Софийски университет
"Св. Климент Охридски"

Факултет по математика и информатика

Катедра числени методи и алгоритми

Дипломна работа

на тема:

Рандомизирани квази-Монте Карло алгоритми за намиране на екстремални собствени стойности

Автор:
Александър Огнянов Маразов
Специалност: Изчислителна математика и математическо моделиране
Факултетен номер: М23074

Ръководител:
Проф. Анета Караиванова
Институт по информационни и комуникационни технологии-БАН

4 ноември 2013 г.

Съдържание

1 Въведение **3**
1.1 Намиране на собствени стойности 4
1.1.1 Кратък обзор на Монте Карло Методите за собствени стойности 5
1.1.2 Степеннен метод 5
1.1.3 Метод на Арнолди 7

2 Разбъркани квазислучайни редици **8**
2.1 Квазислучайни редици 8
2.1.1 Редица на Собол 10
2.1.2 Редица на Холтън 13
2.1.3 Редица на Фор 14
2.1.4 (t, m, s)-мрежи и (t, s)-редици 14
2.2 Разбъркани квазислучайни числа 16

3 Степенен метод с Монте Карло итерации **21**
3.1 Степенният метод и квази-Монте Карло методите 22
3.1.1 Балансиране на грешката и сложност 24

4 Числени експерименти **25**
4.1 Използвани техологии и библиотеки 25
4.2 Конструиране на тестовите матрици 26
4.3 Резултати от експериментите 26

5 Заключение **31**

A Степенен метод с Монте Карло итерации на Python **32**

Абстракт

Рандомизираните квази-Монте Карло методи комбинират предимствата на Монте Карло и квази-Монте Карло методите. Те се базират на разбъркани квазискучайни редици. Рандомизацията има две основни предимства. Първо, тя предоставя практичен метод за оценка на грешката, като третира всяка редица с малък дискрепанс като различна и независима реализация на фамилия от случайни редици с малък дискрепанс. По този начин рандомизираните квази-Монте Карло методи преодоляват основния си недостатък, запазвайки реда на сходимост на квази-Монте Карло методите. На второ място, разбъркването ни дава възможност по прост и унифициран начин да генерираме поредица от квазислучайни редици за паралелни и разпределни изчисления. В същото време, използването на разбъркани редици запазва предимствата на квази-Монте Карло методите по отношение на точността на пресмятанията и скоростта на сходимост. Това ни мотивира да приложим разбъркваните квазислучайни редици за да приближено намиране на доминантни собствени стойности.

Целта на настоящата дипломна реабота е изследване на различни варианти на разбъркване на редицата на Фор и приложението им за приближено намиране на доминантната собствена стойност на матрица с рандомизиран квази-Монте Карло метод . Получените резултати са сравнени с резултатите от Монте Карло и квази-Монте Карло метод.

Означения

Числови множества

$\mathbb{C}$ комплексни числа

$\mathbb{R}$ реални числа

Матрици

Векторите са вектори-колони и са означени с малки латински букви: $x, y, \ldots$.
Матриците са означени с големи латински букви: $A, B, \ldots$.
и за техните елементи се използва Matlab означение: за матрицата A

$A(i,j)$ е елемента на i-ти ред и j-та колона

$A(k:l,j)$ е вектора-колона с елементи от j-тата колона от позиции k до l

$A(i,k:l)$ е вектора ред от i-тия ред, колони от k до l

$A(:,j)$ е j-тата колона

$A(i,:)$ е i-тия ред

A^H е комплексно спрегнатата матрица на A

A^T е транспонираната матрица на A

$\langle x, y \rangle$ е скаларното произведение равно на $x^H y$.

Вероятности

$\mathbb{P}(E)$ вероятноста на събитието E

$\mathbb{E}(X)$ математическото очалване на случайната величина X

Глава 1

Въведение

Монте Карло методите са едни от най-широко използваните числени методи за решаване на задачи в областите статистическа физика, статистическа химия, биология и др. Тяхното основно предимство се проявява при решаване на задачи с много голяма размерност.

Стохастичните числени методи (Монте Карло методи) се основават на симулация на случайни величини и случайни процеси. Те имат проста конструкция и често се използват за моделиране на процеси, чието поведение може да бъде оценено само в стохастичен смисъл.

Квази-Монте Карло методите са детерминистични числени методи, които използват редица с малък дискрепанс (квазислучани редици) за оценка.

Рандомизираните квази-МонтеКарло методи използват рандомизирани редици с малък дискрепанс и комбинират предимствата на Монте Карло методите и кавази-Монте Карло методите.

Прилагането на Монте Карло метод включва няколко етапа:

- преформулиране на задачата
- дефиниране на случайна величина или случаен процес, чието математическо очакване съвпада с търсеното решение
- приближена оценка на статистическите характерисики (дисперсия, очакване)

Без ограничение на общността, разглеждаме интеграл от функция f върху s-мерния единичен куб $[0,1)^s$. Интегралът на функцията $f \cdot q$ може да интерпретираме като очакване от $f(X)$, като случайната величина X има плътност на разпределение q.

$$I(f) = \mathbb{E}\left(f(X)\right) = \int_{[0,1)^s} f(x)q(x)dx \tag{1.1}$$

В обикновения Монте Карло алгоритъм генерираме N реализации на случайната величина X: $x_1, \ldots, x_N$, с разпределение q. Оценката за I тогава е

$$\hat{I}(f) = \hat{I}_N = \frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{1.2}$$

Силният закон за големите числа ни дава

$$\mathbb{P}\left(\lim_{N\to\infty} \hat{I}_N = I\right) = 1 \tag{1.3}$$

Редът на сходимост при такъв подход е $O(N^{-1/2})$, което може да е значително по-бавно от други методи. Действително, ако f има крайна дисперсия $\sigma^2 = \mathrm{Var}(f(x))$, то

$$\mathbb{E}\left((\hat{I}_N - I)^2\right) = \frac{\sigma^2}{N}$$

В много ситуации фактът, че с един и същи Монте Карло алгоритъм могат да се решат различни варианти на една задача надделява над допълнителното време нужно за изчисления.

Квази-Монте Карло методите се различават от обикновените Монте Карло методи по това, че при генерирането на извадкта на случайната величина не се ползва генератор на псевдослучайни числа, а генератор на квазислучайни числа. Това подобрява реда на асимптотичната грешка до $O(N^{-1}(\log N)^s)$, където s е размерността на задачата (вж. глава 5 на [20]).

В настоящата дипломна работа ще представя приложение на Монте Карло алгоритмите за намиране на екстремални собствени стойности. Приложението на Монте Карло и квази-Монте Карло алгоритми са описани от А. Караиванова [12]. Настоящата работа има за цел да разшири изследването с алгоритъм, който ползва рандомизирани квазислучайни числа.

Глава 2 е въведение в квазислучайните редици. В глава 3 е представен метода на степенната итерация за намиране на собствени стойности с Монте Карло стъпки. Описание на използваните библиотеки и написан код, като и резулатите от експеримента са описани в глава 4. В остатъка от тази глава ще опиша задачата за намиране на собствени стойности и методи за решаването й.

1.1 Намиране на собствени стойности

Много задачи от най-различни сфери могат да бъдат сведени до намирането на собствените стойности и вектори на някоя матрица $A \in \mathbb{C}^{n\times n}$. Собствените стойности ще означаваме с λ_k, а съответните собствени вектори с $x^{(k)}$:

$$Ax^{(k)} = \lambda_k x^{(k)}, \text{ за } k = 1, \ldots, n$$

За намирането на собствените стойности на малка матрица се позват методи базирани на факторизацията на матрици, например QR -алгоритъма. Тъй като факторизацията изисква $O(n^3)$ операции (вж. например [8]) те не са подходящи за големи матрици.

Ако матрицата A е много голяма, ние рядко се интересуваме от всички собствени стойности. Често свойствата на системата, описана чрез матрицата, се определят от най-малката и най-голямата собствена стойност: λ_n и λ_1. Тези стойности определят и числото на обусловеност за норамлна матрицата A:

$$\mathrm{cond}(A) = \left|\frac{\lambda_1}{\lambda_n}\right|$$

1.1.1 Кратък обзор на Монте Карло Методите за собствени стойности

Въпреки че едни от първите публикации в областта на Монте Карло методите от 50-те и 60-те години на миналия век се отнасят за задачи на линейната алгебра [7], те не разглеждат намирането на собствени стойности.

Монте Карло методите за намиране на доминантната собствена стойност са предложени от Михайлов (1967) и Собол (1973), а за най-малката – от Михайлов през 1987.

Степенният метод с Монте Карло итерация е предложен от Димов и Караиванова в серия публикации през периода 1995-1998.

Първите работи в световната литература по приложение на квази-Монте Карло методи за намиране на екстремални собствени стойности са на Караиванова и Маскани в периода 2001-2002.

