O‘ZBEKISTON RESPUBLIKASI

OLIY VA O‘RTA MAXSUS TA’LIM VAZIRLIGI

MIRZO ULUG‘BEK NOMIDAGI O’ZBEKISTON MILLIY UNIVERSITETININIG

JIZZAX FILIALI

AMALIY MATEMATIKA FAKULTETI

«KOMPYUTER ILMLARI VA DASTURLASHTIRISH» kafedrasi

“ALGORITMLAR VA BERILGANLAR STRUKTURASI” FANIDAN

MUSTAQIL ISH

Mavzu: Gauss usuli yordamida tenglamalar sistemasini yechish algoritmi. Muammolar & Yechimlar

Bajardi: Jalilov Shavkat .

Qabul qildi: Ulug’murodov Sh. B.

Jizzax – 2022

Reja:

Asosiy qism.

Gauss usuli yordamida tenglamalar sistemasini yechish algoritmi.

Gauss usuli birlik matritsalar uchun ishlamaydi chunki yechim nol bo'ladi.

Kiruvchi ma'lumotlar: N ta noma'lumlar uchun, kiruvchi kengaytirilgan matritsaning qiymati N x (N+1). Bitta qo`shimcha ustun Right Hand Side (RHS) uchun.

mat[N][N+1] = {{3.0, 2.0,-4.0, 3.0},
{2.0, 3.0, 3.0, 15.0},
{5.0, -3, 1.0, 14.0} };

Chiquvchi ma'lumotlar: Sistemaning yechimi:

3.000000

1.000000

2.000000

Izoh:

Berilgan matritsa quyidagi tenglamalarni ifodalaydi.

$3.0X_1 + 2.0X_2 - 4.0X_3 = 3.0$
$2.0X_1 + 3.0X_2 + 3.0X_3 = 15.0$
$5.0X_1 - 3.0X_2 + X_3 = 14.0$

Berilgan tenglamalar sistemasi uchun yagona yechim mavjud bo`lib bular:
$X_1 = 3.0$, $X_2 = 1.0$, $X_3 = 2.0$,

1	2	3	4
0	0	1	3
0	0	0	1

Row echelon form: Agar matritsada quyidagi shartlar mavjud bo'lsa r.e.f.d deyiladi.

1.Har bir satrda yetakchi koeffitsient deb ataladigan nolga teng bo'lmagan birinchi element 1 ga teng.

2.Har bir **y**etakchi koeffitsient oldingi qatorning o'ng tomonidagi ustunda joylashgan.

3.Barcha nolga ega satrlar kamida bitta nolga teng bo‘lmagan elementi bo‘lgan qatorlar ostida joylashgan.

1	2	0	0
0	0	1	0
0	0	0	1

Reduced row echelon form: Agar quyidagi shartlar mavjud bo'lsa - Matritsa r.r.e.f. deyiladi.

1.r.e.f. uchun barcha sharoitlar mavjud.

2.Har bir satrdagi yetakchi koeffitsient uning ustunidagi yagona nolga teng bo'lmagan elementdir.

Algoritm asosan matritsa satrlarida amallar ketma-ketligini bajarishdan iborat. Ushbu amallarni bajarayotganda yodda tutish kerak bo'lgan narsa shundaki, matritsani *row echelon* ko'rinishi bu matrissa diagonalidan past qismi nolga aylantirish. Jarayonlar quyidagicha:

1. Ikki qatorni almashtirish

2.Qatorni nolga teng bo‘lmagan skalerga ko‘paytirish

3.Bir qatorga ikkinchi qatorga karra qo‘shish

Jarayon:

Oldinga yo'q qilish: Diagonaldan pastki qism elementlarini nolga aylantirish. Undan foydalanib, hech qanday yechim yoki noyob yechim yoki cheksiz ko'p echimlar yo'qligini aniqlash mumkin.

Orqaga almashtirish: Nolga aylangan elementlardan tashqari yana elementlar yana elementlarni nolga aylantirish.

Algoritm:

1. Qisman pivotga aylantirish: eng kata qiymatga ega boʻlgan qator yoki ustun pivot holatiga oʻtkazish uchun qatorlarni almashtirish orqali k-burilishni topiladi. Bu algoritmga hisoblash barqarorligini beradi.

2. Pivot ostidagi har bir satr uchun k-chi yozuvni nolga aylantiruvchi f omilni hisoblanadi va qatordagi har bir element uchun k-qatordagi mos keladigan elementning f karrali qismini ayiriladi.

