезентація \mathcal{H} -матриць для стандартної та слабкої умови допустимості

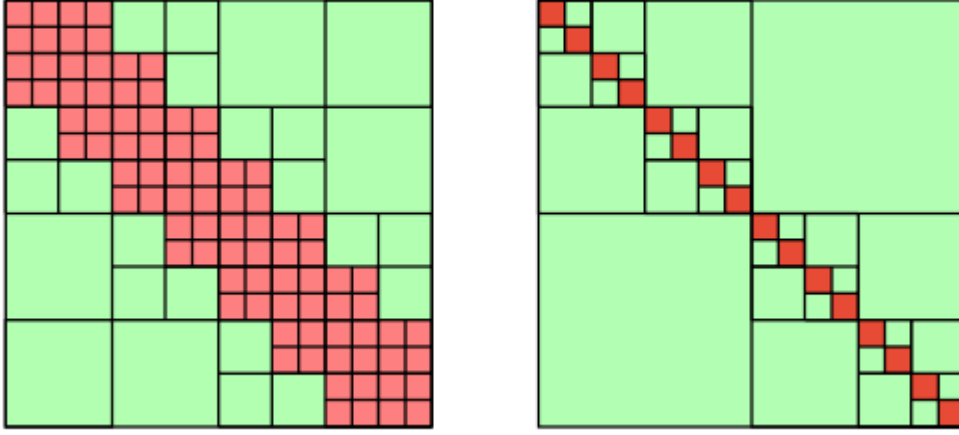
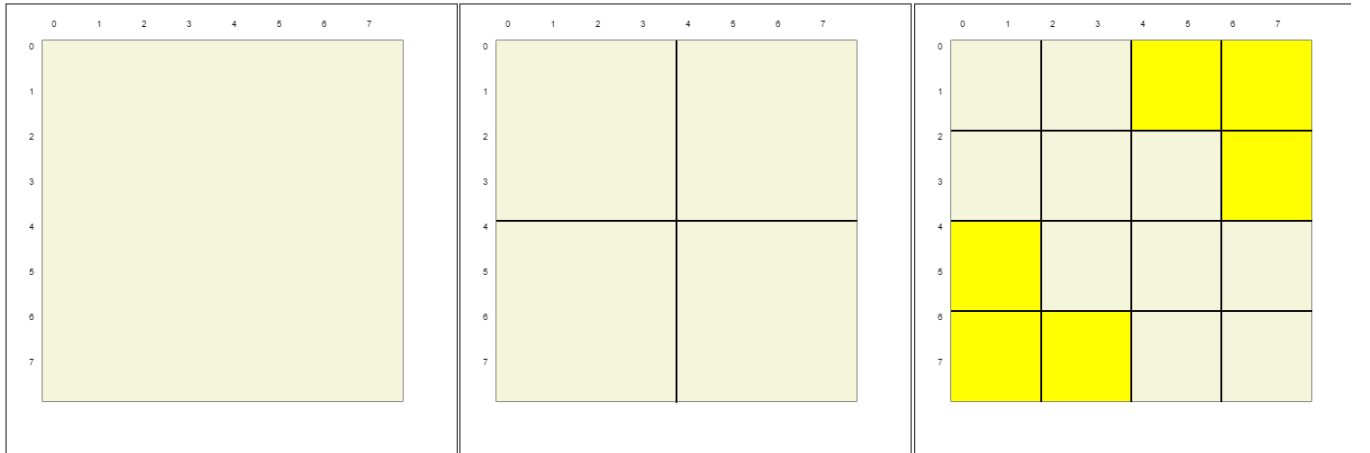


Рис. 3.2: Зліва - слабка допустимість, справа - стандартна.

3.6 Приклад побудови block cluster tree



Кроки побудови block cluster tree при $p=3$:

1. Коренем дерева є блок $\{0, 1, 2, 3, 4, 5, 6, 7\} \times \{0, 1, 2, 3, 4, 5, 6, 7\}$, який не задовільняє умову допустимості, тому що відповідною областю до множини індексів $\{0, 1, 2, 3, 4, 5, 6, 7\}$ є інтервал $[0, 1]$ і

$$\text{diam}([0, 1]) = 1 \leq 0 = \text{dist}([0, 1], [0, 1])$$

2. Чотирьма нащадками кореня в дереві $\mathbb{T}_{I \times I}$ є

$$\begin{aligned} &\{0, 1, 2, 3\} \times \{0, 1, 2, 3\}, \quad \{0, 1, 2, 3\} \times \{4, 5, 6, 7\}, \\ &\{4, 5, 6, 7\} \times \{0, 1, 2, 3\}, \quad \{4, 5, 6, 7\} \times \{4, 5, 6, 7\}. \end{aligned}$$

Жоден з них не задовільняє умову допустимості.