1.1.2 Степеннен метод

Ще разгедаме степенния метод за определяне на най-голямата собствена стойност и собствен вектор съответстащ на тази стойност. Този метод, както и други методи за намиране на собствени стойности могат да бъдат намерени в Golub и Loan [8]. Основния резултат на алгритъма се съдържа в следната теорема:

Теорема 1 *Нека $A \in \mathbb{C}^{n\times n}$ е такава, че*

1. *A има n линейно независими собствени вектора $x^{(k)}$, съответващи на собствени стойности λ_k, $k = 1, \ldots, n$.*
2. *собствените стойности удовлетворяват неравенствата*

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_{n-1}| \geq |\lambda_n|$$

Ако $y_0 \in \mathbb{C}^n$ е начален вектор с представяне

$$y_0 = \sum_{k=1}^{n} a_k x^{(k)}$$

като $a_1 \neq 0$, тогава за $y_{l+1} = Ay_l$, $l = 0, 1, 2, \ldots$, имаме

$$\lim_{l\to\infty} \frac{A^l y_0}{\lambda_1^l} = cx^{(1)}$$

за някое $c \neq 0$. Означавайки скаларното произведение в $\mathbb{C}^n$ с $\langle x, y\rangle = x^H y$ имаме

$$\lim_{l\to\infty} \frac{\langle h, A^l y_0\rangle}{\langle h, A^{l-1} y_0\rangle} = \lambda_1 \tag{1.4}$$

Тъй като $y_l = A^l y_0$ и $A^l x^{(k)} = \lambda_k^l x^{(k)}$

$$y_l = a_1 \lambda_1^l x^{(1)} + \sum_{k=2}^{n} a_k \lambda_k^l x^{(k)}$$

давайки

$$\frac{y_l}{\lambda_1^l} = a_1 x^{(1)} + \sum_{k=2}^{n} a_k \left(\frac{\lambda_k}{\lambda_1}\right)^l x^{(k)}$$
$$= a_1 x^{(1)} + O\left(\left(\frac{\lambda_2}{\lambda_1}\right)^l\right) \tag{1.5}$$

Понеже

$$1 > \left|\frac{\lambda_2}{\lambda_1}\right| \geq \left|\frac{\lambda_3}{\lambda_1}\right| \geq \cdots \geq \left|\frac{\lambda_n}{\lambda_1}\right|$$

Сходимостта е доминирана от члена $\frac{\lambda_2}{\lambda_1}$. Следователно имаме

$$\lim_{l\to\infty} \frac{A^l y_0}{\lambda_1^l} = a_1 x^{(1)}$$

За да докажем второто твърдение, забелязваме, че тъй като $y_l = \sum_{k=1}^{n} a_k \lambda_k^l x^{(k)}$ имаме

$$\langle h, A^l y_0\rangle = \sum_{k=1}^{n} a_k \lambda_k^l \langle h, x^{(k)}\rangle$$

$$\frac{\langle h, A^l y_0\rangle}{\langle h, A^{l-1} y_0\rangle} = \frac{a_1\lambda_1^l\langle h, x^{(1)}\rangle + \sum_{k=2}^{n} a_k\lambda_k^l\langle h, x^{(k)}\rangle}{a_1\lambda_1^{l-1}\langle h, x^{(1)}\rangle + \sum_{k=2}^{n} a_k\lambda_k^{l-1}\langle h, x^{(k)}\rangle}$$
$$= \lambda_1 \left(\frac{a_1\langle h, x^{(1)}\rangle + \sum_{k=2}^{n} a_k \left(\frac{\lambda_k}{\lambda_1}\right)^l \langle h, x^{(k)}\rangle}{a_1\langle h, x^{(1)}\rangle + \sum_{k=2}^{n} a_k \left(\frac{\lambda_k}{\lambda_1}\right)^{l-1} \langle h, x^{(k)}\rangle}\right)$$
$$= \lambda_1 + O\left(\left(\frac{\lambda_2}{\lambda_1}\right)^l\right) \tag{1.6}$$

От 1.5 виждаме, че

$$y_l = A^l y_0 \approx c\lambda_1^l x^{(1)}$$

Така y_l клони към $x^{(1)}$ (с точност до умножение със скалар). Но ако $|\lambda_1| > 1$, то $||y_l|| \to +\infty$ при $l \to \infty$, доакто ако $|\lambda_1| < 1$, то $||y_l|| \to 0$. Това води до числени проблеми при изчисляването на итерациите. За да преодолеем тази трудност, на всяка стъка ще скалираме вектора запазвайки нормата му константна, без значение каква норма сме избрали.

Алгоритъм 1: степенна итерация

$y_0 := y_0/||y_0||$ # нормализираме y_0 до единична норма
$l := 0$
повтаряй до сходимост
начало
  $z_{l+1} := Ay_l$
  $y_{l+1} := z_{l+1}/||z_{l+1}||$
  $\lambda^{(l+1)} := \langle y_{l+1}, Ay_{l+1}\rangle$
  $l := l + 1$
край

За съжаление нормализирането не е лесно постижимо при Монте Карло варианта на този алгоритъм, както ще видим в следващите глави.

Изпозвайки теорема 1. лесно се вижда, че $y_l \to cx_1$ и $\lambda^{(l)} \to \lambda_1$.

Нека дефинираме $A_\mu = A - \mu I$, където I е единичната $n \times n$ матрица и μ е параметър на отместваме. Нека $A = S\Lambda S^{-1}$, където Λ е Жордановата нормална форма на A. Тогава,

$$S^{-1} A_\mu S = \Lambda - \mu I$$

И A_μ^{-1} има собствени стойности

$$\gamma_k = \frac{1}{\lambda_k - \mu}, \text{ for } k = 1, \ldots, n$$

Прилагайки степенния метод към A_μ^{-1} получаваме алгоритъм за намиране на собсвения вектор и собствена стойност най-близки до μ. Алгоритъмът често се нарича обратна итерация(*inverse iteration*).

Алгоритъм 2: обратна итерация

$y_0 := y_0/||y_0||$ # нормализираме y_0 до единична норма
$l := 0$
повтаряй до сходимост
начало
    $A_\mu z_{l+1} := y_l$
    $y_{l+1} := z_{l+1}/||z_{l+1}||$
    $\gamma^{(l+1)} := \langle y_{l+1}, A y_{l+1} \rangle$
    $l := l + 1$
край

Нека разгледаме изчислителната сложност на двата алгоритъма. Ако са необходими k итертации, то сложността на обикновения степенен метод е $O(kn^2)$, а на обратния е $O(n^3 + kn^2)$. Изчислителната сложност на Монте Карло варианта е $O(dm)$, където d е средния брой на състояния във веригата, а m е броя на веригите.

1.1.3 Метод на Арнолди

Стандартният алгоритъм за намиране на собствени стойности и собствени вектори на големи разредени матрици е метода на Арнолди. Повече информация може да бъде намерена в [8].

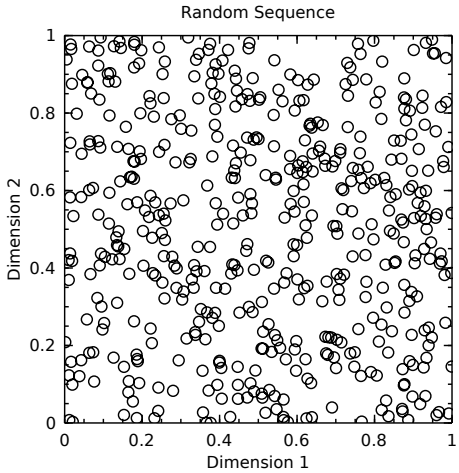
Глава 2

Разбъркани квазислучайни редици

Тази глава е въведение в квазислучайните редици. В част 2.1 въвеждаме основните понятия нужни за анализа на числови редици. Представени са няколко квазислучайни редици, включително редиците на Собол, Холтън и Фор. В глава 2.2 представяме техники за разбъркване на квази случайни редици.

2.1 Квазислучайни редици

Случайните и псевдослучайните числа имат свойството да се скупчват в определени области, докато други области остават незапълнени както може да се види на фигура 2.1. От друга страна, когато генерираме край-



Фигура 2.1: Двуизмерна равномерно разпределна случайна редица. Има големи незапълнени петна.

на извадка за нуждите на Монте Карло оценка, ние искаме точките да

покриват интегруемата област възможно най-равномерно. В идеалния случай точките трябва да са равномерно разпределени във всеки паралелотоп в $[0,1)^s$. При крайни извадки винаги можем да направим паралелотопите достатъчно фини, така че някои от тях да са празни. Степента на равномерно разпределение на числата на една редица е проблем, който е изучаван от Вейл [19], който пръв изследва равномерното разпределение. Kuipers и Niederreiter [14] обобщават тази теория. Мярката на това, колко равномерно са разпределени точките на една редица се нарича дискрепанс. Една от дефинициите за дискрепанс е така наречения *звезда-дискрепанс*

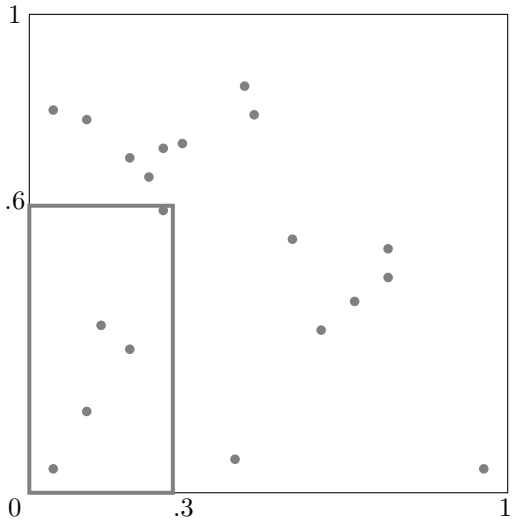
Дефиниция 1 *За всяка редица* $(x_i)_{i=1}^N \in [0,1)^s$ *с* N *елемента и множества* $J(\nu) = [0,\nu_1) \times [0,\nu_2) \times \ldots \times [0,\nu_s)$ *дефинираме звезда-дискрепанс като*

$$D_N^\star = \sup_{0 \le \nu_k < 1} \left| \frac{1}{N} \#\{x_i \in J(\nu)\} - \prod_{k=1}^{s} \nu_k \right| \tag{2.1}$$

Дефиниция 2 *Казваме, че редицата* (x_i) *има* малък дискрепанс *ако*

$$D_N^\star \le C_s \frac{(\log(N))^s}{N}, \tag{2.2}$$

където C_s *е константа, която зависи само от размерността* s*.*



Фигура 2.2: На фигурата за изобразени 20 точки в единичния квадрат и кутия от $(0,0)$ до $(0.3,0.6)$. Кутията има обем 0.18 и съдържа $5/20 = 0.2$ от точките, следователно разликата е $|0.18 - 0.2| = 0.02$.