3. Har bir noma'lum uchun yuqoridagi amallarni takrorlang Qisman r.e.f matritsasi bilan qolishi mumkin

Psevdokod

```
1. Start
2. Input the Augmented Coefficients Matrix (A):
      For i = 1 to n
            For j = 1 to n+1
                  Read Ai,j
            Next j
      Next i
3. Apply Gauss Elimination on Matrix A:
      For i = 1 to n-1
            If Ai,i = 0
                  Print "Mathematical Error!"
                  Stop
            End If
            For j = i+1 to n
                  Ratio = Aj,i/Ai,i
                  For k = 1 to n+1
                        Aj,k = Aj,k - Ratio * Ai,k
                  Next k
            Next j
      Next i
4. Obtaining Solution by Back Substitution:
      Xn = An,n+1/An,n
      For i = n-1 to 1 (Step: -1)
            Xi = Ai,n+1
```

```
            For j = i+1 to n
                  Xi = Xi - Ai,j * Xj
            Next j
            Xi = Xi/Ai,i
      Next i
5. Display Solution:
      For i = 1 to n
            Print Xi
      Next i
6. Stop
```

Quyida yuqoridagi algoritmni amalga oshirish ko'rsatilgan.

```
using System;
class shavkat
{	public static int N = 3;
	static void gaussianElimination(double[,] mat)
	{
		int singular_flag = forwardElim(mat);
		if (singular_flag != -1)
		{
			Console.WriteLine("Birlik matritsa.");
			if (mat[singular_flag, N] != 0)
				Console.Write("Tenglamalar sitemasi emas.");
			else
				Console.Write("Cheksiz ko`p yechimga ega ");
			return;
```

```
using System;
class shavkat
{
	public static int N = 3;
	static void gaussianElimination(double[,] mat)
	{
```

```
        int singular_flag = forwardElim(mat);
        if (singular_flag != -1)
        {
            Console.WriteLine("Birlik matritsa.");
            if (mat[singular_flag, N] != 0)
                Console.Write("Tenglamalar sitemasi emas.");
            else
                Console.Write("Cheksiz ko`p yechimga ega ");
            return;
        }
        backSub(mat);
    }
    static void swap_row(double[,] mat, int i, int j)
    {
        for (int k = 0; k <= N; k++)
        {
            double temp = mat[i, k];
            mat[i, k] = mat[j, k];
            mat[j, k] = temp;
        }
    }

    static void print(double[,] mat)
    {
        for (int i = 0; i < N; i++, Console.WriteLine())
            for (int j = 0; j <= N; j++)
                Console.Write(mat[i, j]);
        Console.WriteLine();
    }
```

```
static int forwardElim(double[,] mat)
{
    for (int k = 0; k < N; k++)
    {
        int i_max = k;
        int v_max = (int)mat[i_max, k];

            for (int i = k + 1; i < N; i++)
        {
            if (Math.Abs(mat[i, k]) > v_max)
            {
                v_max = (int)mat[i, k];
                i_max = i;
            }
            if (mat[k, i_max] == 0)
                return k;
            if (i_max != k)
                swap_row(mat, k, i_max);
            for ( i = k + 1; i < N; i++)
            {
                double f = mat[i, k] / mat[k, k];

                for (int j = k + 1; j <= N; j++)
                    mat[i, j] -= mat[k, j] * f;
                mat[i, k] = 0;
            }
          }
    }
    return -1;
```

```
}
static void backSub(double[,] mat)
{
    double[] x = new double[N];
    for (int i = N - 1; i >= 0; i--)
    {
        x[i] = mat[i, N];
        for (int j = i + 1; j < N; j++)
        {
            x[i] -= mat[i, j] * x[j];
        }
        x[i] = x[i] / mat[i, i];
    }
    Console.WriteLine();
    Console.WriteLine("Tenglalar sistemasi uchun yechim:");
    for (int i = 0; i < N; i++)
    {
        Console.Write("{0:F6}", x[i]);
        Console.WriteLine();
    }
}
public static void Main(String[] args)
{
    double[,] mat =
    {       { 3.0, 2.0, -4.0, 3.0 },
        { 2.0, 3.0, 3.0, 15.0 },
        { 5.0, -3, 1.0, 14.0 }
    };
    gaussianElimination(mat);
```

```
    }
}
```

Time complexity: Har bir pivot uchun uning ostidagi har bir qator uchun uning o'ng tomonidagi qismini hisoblab o'tamiz, O(n)*(O(n)*O(n)) = O(n^3).

Gauss usuli orqali quyidagilar amalga oshiriladi.

1. Matritsaning rangi
2. Matritsa determinanti
3. Kvadrat matritsaga teskari matritsa topish

Muammolar va yechimlar.

Gauss metodi vaqtdan yutqazadi ya'ni ***time complexity**=$O(n^3)$* ga teng . Shu sababli bu turli xil muammolar keltirib chiqaradi. Quyida bu algoritm o`rniga ishlatsa bo`ladigan algoritmlar.