3. Після подальшого поділу, отримуємо такі блоки:

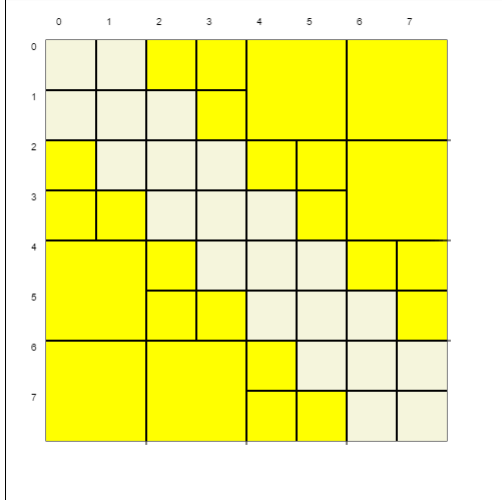
$$\begin{aligned} &\{0, 1\} \times \{0, 1\}, \quad \{0, 1\} \times \{2, 3\}, \quad \{0, 1\} \times \{4, 5\}, \quad \{0, 1\} \times \{6, 7\}, \\ &\{2, 3\} \times \{0, 1\}, \quad \{2, 3\} \times \{2, 3\}, \quad \{2, 3\} \times \{4, 5\}, \quad \{2, 3\} \times \{6, 7\}, \\ &\{4, 5\} \times \{0, 1\}, \quad \{4, 5\} \times \{2, 3\}, \quad \{4, 5\} \times \{4, 5\}, \quad \{4, 5\} \times \{6, 7\}, \\ &\{6, 7\} \times \{0, 1\}, \quad \{6, 7\} \times \{2, 3\}, \quad \{6, 7\} \times \{4, 5\}, \quad \{6, 7\} \times \{6, 7\}. \end{aligned}$$

Деякі з цих вузлів задовольняють умову допустимості, наприклад вузол $\{0, 1\} \times \{4, 5\}$, тому що відповідною областю є $[0, \frac{1}{4}] \times [\frac{1}{2}, \frac{3}{4}]$:

$$\text{diam}([0, \frac{1}{4}]) = \frac{1}{4} = \text{dist}([0, \frac{1}{4}], [\frac{1}{2}, \frac{3}{4}])$$

Вузли на діагоналі не задовольняють умову допустимості (відстань від відповідної області до себе самої рівна нулю) і деякі вузли не на діагоналі (наприклад $\{0, 1\} \times \{2, 3\}$) не задовольняють умову допустимості.

4. Нащадками цих вузлів є $\{(i, j)\}$ для індексів i, j . Кінцева структура буде:



Аналогічно можна побудувати block cluster tree для $p = 4$ чи $p = 5$.

4. Метод граничних елементів (БЕМ)

4.1 Модельна задача

Розглянемо застосування \mathcal{H} -матриць на прикладі одновимірного інтегрального рівняння Фредгольма першого роду. Нехай задано функцію $F : [0, 1] \rightarrow \mathbb{R}$. Шукаємо функцію $u : [0, 1] \rightarrow \mathbb{R}$, яка задовільняє наступне інтегральне рівняння:

$$\int_0^1 \ln |x - y| u(y) dy = F(x), x \in [0, 1]$$

де $g(x, y) = \ln |x - y|$ називається ядром інтегрального рівняння і має невизначеність на діагоналі $x = y$.

Використовуючи метод Гальоркіна, проектуємо дане рівняння на n -вимірний простір $V_n = \text{span}\{\varphi_0, \dots, \varphi_{n-1}\}$ і отримуємо:

$$\int_0^1 \int_0^1 \varphi_i(x) \ln |x - y| u(y) dy dx = \int_0^1 \varphi_i(x) F(x) dx$$

$$0 \leq i < n$$

Потрібно знайти u_n в просторі V_n :