За $x_k \in U[0,1)^s$ равномерно разпределени в s-мерния единичен куб Chung [4] показва, че

$$\limsup_{N\to\infty} \frac{\sqrt{2N} D_N^\star}{\sqrt{\log(\log(N))}} = 1, \text{ с вероятност } 1 \tag{2.3}$$

Така $D_N^\star = O(\sqrt{\log(\log(N))}/\sqrt{N})$, което поради факта, че $\log(\log(N))$ е почти константа, $D_N^\star$ има поведение като $O(N^{-1/2})$. Затова, редиците с малък дискрепанс представляват значително подобрение на равномерното разпределение. Предполага се, но не е доказано, че редици с дискрепанс $D_N^\star = o(\log(N)^s/N)$ не могат да бъдат конструирани.[1]

Има връзка между по-малкия дискрепанс и по-точното Монте Карло интегриране. Коксма доказва следното неравенство на Коксма-Хлавка в едномерния случай (1942), а Хлавка – в многомерния[10]:

$$\left| \frac{1}{N} \sum_{i=1}^{N} f(x_i) - I(f) \right| \leq V(f) D_N^\star \tag{2.4}$$

където $V(f)$ е пълната вариация на f, в смисъла на Харди и Краус. Просто и елегантно доказателство може да бъде намерено в Калфиш [3]. На практика това неравенство не е от особена практична стойност, тъй като намирането на дискрепанса на една редица е трудно. Намирането на пълната вариация на една функция в общия случай е дори още по-трудно.

След като сме дефинирали нужните характеристики за описанието на квазислучайните числа, следва да построим няколко квазислучайни редици.

Един от най-простите генератори на квазислучайни числа е

$$x_k = \{k\alpha\}$$

Където $\{u\} = u - \lfloor u \rfloor$ дробната част на u, $\alpha = (\alpha_1, \alpha_1, \ldots, \alpha_s)$ и $1, \alpha_1, \alpha_1, \ldots, \alpha_s$ са линейно независими над полето на рационалните числа. Тази редица покрива единичния хиперкуб, както е показал Веийл [19].

За редици, които са периодични и достатъчно гладки добър избор на квазислучайна редица са решетки за интегриране. Те имат вида:

$$x_k = \{\frac{k}{N} g\},\ k = 0, \ldots, N-1 \tag{2.5}$$

където $g = (g_1, \ldots, g_s)$ е пораждащ вектор, $\{\cdot\}$ е дробната част приложена по компонентите на вектора. Корабов [13] използва $g = (1, a, a^2, \ldots, a^{s-1}) \mod N$ за $1 \leq a \leq N-1$ и НОД$(a, n) = 1$.

Проблемът с интегралните решетки е, че периодичната им структура води до нежелани ефекти. В следващите секции ще представя квазислучайните редици на Собол (Sobol) (1967, 1976), Холтън (Halton) (1960) и Фор (Faure) (1980).

2.1.1 Редица на Собол

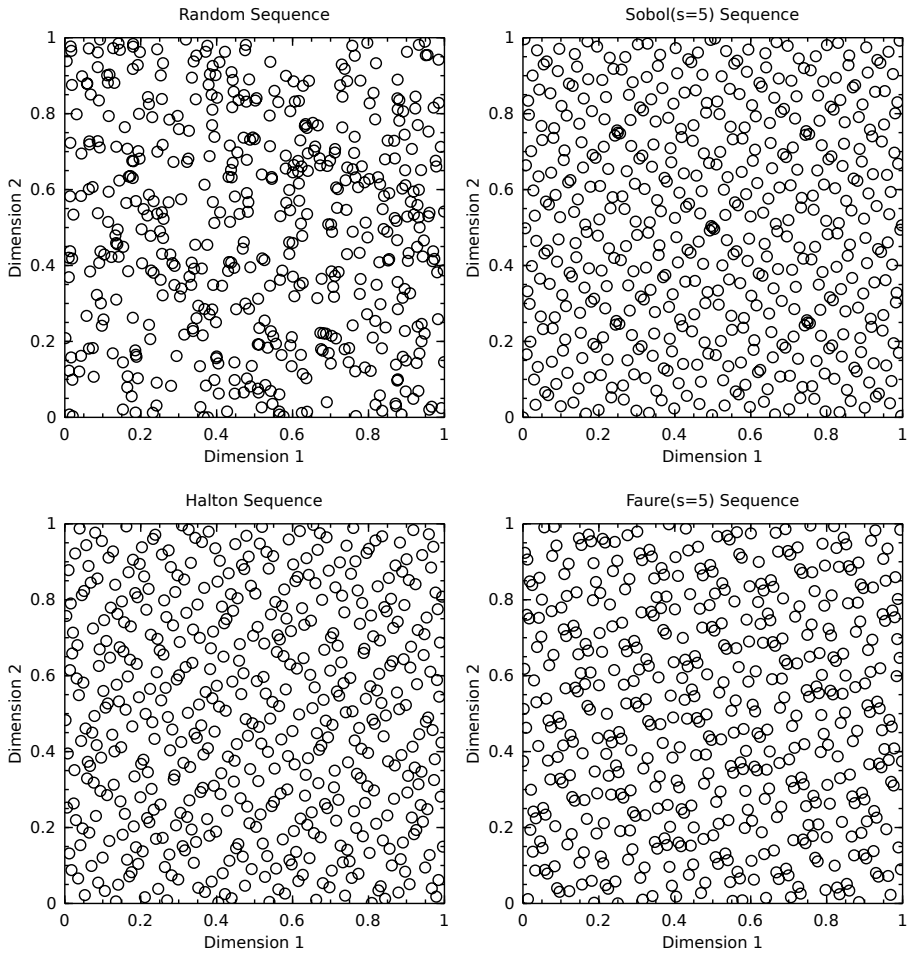
Нека двоичното представяне на k е

$$k = \sum_{j=1}^{w} b_j 2^{j-1} \tag{2.6}$$

Където $b_j \in \{0, 1\}$. i-тата координата на k-тата точка на редицата на Собол се дефинира като

$$x_k^{(j)} = b_1 v_1^{(j)} \oplus b_2 v_2^{(j)} \oplus \cdots \oplus b_w v_w^{(j)} \tag{2.7}$$

[1] вж. Beck и Chen [2]



Фигура 2.3: Проекция на първите две измерения на квазислучайните редици на Sobol, Halton и Faure.

Където $\oplus$ е логическата операция изключващо или (XOR). $v_i^{(j)}$ е двоична дроб наричана i-то направляващо число на j-тото измерение. За да получим $v_i^{(j)}$ за всяко $j \leq n$ избираме примитивен полином[2] над полето $\mathbb{Z}_2$ от цели числа по модул 2 :

$$P_j(x) = x^{d_j} + a_1^{(j)} x^{d_j - 1} + \cdots + a_{d_j - 1}^{(j)} x + 1 \tag{2.8}$$

След като сме избрали примитивен полином неговите коефициенти се използват да се дефинира $v_i^{(j)}$ със зависимостта

$$v_i^{(j)} = a_1^{(j)} v_{i-1}^{(j)} \oplus a_2^{(j)} v_{i-2}^{(j)} \oplus \cdots \oplus a_{d_j-1}^{(j)} v_{i-d_j+1}^{(j)} \oplus v_{i-d_j}^{(j)} \oplus \frac{v_i^{(j)}}{2^{d_j}}, \quad i > d_j \tag{2.9}$$

Ако дефинираме $v_i^{(j)} = m_i^{(j)} / 2^{d_j}$, където m_i е нечетно цяло число и $0 < m_i < 2^{d_j}$, рекурентната връзка може да бъде записан като

$$m_i^{(j)} = 2 a_1^{(j)} m_{i-1}^{(j)} \oplus 2^2 a_2^{(j)} m_{i-2}^{(j)} \oplus \cdots \oplus 2^{d_j - 1} a_{d_j-1}^{(j)} m_{i-d_j+1}^{(j)} \oplus 2^{d_j} m_{i-d_j}^{(j)} \oplus m_i^{(j)}$$

Което включва само целочислени операции. Стойностите на $m_1^{(j)}, m_2^{(j)}, \ldots, m_d^{(j)}$ могат да бъдат произволно избрани стига да са нечетни и $m_i^{(j)} < 2^I$.

Има още един начин за конструиране на редиците на Собол, който се основава на кодове на Грей. Редицата конструирана от Антонов и Салеев [1] има асимптотино същия дискрепанс като редицата на Собол построена в (2.7).

$$x_n^{(j)} = g_1 v_1^{(j)} \oplus g_2 v_2^{(j)} \oplus \cdots, \text{ for } j = 1, \ldots, s \tag{2.10}$$

Където $\ldots g_3 g_2 g_1$ е двоичното представяне на k в код на Грей. Кодовете на грей имат следните свойства:

1. Използвайки двоично представяне (2.6) кода на Грей на k е

$$\ldots g_3 g_2 g_1 = \ldots b_3 b_2 b_1 \oplus \ldots b_4 b_3 b_2$$

2. Кодовете на Грей за k и $k+1$ се различават само в една цифра. За да намерим коя е тази цифра, трябва да намерим най-дясната нула в двоичното представяне на k.

Ако използваме тези свойства може да видим, че за да изчисим $x_{n+1}^{(j)}$ са ни нужни само XOR операции в 2.10 на позиция c където b_c е най-десния нулев бит на k.

$$x_{n+1}^{(j)} = x_n^{(j)} \oplus v_c^{(j)}, \text{ за } j = 1, \ldots, s \tag{2.11}$$

За първата точка $x_0 = 0$. Сложността за конструиране на редица на Собол с N точки е $O(N(s + \log N))$. Ако кешираме стойностите на направляващите вектори m_i или v_i, бихме спестили повтарящи се изчисления.

[2] един полином е примитивен ако коефициентите му имат най-голям общ делител равен на 1

2.1.2 Редица на Холтън

Нека $b \geq 2$ е цяло число, тогава за всяко цяло число $k \geq 0$ има единствено представяне при основа b

$$k = \sum_{j=0}^{\infty} k_j b^j \tag{2.12}$$

Където $k_j \in \mathbb{Z}_b$ за $j \geq 0$ и $k_j = 0$ за всички достатъчно големи j.

Дефиниция 3 *За всяко $b \geq 2$ дефинираме обръщащата функция Φ_b при степен b като*

$$\Phi_b(k) = \sum_{j=0}^{\infty} k_j b^{-j-1}, \quad k \geq 0 \tag{2.13}$$

Където k представено като в 2.12.

Редицата $x_k = \Phi_b(k)$ се нарича редица на ван дер Корпут(van der Corput) при основа b. През 1937 ван дер Корпут публикува тази редица при основа $b = 2$, което е първата построена квазислучайна редица [20].