$$u_n = \sum_{j=0}^{n-1} u_j \varphi_j$$

таке, що вектор коефіцієнтів u є розв'язком лінійної системи

$$Gu = f$$

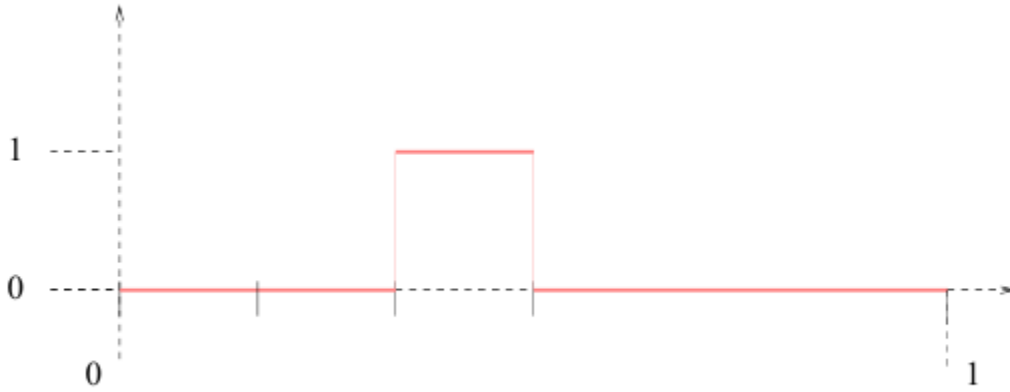
$$G_{ij} = \int_0^1 \int_0^1 \varphi_i(x) \ln |x - y| \varphi_j(y) dy dx$$

$$f_i = \int_0^1 \varphi_i(x) F(x) dx$$

В цьому прикладі визначаємо базисні функції як

$$\varphi_i(x) = \begin{cases} 1, & \text{якщо } \frac{i}{n} \leq x \leq \frac{i+1}{n} \\ 0, & \text{інакше} \end{cases}$$

які в загальному матимуть вигляд



Матриця G є щільною, тобто всі елементи не є нулями. Потрібно знайти наближену матрицю \tilde{G} , яка може бути збереженою в розрідженому форматі. Для того, щоб це досягти потрібно замінити ядро $g(x, y) = \ln |x - y|$ на розкладене ядро

$$\tilde{g}(x, y) = \sum_{v=0}^{k-1} g_v(x) h_v(y)$$

Таким чином, інтегрування за змінною x буде відкремленим від інтегрування за змінною y . Проте, ядро $g(x, y) = \ln |x - y|$ не можна наблизити розкладеним ядром на цілій області $[0, 1] \times [0, 1]$ (хіба що при великому k). Змість цього, ми будемо локальні наближення на підобластях $[0, 1] \times [0, 1]$, де g є гладкою.

4.2 Розклад ядра в ряд Тейлора

Нехай $\tau := [a, b]$, $\sigma := [c, d]$, $\tau \times \sigma \subset [0, 1] \times [0, 1]$ буде підобластю з властивістю $b < c$ і інтервали є роз'єднаними, тобто

$$\tau \cap \sigma = \emptyset$$

Тоді ядро є визначеним на $\tau \times \sigma$. $x_0 := (a + b)/2$

Лема 4.1 (Похідні $\ln |x - y|$) *Похідні $g(x, y) = \ln |x - y|$ для $x \neq y$ і $v \in \mathbb{N}$ мають вигляд*

$$\begin{aligned} \partial_x^v g(x, y) &= (-1)^{v-1} (v-1)! (x-y)^{-v} \\ \partial_y^v g(x, y) &= (v-1)! (x-y)^{-v} \end{aligned}$$

Лема 4.2 (Розклад Тейлора для $\ln |x - y|$) *Для будь-якого $k \in \mathbb{N}$ функція*

$$\tilde{g}(x, y) = \sum_{v=0}^{k-1} \frac{1}{v!} \partial_x^v g(x_0, y) (x - x_0)^v$$

наближає ядро $g(x, y) = \ln |x - y|$ з похибкою

$$|g(x, y) - \tilde{g}(x, y)| \leq (1 + \frac{|x_0 - a|}{|c - b|})(1 + \frac{|c - b|}{|x_0 - a|})^{-k}$$

Доведення. Нехай $x \in [a, b]$, $a < b$ і $y \in [c, d]$. В радіусі збіжності ряд Тейлора для ядра $g(x, y)$ в точці x_0 задовільняє

$$g(x, y) = \sum_{v=0}^{\infty} \frac{1}{v!} \partial_x^v g(x_0, y) (x - x_0)^v$$

Залишок $g(x, y) - \tilde{g}(x, y) = \sum_{v=k}^{\infty} \frac{1}{v!} \partial_x^v g(x_0, y) (x - x_0)^v$ може бути оцінений як:

$$\begin{aligned} \left| \sum_{v=k}^{\infty} \frac{1}{v!} \partial_x^v g(x_0, y) (x - x_0)^v \right| &= \left| \sum_{v=k}^{\infty} (-1)^{v-1} \frac{(v-1)!}{v!} \left(\frac{x - x_0}{x_0 - y} \right)^v \right| \\ &\leq \sum_{v=k}^{\infty} \left| \frac{x - x_0}{x_0 - y} \right|^v \leq \sum_{v=k}^{\infty} \left(\frac{|x_0 - a|}{|x_0 - a| + |c - b|} \right)^v \\ &= (1 + \frac{|x_0 - a|}{|c - b|})(1 + \frac{|c - b|}{|x_0 - a|})^{-k} \end{aligned}$$

Радіус збіжності покриває весь інтервал $[a, b]$.

Якщо $c \rightarrow b$, то оцінка залишку прямує до нескінченості і наближення буде як завгодно поганим. Проте, якщо замінити умову $b < c$ (диз'юнкція інтервалів) сильнішою умовою допустимості

$$diam(\tau) \leq dist(\tau, \sigma)$$

то похибка апроксимації може бути оцінена як

$$|g(x, y) - \tilde{g}(x, y)| \leq \frac{3}{2} (1 + \frac{2}{1})^{-k} = \frac{3}{2} 3^{-k}$$

Це означає, що ми отримуємо рівномірну оцінку для похибки наближення незалежно від інтервалів, якщо умова допустимості виконується. Похибка зростає експоненціально в залежності від порядку k .

4.3 Наближення низького рангу блоків матриці

Множина індексів $I = \{0, 1, \dots, n-1\}$ містить індекси базових функцій φ_i , які використовуються в дискретизації Галеркіна. Фіксуємо дві підмножини t і s множини індексів I та визначимо відповідні області:

$$\tau = \bigcup_{i \in t} supp(\varphi_i)$$

$$\sigma = \bigcup_{i \in s} supp(\varphi_i)$$

Якщо $\tau \times \sigma$ задовільняє умову допустимості, то ми можемо наблизити ядро g в цій підобласті рядом Тейлора \tilde{g} і замінити елементи матриці

$$G_{ij} = \int_0^1 \int_0^1 \varphi_i(x) g(x, y) \varphi_j(y) dy dx$$

використовуючи вироджене ядро $\tilde{g} = \sum_{v=0}^{k-1} g_v(x) h_v(y)$ для індексів $(i, j) \in t \times s$:

$$\tilde{G}_{ij} = \int_0^1 \int_0^1 \varphi_i(x) \tilde{g}(x, y) \varphi_j(y) dy dx$$

Розділяємо подвійний інтеграл на два звичайні

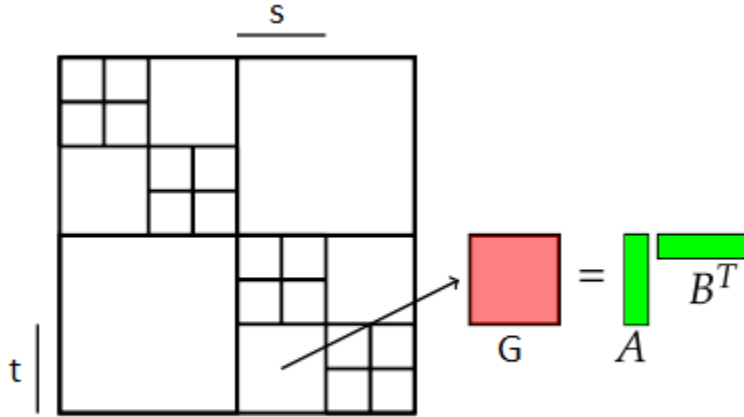
$$\begin{aligned} \tilde{G}_{ij} &= \int_0^1 \int_0^1 \varphi_i(x) \sum_{v=0}^{k-1} g_v(x) h_v(y) \varphi_j(y) dy dx \\ &= \sum_{v=0}^{k-1} \left(\int_0^1 \varphi_i(x) g_v(x) dx \right) \left(\int_0^1 \varphi_j(y) h_v(y) dy \right) \end{aligned}$$

Підматриця $G|_{t \times s}$ може бути записана в факторизованій формі

$$G|_{t \times s} = AB^\top, \quad A \in \mathbb{R}^{t \times \{0, \dots, k-1\}}, \quad B \in \mathbb{R}^{s \times \{0, \dots, k-1\}}$$

де елементами матриць A і B

$$A_{iv} := \int_0^1 \varphi_i(x) g_v(x) dx, \quad B_{jv} := \int_0^1 \varphi_j(y) h_v(y) dy$$



Матриця AB^\top має найбільший ранг k в незалежності від потужності s і t . Похибка наближення блоку матриці оцінена в наступній лемі.