Холтън [9] обобщава редицата за размерност s. Нека $b_1, b_2, \ldots, b_s$ са взаимно прости числа. Редица на Холтън при основи $b_1, b_2, \ldots, b_s$ е s-мерната редица $x_0, x_1, \ldots$ дефинирана чрез

$$x_k = (\Phi_{b_1}(k), \ldots, \Phi_{b_s}(k)) \in [0,1)^s \text{ за всяко } k \geq 0. \tag{2.14}$$

Следва векторизиран алгоритъм (на Python) със сложност $O(sN \log N)$ за конструиране на редицата на Холтън.

Алгоритъм 3: редица на Холтън на Python

```
import numpy as np
def Phi(n, b):
    res = np.zeros(np.shape(b), np.float64)
    k = n*np.ones(np.shape(b), np.int32)
    bp = np.ones(np.shape(b), np.float64)
    while np.any(k>0):
        bp = bp/b
        a = k % b
        res += a*bp
        k = np.floor(k/b)
    return res
```

Редиците на Холтън могат да бъдат разширени по s и N. Само че те работят най-добре за малки прости числа b_j като основи, както може да се види на фигура 2.4.

$$
\begin{aligned}
x_0 &= (0.0000, 0.0000, 0.0000) \\
x_1 &= (0.5000, 0.3333, 0.2000) \\
x_2 &= (0.2500, 0.6667, 0.4000) \\
x_3 &= (0.7500, 0.1111, 0.6000) \\
x_4 &= (0.1250, 0.4444, 0.8000) \\
x_5 &= (0.6250, 0.7778, 0.0400) \\
x_6 &= (0.3750, 0.2222, 0.2400) \\
x_7 &= (0.8750, 0.5556, 0.4400) \\
x_8 &= (0.0625, 0.8889, 0.6400) \\
x_9 &= (0.5625, 0.0370, 0.8400)
\end{aligned}
$$

Таблица 2.1: Първите 10 точки от 3-измерната редица на Холтън $x_k = (\Phi_2(k), \Phi_3(k), \Phi_5(k))$ при основи $b = (2, 3, 5)$

2.1.3 Редица на Фор

Лошото разпределение при големи прости числа за основи на редицата на Холтън може да бъде преодоляно до известна степен, ако се ползва пермутация на Фор [6]. Нека $b \geq 2$ е цяло число и k има представяне при основа b

$$k = \sum_{j=0}^{w} k_j b^j$$

Нека дефинираме функцията

$$\Phi_b(\mathbf{k}) = \frac{k_0}{b} + \frac{k_1}{b^2} + \cdots + \frac{k_w}{b^w}$$

Където $\mathbf{k} = (k_0, k_1, \ldots, k_w)^T$. k-тият елемент на редицата на Фор се дефинира като

$$x_k = (\Phi_b(P^0\mathbf{k}), \Phi_b(P^1\mathbf{k}), \ldots, \Phi_b(P^{s-1}\mathbf{k}))$$

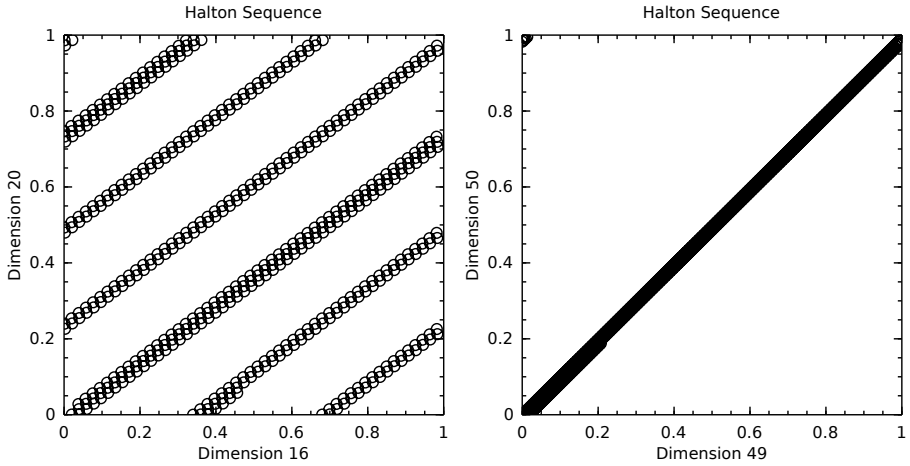
Където b е просто число по-голямо или равно на s; P^l е матрицата на Паскал повдигната на l-степен и взет модул b поелементно: $P(i,j) = \binom{i-1}{j-1} \mod b$ и $P^l(i,j) = l^{j-i}\binom{i-1}{j-1} \mod b$. Времето за изчисление на една точка на Фор е $O(s(\log N)^2)$.

2.1.4 (t, m, s)-мрежи и (t, s)-редици

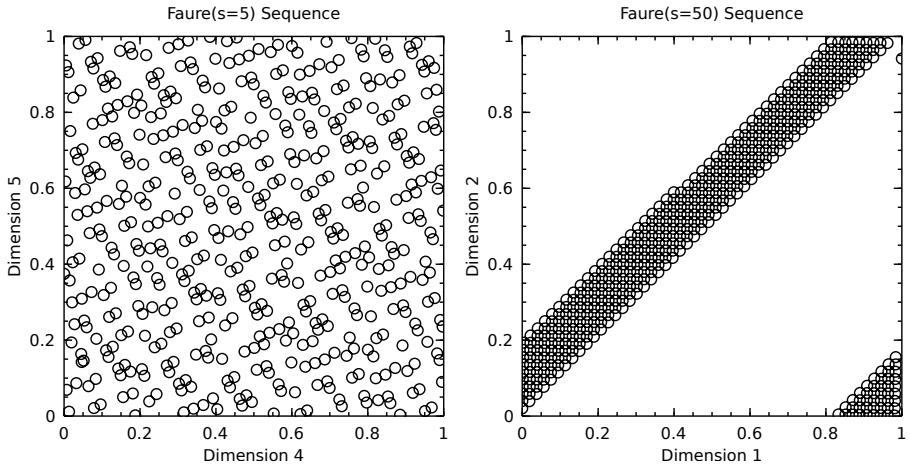
Цифровите мрежи предоставят по-задоволително обобщение на обръщащите функции за размерности $s \geq 2$. Нека дефинираме s-измерен елементарен интервал при основа b

$$\prod_{j=1}^{s} \left[\frac{l_j}{b^{k_j}}, \frac{l_j + 1}{b^{k_j}} \right) \tag{2.15}$$

За цели числа $k_j \geq 0$ и $0 \leq l_j < b^{k_j}$. Този елементарен интервал има обем b^{-K} където $K = k_1 + \cdots + k_s$. В идеалния случай искаме да имаме nb^{-K}



Фигура 2.4: Редицата на Холтън показва значителна корелация при големите размерности



Фигура 2.5: Ниско размерна (в ляво) и високо размерна (в дясно) редица на Фор. Високо размерната редица отново показва значителна корелация.

на брой точки във всеки такъв интервал. Числовите мрежи удолетворяват това, поне за достатъчно малки K.

Дефиниция 4 *Нека $b \geq 2$ целочислена основа и $m \geq t \geq 0$ е цяло число. (t, m, s)-мрежа при основа b наричаме крайна редица $x_1, \ldots, x_{b^m}$, за която всеки елементарен интервал при основа b и с обем b^{t-m} съдържа точно b^t на брой точки от редицата.*

Обобщението на безкрайни редици се получава чрез дефиницията за цифрова редица.

Дефиниция 5 (t, s)-редица при основа b *е безкрайна редица $(x_i)_{i=1}^{\infty}$ такава, че за всички цели числа $r \geq 0$ и $m \geq t$, точките $x_{rb^m+1}, \ldots, x_{(r+1)b^m}$ образуват (t, m, s)-мрежа при основа b.*

Следователно (t, s)-редиците могат да бъдат представени като безкрайна последователност от (t, m, s)-мрежи за всички $m \geq t$.

За $m \geq t$ и $1 \leq \lambda < b$, първите λb^m на брой точки от (t, s)-редица са балансирани по отношение на всички елементарни интервали при основа b и с обем b^{t-m} или по-голям.

Ако редицата (x_i) е (t, s)-редица, то тя има малък дискрепанс, както е показъл Нийдерийтер (Niederrieter) [15]. При някои условия и трите квазислучайни редици конструирани по-рано са (t, s)-редици:

- *редицата на Собол* е (t, s)-редица, ако примитивните полиноми 2.8 са различни и
$$t = \sum_{j=1}^{s} (\deg(P_j) - 1)$$
Следователно, колкото по-малка е степента на полиномите, толкова по-добре.
- *редицата на Фор* е $(0, s)$-редица ако основата е просто число $b \geq s$.