Лема 4.3 *Поелементна похибка для елементів матриці G_{ij} апроксимується ядром \tilde{g} в допустимих блоках $t \times s$ (g в інших блоках) обмежена наступним чином*

$$|G_{ij} - \tilde{G}_{ij}| \leq \frac{3}{2} n^{-2} 3^{-k}$$

Доведення. $|G_{ij} - \tilde{G}_{ij}| = |\int_0^1 \int_0^1 \varphi_i(x)(g(x, y) - \tilde{g}(x, y)\varphi_j(y)dydx|$

$$\leq \int_0^1 \int_0^1 |\varphi_i(x)| \frac{3}{2} 3^{-k} |\varphi_j(y)| dydx$$

$$= \frac{3}{2} 3^{-k} \int_0^1 \varphi_i(x) dx \int_0^1 \varphi_j(y) dy$$

$$= \frac{3}{2} n^{-2} 3^{-k} \quad \blacksquare$$

Припустимо, що ми поділили множину індексів $I \times I$ над матрицею G на допустимі блоки, де застосовується апроксимація низького рангу, і недопустимі блоки, де використовуємо елементи матриці G .

$$I \times I = \bigcup_{v=1, \dots, b} t_v \times s_v$$

Глобальну похибку наближення оцінюємо, застосовуючи норму Фробеніуса

$$\|M\|_F^2 := \sum M_{ij}^2$$

Лема 4.4 *Похибка наближення $\|G - \tilde{G}\|_F$ для матриці \tilde{G} , побудованої за допомогою ядра \tilde{g} в допустимих блоках $t_v \times s_v$ та з допомогою g на недопустимих блоках, обмежена наступним чином*

$$\|G - \tilde{G}\|_F \leq \frac{3}{2} n^{-1} 3^{-k}$$

Постає питання, як поділити множину індексів $I \times I$ на допустимі та недопустимі блоки. Тривіальним поділом був би $\mathcal{P} := \{(i, j) | i \in I, j \in I\}$, де є тільки блоки розмірності 1×1 , ранг рівний 1. В цьому випадку, матриця \tilde{G} є ідентичною до G , але в цьому випадку ми не апроксимуємо матрицю у великих підблоках матрицями низького рангу.

5. Збірка, пам'ять і множення матриці на вектор

5.1 Побудова матриці

Ієрархічна матриця розкладається на допустимі і недопустимі листки дерева $\mathbb{T}_{I \times I}$. Для них створені два підкласи, опрацювання яких різняться.

5.1.1 Недопустимі листки

В недопустимих, але малих блоках $t \times s \subset I \times I$ обчислюємо елементи матриці (i, j) за формулою

$$\begin{aligned}\tilde{G}_{ij} &:= \int_0^1 \int_0^1 \varphi_i(x) \ln |x - y| \varphi_j(y) dy dx \\ &= \int_{i/n}^{(i+1)/n} \int_{j/n}^{(j+1)/n} \ln |x - y| dy dx\end{aligned}$$

Означення 5.1 (*репрезентація fullmatrix*) Кажуть, що матриця M розмірності $n \times m$ зберігається у вигляді fullmatrix, якщо її елементи M_{ij} зберігаються як дійсні числа у масиві довжиною mn в стовпцевому порядку

$$M_{11}, \dots, M_{n1}, M_{12}, \dots, M_{n2}, \dots, M_{1m}, \dots, M_{nm}$$

Порядок елементів матриці в репрезентації fullmatrix є таким самим, як і в стандартних пакетах лінійної алгебри (MATLAB, BLAS, LAPACK тощо).

Реалізація на мові програмування C#:

```
public class Fullmatrix
{
    public int rows;
    public int cols;
    public double[] e;
}
```

5.1.2 Допустимі листки

В допустимих блоках $t \times s \subset I \times I$ з відповідними областями $[a, b] \times [c, d]$ і $x_0 := (a + b)/2$ обчислюємо відповідну матрицю у факторизованій формі

$$\tilde{G}|_{t \times s} := AB^\top$$

$$A_{iv} := \int_{i/n}^{(i+1)/n} (x - x_0)^v dx$$

$$B_{jv} := \begin{cases} (-1)^{v+1} v^{-1} \int_{j/n}^{(j+1)/n} (x_0 - y)^{-v} dy, & \text{якщо } v > 0 \\ \int_{j/n}^{(j+1)/n} \ln |x_0 - y| dy, & \text{якщо } v = 0 \end{cases}$$

Підходящою репрезентацією для відповідної матриці $\tilde{G}|_{t \times s}$ є формат `rkmatrix` наведений нище.

Означення 5.2 (репрезентація `rkmatrix`) Кажуть, що матриця M розмірності $n \times m$ найбільшого рангу k зберігається у вигляді `rkmatrix`, якщо вона зберігається у факторизованій формі $M = AB^T$, де обидві матриці $A \in \mathbb{R}^{n \times k}$ і $B \in \mathbb{R}^{m \times k}$ зберігаються як масиви (в стовпцевому порядку).