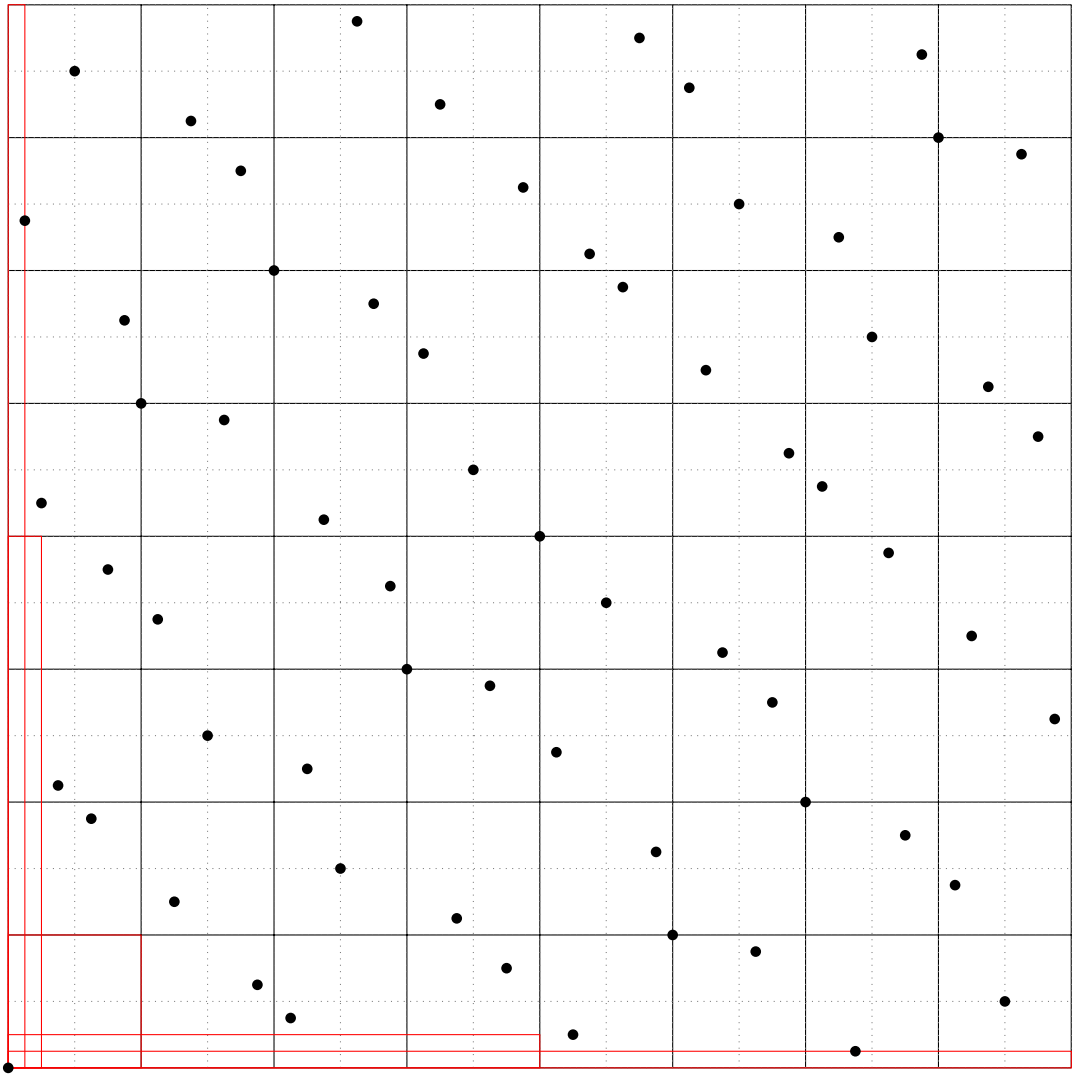
2.2 Разбъркани квазислучайни числа

Както вече видяхме, високоразмерните квазислучайни редици имат силно корелирани точки върху нискоразмерни подпространства (фиг. 2.5). За да преодолеем този ефект, взимаме квазислучайната редица (a_i) и трансформираме точките ѝ в случайни точки (x_i) така, че x_i да запазят някои от свойствата на първоначалната квазислучайна редица и очакването на $\hat{I}$ да е равно на I. С рандомизираните квази-Монте Карло методи можем да проведем R независими симулации получавайки оценки $\hat{I}_1, \ldots, \hat{I}_R$. Комбинираната оценка $\hat{I} = R^{-1} \sum_{r=1}^{R} \hat{I}_r$ има математическо очакване I. Неизметена оценка за средната квадратична грешка (mean-squared error (MSE)) за $\hat{I}$ е

$$MSE(\hat{I}) = \frac{1}{R(R-1)} \sum_{r=1}^{R} (\hat{I}_r - \hat{I})^2$$

За рандомизирането на квазислучайниредици има два основни подхода. Cranley и Patterson [5] предлагат ротация със случайно число по модул 1:

$$z_k = x_k + r \mod 1$$



Фигура 2.6: Първите 64 точки от редицата на Собол. Те образуват $(2, 2)$-мрежа. Червените правоъгълници са елементарни интервали при основа 2, закрепени за началото на координатната система. Когато транслираме червените правоъгълници получаваме всички елементарни интервали споментаи в дефиницията.

Където x_k е квазислучайно число в $[0,1)^s$ и r е s-мерна равномерно разпределана случайна величина. Различните r дават различни реализации (z_k) на първоначалната редица (x_k). Tuffin [18] предложил тези ротации да бъдат приложени към цифрови мрежи. Те загубват свойството си да бъдат мрежи, но все пак изглеждат равномерни.

Вторият подход е базиран на пермутации.

Схема на Оуен за разбъркване

Оуен [16] предлага схема за рандомизиране на произволни квазислучайни редици и точкови множества. Разглеждаме редица (x_n), $x_n = (x_n^{(i)})_{i=1}^s$, като i-тата компонента на n-тия елемент има представяне при основа b $x_n^{(i)} = \sum_{j=1}^{\infty} x_{n,j}^{(i)} b^{-j}$ А за рандомизираната точка полагаме $z_n^{(i)} = \sum_{j=1}^{\infty} z_{n,j}^{(i)} b^{-j}$, като $z_{n,j}^{(i)}$ са случайни пермутации на $x_{n,j}^{(i)}$.

$$
\begin{aligned}
z_{n,1}^{(i)} &= \pi_i(x_{n,1}^{(i)}) \\
z_{n,2}^{(i)} &= \pi_{i,x_{n,1}^{(i)}}(x_{n,2}^{(i)}) \\
z_{n,3}^{(i)} &= \pi_{i,x_{n,1}^{(i)},x_{n,2}^{(i)}}(x_{n,3}^{(i)}) \\
&\dots \\
z_{n,k}^{(i)} &= \pi_{i,x_{n,1}^{(i)},x_{n,2}^{(i)},\dots,x_{n,k-1}^{(i)}}(x_{n,k}^{(i)})
\end{aligned}
$$

Всяка от пермутациите π е равномерно разпределена измежду $b!$ пермутации на $\{0,1,\dots,b-1\}$ и пермутациите са взаимно независими. Този метод е труден за практично изпълнение, за това се ползват опростени варианти.

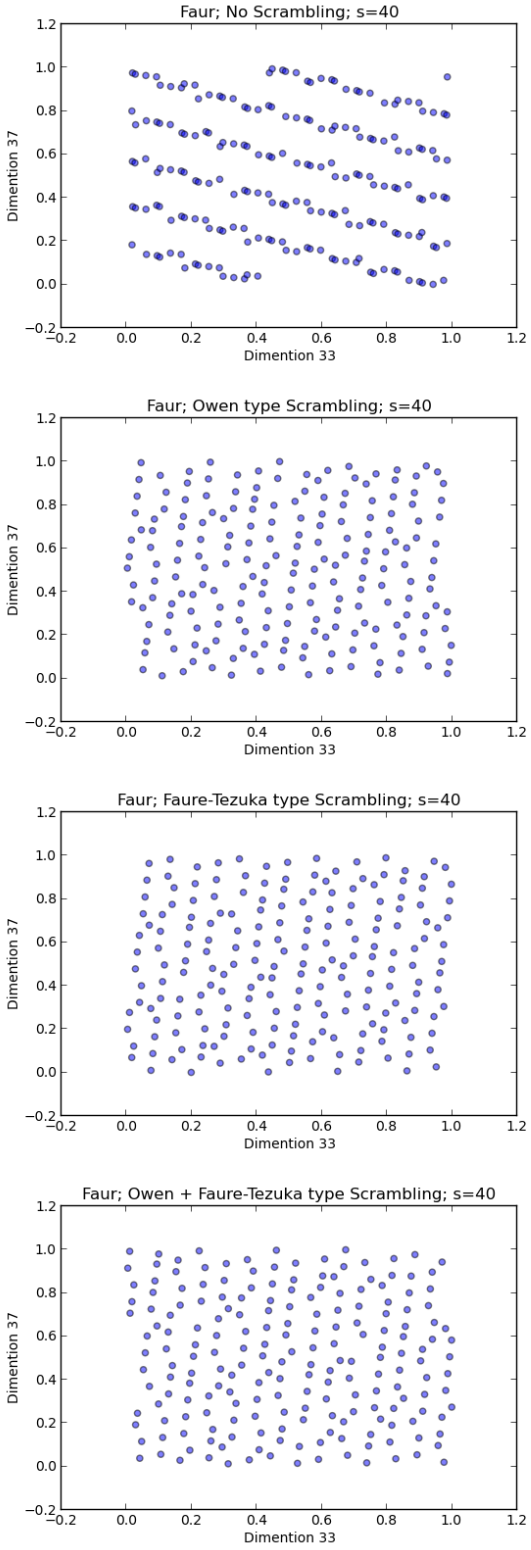
Обобщена редица на Фор

Обобщената редица на Фор е предложена от Тезука, като за размерност j генераторната матрица има вида $C^{(j)} = A^{(j)} P^{j-1}$, където $A^{(j)}$ за $j = 1,\dots,s$ са произволни неособени долно триъгълни матрици.

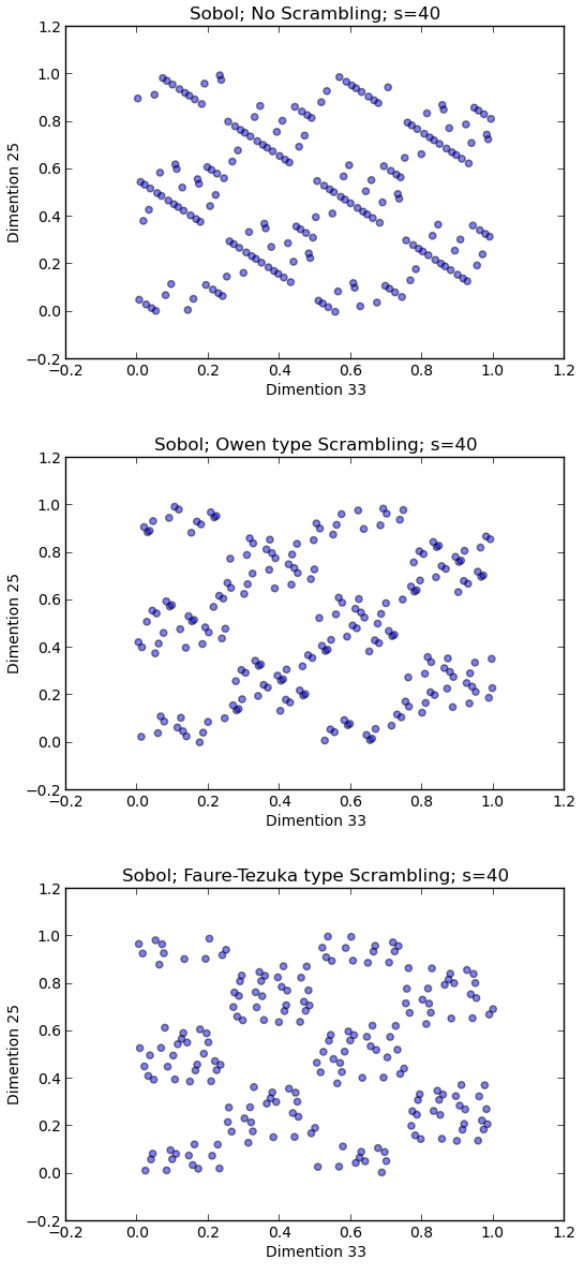
За числените експерименти ще използваме разбъркване на Оуен и на Тезука, както и хибриден вариант между двата метода. На фигура 2.7 се вижда ефектът от разбъркването за преодоляване на корелацията при голяма размерност на редицата.