Реалізація на мові програмування C#:

```
public class Rkmatrix
{
    public int k;
    public int rows;
    public int cols;
    public double[] a;
    public double[] b;
}
```

5.1.3 Репрезентація ієрархічної матриці

Означення 5.3 (репрезентація \mathcal{H} -матриці) Нехай $\mathbb{T}_{I \times I}$ - block cluster tree над множиною індексів I . Кажуть, що матриця $M \in \mathcal{H}(\mathbb{T}_{I \times I}, k)$ зберігається в \mathcal{H} -matrix репрезентації, якщо підматриці, що відповідають недопустимим листкам, зберігаються у вигляді `fullmatrix`, а ті, що відповідають допустимим листкам - у вигляді `rkmatrix`.

Однією можливою реалізацією \mathcal{H} -matrix репрезентації є зберігання допустимих і недопустимих блоків матриці в списку. Збірка і множення матриці на вектор робиться для кожного блоку окремо. Проте, ми використаємо іншу реалізацію, яка базується на структурі block cluster tree $\mathbb{T}_{I \times I}$ (не тільки на листках) і таким чином зберігає матрицю у більш структурованому вигляді.

Кожен блок $t \times s$ в дереві $\mathbb{T}_{I \times I}$ може бути

- листком - тоді відповідний блок матриці представлений у вигляді `fullmatrix` або `rkmatrix`.
- не листком - тоді блок $t \times s$ розкладають на його синів $t' \times s'$ з $t' \in S(t)$ та $s' \in S(s)$. Це означає матриця, що відповідає блоку $t \times s$ - *supermatrix* і вона складається з підматриць, що відповідають блоку $t' \times s'$.

Реалізація на мові програмування C#:

```

public class Supermatrix
{
    public int rows;
    public int cols;
    public int blockrows;
    public int blockcols;
    public Rkmatrix r;
    public Fullmatrix f;
    public Supermatrix[,] s;
}

```

$$M \in \mathbb{R}^{rows \times cols}$$

Матриця може бути

- rkmatrix - тоді

$$r \neq null, \quad f = null, \quad s = null$$

Матриця r - це репрезентація rkmatrix матриці M .

- fullmatrix - тоді

$$r = null, \quad f \neq null, \quad s = null$$

Матриця f - це репрезентація fullmatrix матриці M .

- supermatrix - тоді

$$r = null, \quad f = null, \quad s \neq null$$

Матриця s містить вказівники на підматриці $M_{i,j}$:

$$\begin{pmatrix} M_{1,1} & \dots & M_{1,blockcols} \\ \vdots & \ddots & \vdots \\ M_{blockrows,1} & \dots & M_{blockrows,blockcols} \end{pmatrix}$$

в порядку

$$M_{1,1}, \dots, M_{blockrows,1}, M_{1,2}, \dots, M_{blockrows,2}, \dots, M_{1,blockcols}, \dots, M_{blockrows,blockcols}$$

Реалізацією \mathcal{H} -матриці є дерево з вузлами, що реалізовані як *supermatrix*. На додаток, структура та ж сама, що і в block cluster tree $\mathbb{T}_{I \times I}$ (нащадки \equiv підматриці) і підматриці, що відповідають допустимим та недопустимим листкам, зберігаються в форматі rkmatrix і fullmatrix.

5.2 Метод спряжених градієнтів

Метод спряжених градієнтів (conjugate gradient method) - це алгоритм для чисельного розв'язування певних систем лінійних рівнянь, особливо тих в яких матриця є симетричною і додатньо визначеною. Часто релізують як ітераційний алгоритм, що застосовується до розріджених систем, які завеликі для розв'язування прямими методами (наприклад з допомогою методу Холецького).

Метод спряжених градієнтів вимагає від матриці тільки можливості помножити її на вектор, що дає можливість застосовувати спеціальні формати зберігання матриці.

Algorithm 2 Алгоритм методу спряжених градієнтів

```
 $r_0 := b - Ax$   
 $p_0 := r_0$   
 $k := 0$   
while true do  
   $\alpha_k := \frac{r_k^\top r_k}{p_k^\top A p_k}$   
   $x_{k+1} := x_k + \alpha_k p_k$   
   $r_{k+1} := r_k - \alpha_k A p_k$   
  if  $r_{k+1}$  достатньо мале then  
    вийти з циклу  
  end if  
   $\beta_k = \frac{r_{k+1}^\top r_{k+1}}{r_k^\top r_k}$   
   $p_{k+1} := r_{k+1} + \beta_k p_k$   
   $k := k + 1$   
end while  
результатом є  $x_{k+1}$ 
```

5.3 Програмна реалізація

Програмна реалізація на мові # в стані доробки. Працюють такі описані у роботі алгоритми(код в додатках):

1. побудова cluster tree для $n = 2^p$.
2. побудова block cluster tree.
3. множення \mathcal{H} —матриці на вектор.
4. реалізація методу спряжених градієнтів.