Разбъркване на редицата на Собол

За редицата на Собол може да се приложат гореописаните методи за разбъркване. Ефектът от разбъркването може да бъде видян на фигура 2.8.



Фигура 2.7: Разбъркване на редицата на Фор



Фигура 2.8: Разбъркване на редицата на Собол

Глава 3

Степенен метод с Монте Карло итерации

В тази секция ще представя Монте Карло алгоритъм базиран на степенния метод (power method), описани в секция 1.1.2 на страница 5.

Както е обелязано в [12] Монте Карло алгоритмите в линейната алгебра са подходящи за големи разредени матрици.

Поради естеството на алгоритъма най-удачно е матриците да се съхраняват в редови формат, наричан още *a,ja,ia* схема[1] (Duff, 1986).

За да приложим Монте Карло алгоритъм ще третираме оригиналната матрица $A \in \mathbb{R}^{n\times n}$, след подходяща нормализация, като матрица на Марковска верига. Нека констуираме следната матрица

$$P(i,j) = \mathbb{P}\left(X_t = j | X_{t-1} = i\right) = \frac{|A(i,j)|}{\sum_{k=1}^{n} |A(i,k)|} \tag{3.1}$$

Тук X_t е дискретна случайна величина за $t = 0, 1, \ldots$. За стартовия вектор използваме разпределение

$$p(i) = \mathbb{P}\left(X_0 = i\right) = \frac{|h(i)|}{\sum_{j=1}^{n} |h(j)|} \tag{3.2}$$

Нека сега дефинираме случайната величина Y_t използвайки следната рекурсивна формула:

$$Y_0 = \frac{h(X_0)}{p(X_0)},\ Y_t = Y_{t-1}\frac{A(X_{t-1}, X_t)}{P(X_{t-1}, X_t)},\ t = 1, 2, \ldots, m \tag{3.3}$$

Може да се покаже, че

$$\langle h, A^m f\rangle = \mathbb{E}\left(Y_m f(X_m)\right) \tag{3.4}$$

Действително

$$Y_m = \frac{h(X_0)A(X_0,X_1)A(X_1,X_2)\ldots A(X_{m-1},X_m)}{p(X_0)P(X_0,X_1)P(X_1,X_2)\ldots P(X_{m-1},X_m)}$$

[1] това включва скаларен масив $a(1:nz)$ съдържащ ненулевите елементи на матрицата, съхранявани по редове, целочислен масив $ja(1:nz)$ съдържащ колоната на съответния елемент в матрицата и целочислен масив $ia(1:n+1)$, i-тия елемент, на който сочи към началото на масивите и ja на последователните редове.

Използвайки условна вероятност на $X_m = i_m, X_{m-1} = i_{m-1}, \ldots, X_0 = i_0$ и използвайки формулата за повторно очакване получаваме

$$\begin{aligned}
&\mathbb{E}\left(Y_m f(X_m)\right) \\
&= \sum_{i_0=1}^{n} \cdots \sum_{i_m=1}^{n} \frac{h(i_0)A(i_0,i_1)\ldots A(i_{m-1},i_m)}{p(i_0)P(i_0,i_1)\ldots P(i_{m-1},i_m)} f(i_m)p(i_0)P(i_0,i_1)\ldots P(i_{m-1},i_m) \\
&= \sum_{i_0=1}^{n} \cdots \sum_{i_m=1}^{n} h(i_0)A(i_0,i_1)\ldots A(i_{m-1},i_m)f(i_m) \\
&= \langle h, A^m f\rangle
\end{aligned}$$

За да формулираме метода на обратната итерация (inverse iteration) с Монте Карло стъпки използвайки (3.4) имаме

$$\langle h, (I-\mu A)^{-m} f\rangle = \mathbb{E}\left(\sum_{i=0}^{\infty} \binom{i}{m+i-1} \mu^i Y_i f(X_i)\right) \tag{3.5}$$

Тъй като $|\mu\lambda| < 1$ можем да развием $(I-\mu A)^{-m} = \sum_{i=0}^{\infty} \binom{i}{m+i-1} \mu^i A^i$.

Ще разгледаме следните Монте Карло версии на степенния метод, които се базират на (1.4) и (3.4)

1. обикновения степенен метод (алгоритъм 1) с Монте Карло итерации за намиране на най-голямата собствена стойност

$$\lambda_{\max} \approx \frac{\mathbb{E}\left(Y_i f(X_i)\right)}{\mathbb{E}\left(Y_{i-1} f(X_{i-1})\right)}$$

2. обратния степенен метод с Монте Карло итерации за намиране на най-малката собствена стойност

$$\lambda_{\min} \approx \frac{\mathbb{E}\left(\sum_{i=0}^{L} \binom{i}{m+i-1} Y_{i+1} f(X_{i+1})\right)}{\mathbb{E}\left(\sum_{i=0}^{L} \binom{i}{m+i-1} Y_i f(X_i)\right)}$$

3. обратна итерация с изместване за намиране на собствена стойност λ най-близо до изместващия параметър μ

$$\lambda \approx \frac{\mathbb{E}\left(\sum_{i=0}^{L} \binom{i}{m+i-1} \mu^i Y_{i+1} f(X_{i+1})\right)}{\mathbb{E}\left(\sum_{i=0}^{L} \binom{i}{m+i-1} \mu^i Y_i f(X_i)\right)}$$

Като сме отрязали безкрайния ред (3.5) до L-тия член.

3.1 Степенният метод и квази-Монте Карло методите

За да свържем теорията представена в глава 2 и приближенията представени в тази глава, ще покажем, че изчисляването на $\langle h, A^i f\rangle$ е еквивалентно

на намирането на $(i+1)$-размерен интеграл, както е описано от Караиванова в [12]. Така можем да проведем анализ на грешката, използвайки граници за числено интегриране като неравенството на Коксма-Хлавка 2.4. Действително, нека дефинираме $G_i = \left[\frac{i-1}{n}, \frac{i}{n}\right)$ за $i = 1, \ldots, n$, $f(x) = f(i)$ и $h(x) = h(i)$ за $x \in G_i$, $A(x,y) = A(i,j)$ за $x \in G_i$ и $y \in G_j$. Следователно имаме

$$\begin{aligned}\int_0^1 \int_0^1 h(x)A(x,y)f(y)dxdy &= \sum_{i=1}^{n}\sum_{j=1}^{n}\int_{G_i}\int_{G_j} h(i)A(i,j)f(j)dxdy \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} h(i)A(i,j)f(j)\int_{G_i}\int_{G_j} dxdy \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} h(i)A(i,j)f(j)\ \frac{1}{n^2} \\ &= \frac{1}{n^2}\langle h, Af\rangle\end{aligned}$$

Нека (x_k, y_k) е редица от N точки в $[0,1)^2$, тогава

$$\begin{aligned}\frac{1}{N}\sum_{k=1}^{N} h(x_k)A(x_k,y_k)f(y_k) &= \frac{1}{N}\sum_{G_i\times G_j}\left(\sum_{(x_k,y_k)\in G_i\times G_j} h(x_k)A(x_k,y_k)f(y_k)\right) \\ &= \frac{1}{N}\sum_{i=1}^{n}\sum_{j=1}^{n} h(i)A(i,j)f(j)\ \#\{(x_k,y_k)\in G_i\times G_j\}\end{aligned}$$

Следователно разликата между скаларното произведение и неговата оценка става

$$\begin{aligned}\left|\frac{1}{n^2}\langle h, Af\rangle - \frac{1}{N}\sum_{k=1}^{N} h(x_k)A(x_k,y_k)f(y_k)\right| &\le \sum_{i=1}^{n}\sum_{j=1}^{n}|h(i)A(i,j)f(j)|D_N^\star \\ &= |h|^T|A||f|D_N^\star,\end{aligned}$$

където $|.|$ е приложен покомпонентно за матрицата A и векторите h и f.

Аналогично, взимайки $\langle h, A^l f\rangle$ може да използваме $l+1$ размерна редица

$$\left|\frac{1}{n^{l+1}}\langle h, A^l f\rangle - \frac{1}{N}\sum_{k=1}^{N} h(x_k)A(x_k,y_k)\ldots A(z_k,w_k)f(w_k)\right| \le |h|^T|A|^l|f|D_N^\star$$

Нека A е разредена матрица с d_i ненулеви елемента на ред i. Следното изображение отговаря на метода на съществената извадка (importance sampling).

$$G_i = \left[\frac{\sum_{j=1}^{i-1}|A(i,j)|}{\sum_{j=1}^{n}|A(i,j)|}, \frac{\sum_{j=1}^{i}|A(i,j)|}{\sum_{j=1}^{n}|A(i,j)|}\right),\ i = 1, \ldots, n$$

В този случай след подобни изчисления може да докажем, че горна граница за грешката се дава чрез

$$\left|\frac{1}{n^{l+1}}\langle h, A^l f\rangle - \frac{1}{N}\sum_{k=1}^{N} h(x_k)A(x_k,y_k)\ldots A(z_k,w_k)f(w_k)\right| \le (d|A|)^l D_N^\star,$$

където d е средния брой ненулеви елементи на матрицата A за ред. Ако използваме $(l+1)$ размерна редица с малък дискрепанс и A е разредена $(d \ll n)$, то горната оценка е от ред $O((\log(N))^l/N)$.

3.1.1 Балансиране на грешката и сложност

Сходимостта на методите описани в тази глава зависят от два фактора:

1. систематична грешка идваща от степенния метод (1.6)

$$\text{за обикновения степенен метод: } O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^m\right)$$
$$\text{за обратния степенен метод: } O\left(\left|\frac{\lambda_n}{\lambda_{n-1}}\right|^m\right)$$
$$\text{за } \mu\text{-изместената обратна итерация: } O\left(\left|\frac{\lambda_1-\mu}{\lambda_2-\mu}\right|^m\right)$$

2. стохастична грешка описана в миналата част:

$$\text{за псевдослучайна редица: } O(N^{-1/2})$$
$$\text{за редица с малък дискрепанс: } O((\log(N))^s/N)$$

Сумирането на тези два члена дават грешката на метода. Постигането на твърде много точност в едното събираемо не гарантира по-добро поведение, ако другото събираемо дава голяма грешка. За това е добре да се стремим да балансираме двете грешки да са от един порядък.

Глава 4

Числени експерименти

4.1 Използвани техологии и библиотеки

Python, NumPy, SciPy, mpi4py

Основната част от програмирането е извършено на езика Python. Използвани са широко разпространените библиотеки NumPy и SciPy ([11]). Използвал съм библиотеката за разредени матрици scipy.sparse. Там може да бъде намерен и порт на ARPACK за Python. ARPACK е имплементация на алгоритъма на Арнолди за намиране на собствени стойности и вектори написана на Fortran. NumPy и SciPy са стандрта за извършване на числени експерименти на Python.

При написването на скрипта използвах библиотеката mpi4py, която предоставя интерфейса на MPI за Python. Така се възползвах от тривиално паралелизиране на алгоритъма при провеждане на поредица от симулации.

Fortran, f2py

Генераторите на квазислучайни числа ми бяха предоставени от проф. Анета Караиванова, които са публикувани в Transactions on Mathematical Software, том 14, No. 1, стр.88. Кода е написан на Fortran 77. С цел да се улеснят числените експерименти използвах програмта *f2py*, която е част от SciPy. *f2py* компилира кода на Fortran до споделена библиотека, която може да бъде извиквана от Python. Понеже библиотеката се компилира до машинен код, скоростта на функциите в библиотеката е същата като на програма написана изцяло на компилиран език.

IPython

В последните години интерактивните възможности на IPython [17] са напреднали значително. Особено внимание трябва да се отдели на IPython Notebook, който позволява да се визуализират резултатите от експериментите интерактивно в интернет браузър. Резултатът е среда подобна на Mathematica, но с отворен код и използваща широко разпространен език като Python. Много от графиките в тази работа са генерирани по този начин.

4.2 Конструиране на тестовите матрици

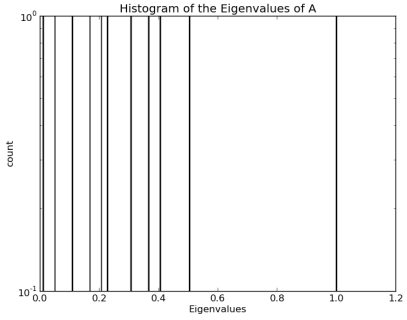
Тъй като грешката на Монте Карло метода за намиране на собствени стойности идва от две места, матриците, за които той е подходящ, трябва да отговарят на следните условия. Първо, трябва отношението между най-големия и следващия по абсолютна стойност собствени ветори да е малко ($\ll 1$). Второ, за да имаме малка стохастична грешка, трябва матрицата да е добре балансирана по редове. В екстремалния случай когато всички редове имат една и съща стойност стохастичната грешка става 0.

Уголемяване на матрици

Нека A е $(na \times na)$ матрица с известни собствени стойност $\lambda_1, \ldots, \lambda_{na}$. Нека Q е $(nq \times nq)$ матрица с известни собствени стойности $\mu_1, \ldots, \mu_{nq}$.

Ако дефинираме произведението на Кронекер между A и Q: $X = A \otimes Q$, то собствените стойности на X са $\nu_{i,j} = \lambda_i \mu_j$ за $i = 1, \ldots, na$ и $j = 1, \ldots, nq$. X има блокова структура на A, а всеки блок има структурата на X.

За A избирам 11×11 матрица със собствени стойности разпределени в $(0, 1]$, като е показано на фигура 4.1.



Фигура 4.1: Разпределение на собствените стойности на A

За Q избирам $Q_{ij} = \begin{cases} 1/(nq - j) & \text{ако } j >= i \\ 0 & \text{другаде} \end{cases}$,

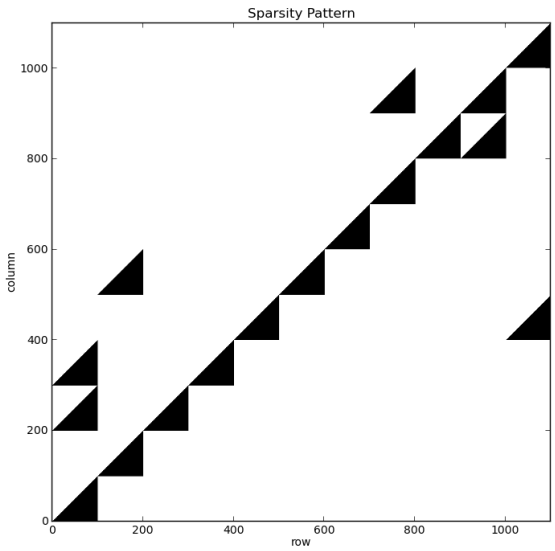
Така матрицата X с размери (1100×1100) има структура показана на фигура 4.2.

Разпределението на собствените стйности на X може да бъде видяно на фигура 4.3. Отношението, което определя систематичната грешка е приблизително равно на 0.5.

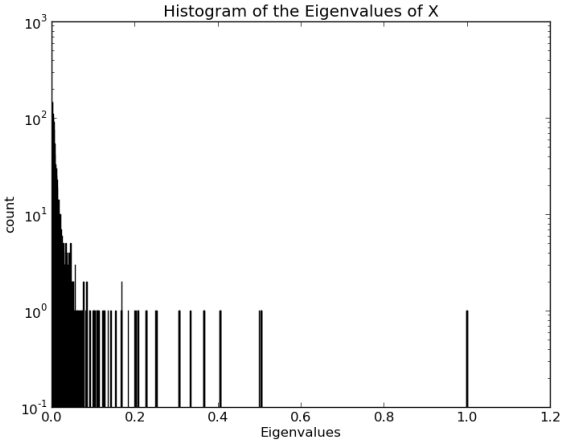
4.3 Резултати от експериментите

Експериментите са проведени на 64-ядрен компютър, поради което паралелизирането ни дава по-малки времена за изпълнение на програмта.

На фигура 4.4 са представени резултатите за малката матица A. На фигура 4.5 са представени резултатите за малката матица X. Поради малката експонента систематичната грешка не позволява да се намерят с голяма



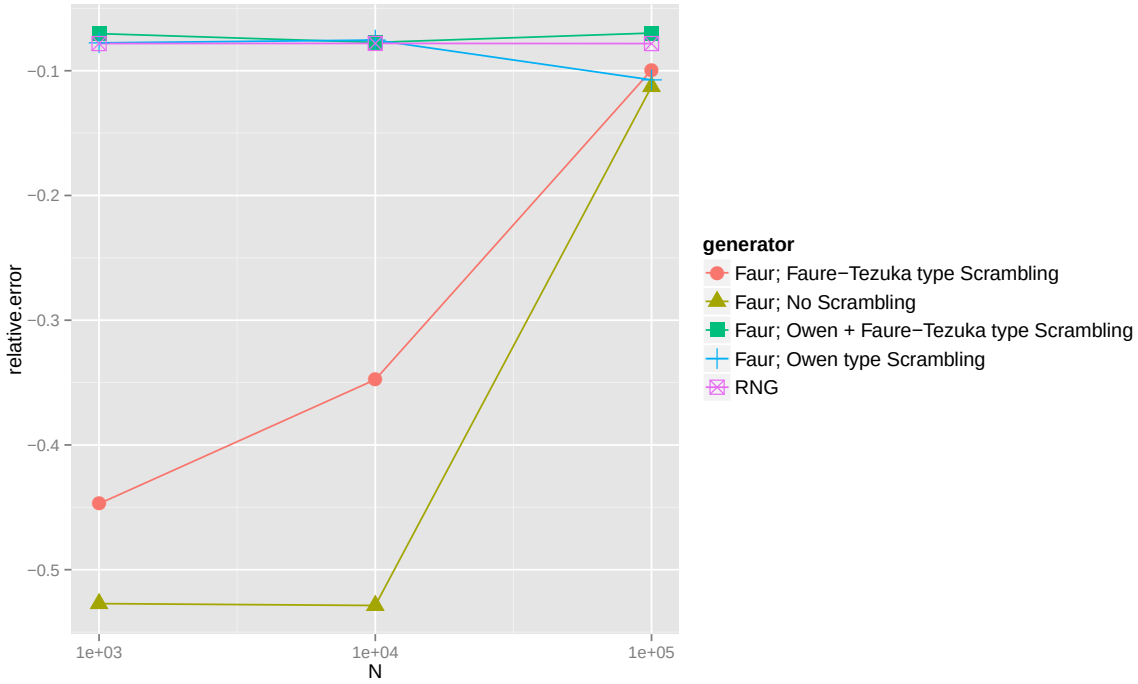
Фигура 4.2: Структура на голямата матрица X