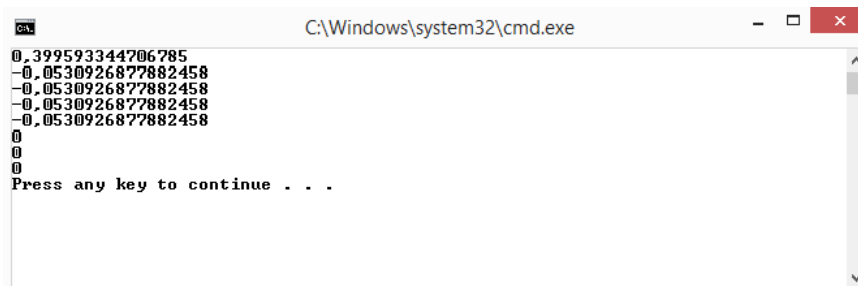


Рис. 5.1: Реалізація ВЕМ.

Висновок

В цій роботі розглянуто основні принципи побудови та використання \mathcal{H} -матриць. Описано такі ключові поняття як дерево, cluster tree та block cluster tree, які лягли в основу означення структури ієрархічної матриці. Також розглянуто відповідні структури через які ієрархічні матриці реалізують програмно: `rkmatrix`, `fullmatrix` та `supermatrix`. Наведено алгоритми реалізації block cluster tree і методу спряжених градієнтів.

Застосування \mathcal{H} -матриць розглянуто на прикладі методу скінченних елементів (ВЕМ). У даному прикладі застосовується метод Галеркіна та метод спряжених градієнтів.

Додатки

1. Побудова Cluster Tree

```
public static ClusterTree buildClusterTreeRec(int b, int e, int level,
int num, int[] arr)
{
    ClusterTree t = new ClusterTree();
    t.level = level;
    t.numberOfLeaf = num;
    int n = e - b + 1;
    t.leaf = new int[n];
    for (int i = 0; i < t.leaf.Length; i++)
        t.leaf[i] = i + b;
    int m = (int)Math.Pow(2, level) - arr[level];
    arr[level]--;
    if (n != 1)
    {
        t.leftTree = buildClusterTreeRec(b, (b + e + 1) / 2 - 1,
            level + 1, m, arr);
        m = (int)Math.Pow(2, level) - arr[level];
        arr[level]--;
        t.rightTree = buildClusterTreeRec((b + e + 1) / 2, e,
            level + 1, m, arr);
    }
    else
    {
        t.leftTree = null;
        t.rightTree = null;
    }
    return t;
}
```

2. Побудова Block Cluster Tree

```
public static Supermatrix BuildBlockClusterTree(ClusterTree t, ClusterTree s,
Supermatrix spr, int n)
{
    if (IsAdmissible(t, s))
```



```

{
    spr.s = null;
    spr.r = new Rkmatrix();
    spr.r.rows = t.leaf.Length;
    spr.r.cols = s.leaf.Length;
    spr.r.k = spr.r.rows;
    spr.r.a = new double[spr.r.rows * spr.r.cols];
    spr.r.b = new double[spr.r.rows * spr.r.cols];
    FillRkmatrix(t,s,out spr.r.a,out spr.r.b);
    return spr;
}
else
{
    if (t.leaf.Length != 1)
    {
        spr.rows = n;
        spr.cols = n;
        spr.blockrows = 2;
        spr.blockcols = 2;
        spr.s = new Supermatrix[spr.blockrows, spr.blockcols];
        for (int i = 0; i < spr.s.GetLength(0); i++)
        {
            for (int j = 0; j < spr.s.GetLength(1); j++)
                spr.s[i, j] = new Supermatrix();
        }
        spr.s[0, 0] = BuildBlockClusterTree(t.leftTree,
            s.leftTree, spr.s[0, 0],n);
        spr.s[0, 1] = BuildBlockClusterTree(t.rightTree,
            s.leftTree, spr.s[0, 1],n);
        spr.s[1, 0] = BuildBlockClusterTree(t.leftTree,
            s.rightTree, spr.s[1, 0],n);
        spr.s[1, 1] = BuildBlockClusterTree(t.rightTree,
            s.rightTree, spr.s[1, 1],n);
    }
    else
    {
        spr.rows = n;  spr.cols = n;
        spr.s = null;
        spr.f = new Fullmatrix();
        spr.f.cols = 0;  spr.f.rows = 0;
        FillFullmatrix(t,s,out spr.f.e);
    }
    return spr;
}
}

```

3. Множення \mathcal{H} -матриці на вектор

```

static public double[] MultHMatrixByVector(Supermatrix spr, double[] vct)
{
    if (spr.s != null)
    {
        int n=vct.Length;
        double[] vct1=new double[(int)(n/2)];
        double[] vct2=new double[(int)(n/2)];
        for (int i=0;i<n/2;i++){
            vct1[i]=vct[i];
            vct2[i]=vct[i+(int)(n/2)];
        }
        double[] a1=MultHMatrixByVector(spr.s[0, 0], vct1);
        double[] a2=MultHMatrixByVector(spr.s[0, 1], vct2);
        double[] b1=MultHMatrixByVector(spr.s[1, 0], vct1);
        double[] b2=MultHMatrixByVector(spr.s[1, 1], vct2);
        double[] res = new double[n];
        for (int i = 0; i < (int)n / 2; i++)
        {
            res[i] = a1[i] + a2[i];
            res[i + (int)n / 2] = b1[i] + b2[i];
        }
        return res;
    }
    else
    {
        if (spr.r != null)
        {
            double[,] tempa=new double[spr.r.rows,spr.r.cols];
            double[,] tempb=new double[spr.r.rows,spr.r.cols];
            int k = 0;
            for (int i = 0; i < spr.r.rows; i++)
            {
                for (int j = 0; j < spr.r.cols; j++)
                {
                    tempa[i, j] = spr.r.a[k];
                    tempb[i, j] = spr.r.b[k];
                    k++;
                }
            }
            double[] first = GradientMethod.
                MultiplyMatrixByVector(tempb, vct);
            double[] second = GradientMethod.
                MultiplyMatrixByVector(tempa, first);
            return second;
        }
        else if (spr.f != null)
        {

```

```

        double[] res = new double[1];
        res[0]=GradientMethod.MultiplyVectorByVector(spr.f.e, vct);
        return res;
    }
}

```

4. Реалізація алгоритму методу спряжених градієнтів

```

public static double[] ConjugateGradientMethodHMatrix(Supermatrix a,
double[] b, double[] x0)
{
    int n = b.Length;
    double[] x1 = new double[n];
    double[] temp = BlockClusterTree.MultHMatrixByVector(a, x0);
    double[] r0 = new double[n];
    double[] p0 = new double[n];
    double[] r1 = new double[n];
    double[] p1 = new double[n];
    double alphak = 0;
    double betak = 0;
    for (int i = 0; i < n; i++)
        r0[i] = b[i] - temp[i];
    p0 = r0;
    int k = 0;
    while (k != n)
    {
        double temp1 = MultiplyVectorByVector(r0, r0);
        double[] temp2 = BlockClusterTree.MultHMatrixByVector(a, p0);
        double temp3 = MultiplyVectorByVector(temp2, p0);
        alphak = temp1 / temp3;
        for (int i = 0; i < n; i++)
        {
            x1[i] = x0[i] + alphak * p0[i];
            r1[i] = r0[i] - alphak * temp2[i];
        }
        betak = MultiplyVectorByVector(r1, r1) /
            MultiplyVectorByVector(r0, r0);
        for (int i = 0; i < n; i++)
            p1[i] = r1[i] + betak * p0[i];
        p0 = p1;
        r0 = r1;
        x0 = x1;
        k++;
    }
    return x1;
}

```

Бібліографія

- [1] Steffen Börm *Hierarchical Matrices* / Lars Grasedyck, Wolfgang Hackbusch — електронний ресурс, 2005. — 136 с.
- [2] Mohammad Izadi *Hierarchical Matrix Techniques on Massively Parallel Computers. Dissertation*—К.: Max Planck Institute for Mathematics in the Sciences,2012.— 212 с.
- [3] Нікольський Ю.В. *Дискретна математика* / Пасічник В.В., Щербина Ю.М.— К.: Видавнича група BHV,2007.—368 с.
- [4] Wolfgang Hackbusch *Hierarchical Matrices: Algorithms and Analysis*—К.: Springer-Verlag,2015.— 505 с.
- [5] Lin Lin *Fast construction of hierarchical matrix representation from matrix-vector multiplication* / Jianfeng Lu, Lexing Ying — К.: Journal of Computational Physics 230 4071–4087,2011.—17 с.
- [6] Jonathan Richard Shewchuk *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain* —К.:School of Computer Science, Pittsburg,PA 15213,1994.—64 с.