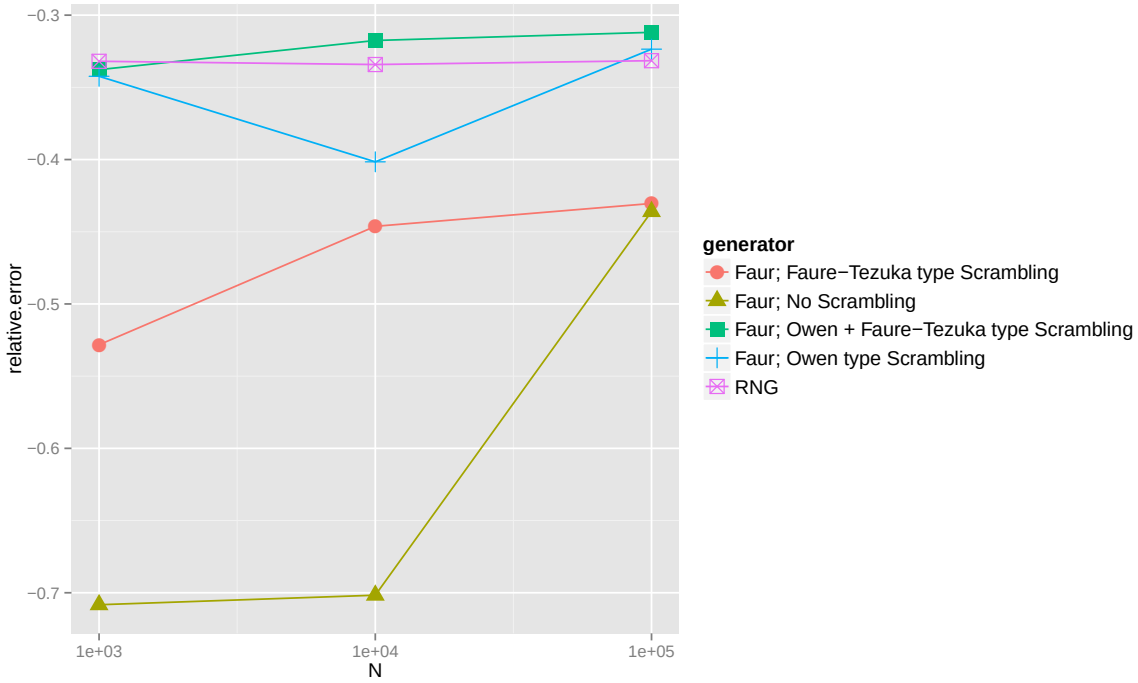
Фигура 4.3: Хистограма на собствените стойности на голямата матрица X

точност собствената стойност. Все пак при разбърканите редици тази граница се достига по-точно и за по-малко итерации. В случая експериментите с псевдо случайни числа дават добри резултати, но в известна степен причина за това е и особеност на конкретната извадка. Квазислучайните редици в много по-малка степен зависят от конкретната извадка.

Най-добри резултати дава редицата на Фор с разбъркване на Оуен и Фор-Тезука и за голямата и за малката матрица.



Фигура 4.4: Резултати за матрицата A



Фигура 4.5: Резултати за матрицата X

Да отбележим, че времената за изпълнение на алгоритъма зависят от размерността на матрицата като корен квадратен от броя редове $\sim n^{-1/2}$, както може да се види в таблица 4.1. Това е значително подобреие над стандартния алгоритъм 1 или 2 разгледани в глава 1. При тях сложността е съответно $O(Nn^2)$ и $O(n^3 + Nn^2)$. Ако матриците имат един и същи среден брой ненулев брой елементи на ред, то допълнителната нужната памет е пропорционална на броя редове n.

N	generator	m	matrix	time
1000	Faur; Faure-Tezuka type Scrambling	5	A	0.60600
			X	4.94200
	Faur; No Scrambling	5	A	0.61000
			X	6.14800
	Faur; Owen + Faure-Tezuka type Scrambling	5	A	0.55600
			X	5.22800
	Faur; Owen type Scrambling	5	A	0.59600
			X	5.19200
	RNG	5	A	0.58400
			X	4.92800
10000	Faur; Faure-Tezuka type Scrambling	5	A	5.99800
			X	41.13400
	Faur; No Scrambling	5	A	6.10800
			X	61.72000
	Faur; Owen + Faure-Tezuka type Scrambling	5	A	5.66200
			X	52.59800
	Faur; Owen type Scrambling	5	A	5.95000
			X	50.16400
	RNG	5	A	5.87800
			X	48.63800
100000	Faur; Faure-Tezuka type Scrambling	5	A	59.62800
			X	562.52546
	Faur; No Scrambling	5	A	59.49800
			X	519.63746
	Faur; Owen + Faure-Tezuka type Scrambling	5	A	58.41000
			X	525.80000
	Faur; Owen type Scrambling	5	A	60.13200
			X	584.88546
	RNG	5	A	58.52200
			X	502.95746

Таблица 4.1: Време за провеждане на числения експеримент с матрица A с размери 11×11 и с матрица X с размери 1100×1100

Глава 5

Заключение

В настоящата дипломна работа е изследван рандомизиран квази-Монте Карло алгоритъм за намиране на доминантна собствена стойност. При направените експерименти редицата на Фор с разбъркване на Оуен и Фор-Тезука дава най-добри резултати, дори при относително малък брой итерации.

Допълнителното време за изчисления при фиксиран брой итерации е пропоционално на корен квадратен от на броя на редове, а допълнителната памет е пропорционална на средния брой ненулеви елементи на ред и броя редове. Това е едно от основните предимства на алгоритъма.

Приложение А

Степенен метод с Монте Карло итерации на Python

Основната функция е eigmcPowerMethod(A, m, N, numseq). numseq може да бъде произволна числова редица.

```
import numpy as np
import scipy.sparse as sp

def eigmcPowerMethod(A, m, N, numseq):
    (n, nn) = A.shape
    if(n != nn):
        raise Exception, "non square matrix"
    h = np.repeat(1.0, n)
    p0 = h / sum(h)
    P = normalize(A)
    sumW = np.float96(0.0)
    sumW_1 = np.float96(0.0)
    for i in xrange(N):
        k = getWeighted(range(n), p0, numseq)
        W = h[k] / p0[k]
        W_1 = W
        for j in xrange(m):
            k_1 = k
            vals = P.indices[P.indptr[k_1]:P.indptr[k_1+1]]
            weights = P.data[P.indptr[k_1]:P.indptr[k_1+1]]
            k = getWeighted(vals, weights, numseq)
            W_1 = W
            W *= A[k_1, k] / P[k_1, k]
        sumW += W
        sumW_1 += W_1
    return sumW / sumW_1

def normalize(A):
    (data, indices, indptr) = (A.data.copy(), A.indices, A.indptr)
    for i in xrange(len(indptr) - 1):
        row = np.abs(data[indptr[i]:indptr[i + 1]])
        rsum = np.sum(row)
        if(rsum > 0.0):
```

```
            data[indptr[i]:indptr[i + 1]] = row / rsum
        else:
            raise Exception("zero line")
    return sp.csr_matrix((data, indices, indptr), shape=A.shape)

def getWeighted(vals, weights, rand):
    if (np.abs(np.sum(weights) - 1) > 1e-6):
        raise Exception("weigths should sum up to 1!")
    r = rand()
    s = 0.0
    i = 0
    while(s <= r):
        while(weights[i] <= 0): i += 1
        s += weights[i]
        i += 1
    return vals[i - 1]
```

Библиография

[1] I. A. Antonov and V. M. Saleev. An economic method of computing LPτ-sequences. *USSR Computational Mathematics and Mathematical Physics*, 19:252–256, 1979.

[2] J. Beck and W. W. L. Chen. *Irregularities of Distribution.* Cambridge University Press, New York, 1987.

[3] R. E. Calfisch. Monte carlo and quasi-Monte Carlo methods. *Acta Numerica*, (7):1–49, 1998.

[4] K.-L. Chung. An estimate concerning the Kolmogoroff limit distribution. *Transaction of the American Mathematical Society*, (67):36–50, 1949.

[5] R. Cranley and T. N. L. Patterson. Randomization of number theoretic methods for multiple integration. *SIAM Journal of Numerical Analysis*, (13):904–914, 1976.

[6] Henri Faure. Good permutations for extreme discrepancy. *Journal of Number Theory*, (42):47–56, 1992.

[7] G. E. Forsythe and R. A. Leibler. Matrix inversion by a monte carlo method. *Mathematical Tables and Other Aids to Computation*, (4):127–129, 1950.

[8] Charles van Loan Gene H. Golub. *Matrix Computations.* The Johns Hopkins University Press, 3 edition, 1996.

[9] J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, (2):84–90, 1960.

[10] E. Hlawka. Funktionen von beschraenkter Variation in der Theorie der Gleichvertreilung. *Annali di Matematica Pura ed Applicata*, (54):325–333, 1961.

[11] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[12] Aneta Karaivanova. Quasi-Monte Carlo methods for some linear algebra problems. convergence and complexity. *Serdica J. Computing, Bulgarian Academy of Sciences*, (4):57–72, 2010.

[13] N. M. Korobov. The approximate computation of multiple integrals. *Dokl. Akad. Nauk SSSR*, (124), 1959.

[14] L. Kuipers and H. Niederreiter. *Uniform distribution of sequences*. John Wiley and Son, New York, 1976.

[15] Harald Niederreiter. Random number generation and quasi-Monte Carlo methods. *S.I.A.M.*, 1992.

[16] A. Owen. Variance with alternative scramblings of digital nets. *ACM Transactions on Modeling and Computer Simulation*, 4(14):363–378, 2003.

[17] Fernando Pérez and Brian E. Granger. IPython: a System for Interactive Scientific Computing. *Comput. Sci. Eng.*, 9(3):21–29, May 2007.

[18] Bruno Tuffin. On the use of low discrepancy sequences in Monte Carlo methods. Technical Report 1060, I.R.I.S.A., Rennes, France, 1996.

[19] H. Weyl. Uber die gleichverteilung von Zahlen mod. eins. *Mathematische Annalen*, 77:313–352, 1916.

[20] Анета Караиванова. *Стохастични числени методи и симуляции*. Деметра ЕООД, София, 2012.