Strassen's submatritsiyasi

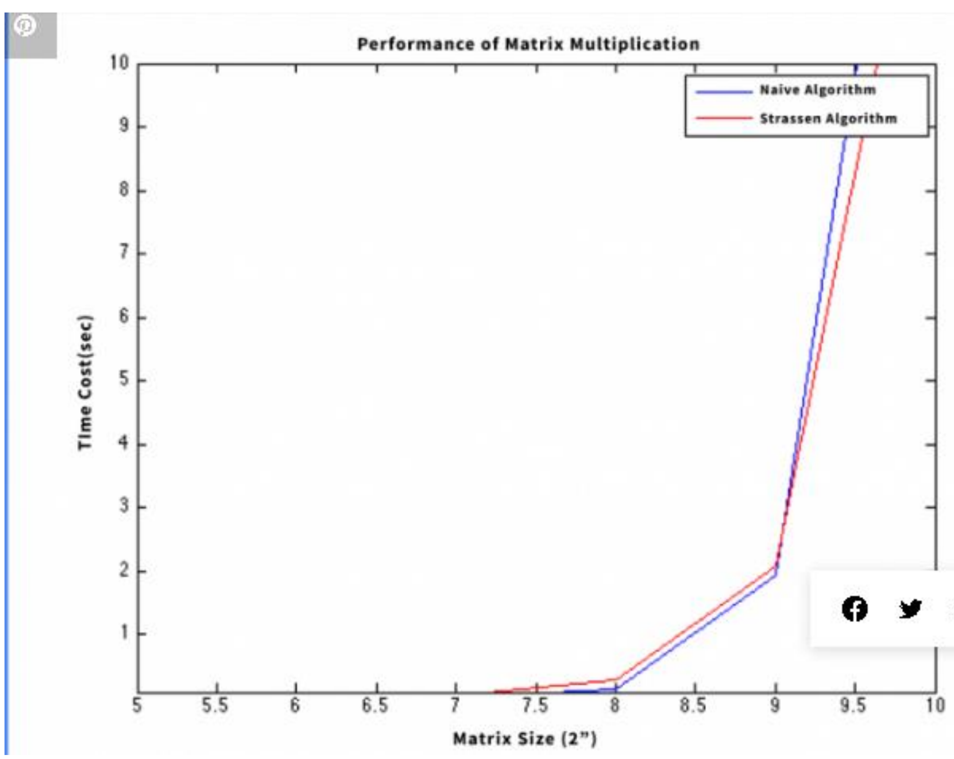
```
a * (f - h)
 (a + b) * h
 (c + d) * e
 d * (g - e)
 (a + d) * (e + h)
(b - d) * (g + h)
 (a - c) * (e + f)
matrix C = |p5+p4-p2+p6 p1+p2 |
           | p3+p4 p1+p5-p3-p7|
```

p5+p4-p2+p6 = (a+d)*(e+h) + d*(g-e) - (a+b)*h + (b-d)*(g+h) = (ae+de+ah+dh) + (dg-de) - (ah+bh) + (bg-dg+bh-dh) = ae+bg

p1+p2 = a*(f-h) + (a+b)*h = (af-ah) + (ah+bh) = af+bh

p3+p4 = (c+d)*e + d*(g-e) = (ce+de) + (dg-de) = ce+dg

p1+p5-p3-p7 = a*(f-h) + (a+d)*(e+h) - (c+d)*e - (a-c)*(e+f) = (af-ah) + (ae+de+ah+dh) -(ce+de) - (ae-ce+af-cf) = cf+dh



T(n) = 7T(n/2) + O(n^2) = **O(n^log(7))** runtime.

Taxminan time complexity= **O(n^2.8074)** bu esa O(n^3) yaxshiroq.

Strassen ko'paytirishning psevdokodi

1. Yuqoridagi diagrammada ko‘rsatilganidek, A va B matritsalarini N/2 x N/2 o‘lchamdagi 4 ta kichik matritsaga bo‘linadi.
2. 7 matritsaning ko‘paytmasini rekursiyasi hisoblanadi.
3. C submatritsalarini hisoblanadi.
4.Ushbu submatritsalarni yangi C matritsasiga birlashtiriladi.

Time complexity

1. Worst case time complexity: **Θ(n^2.8074)**
2. Best case time complexity: **Θ(1)**
3. Space complexity: **Θ(logn)**

```
import numpy as np


def strassen_algorithm(x, y):
    if x.size == 1 or y.size == 1:
        return x * y

    n = x.shape[0]

    if n % 2 == 1:
        x = np.pad(x, (0, 1), mode = 'constant')
        y = np.pad(y, (0, 1), mode = 'constant')

    m = int(np.ceil(n / 2))
    a = x[: m, : m]
    b = x[: m, m:]
    c = x[m:, : m]
    d = x[m:, m:]
    e = y[: m, : m]
    f = y[: m, m:]
    g = y[m:, : m]
    h = y[m:, m:]
    p1 = strassen_algorithm(a, f - h)
    p2 = strassen_algorithm(a + b, h)
    p3 = strassen_algorithm(c + d, e)
    p4 = strassen_algorithm(d, g - e)
    p5 = strassen_algorithm(a + d, e + h)
    p6 = strassen_algorithm(b - d, g + h)
    p7 = strassen_algorithm(a - c, e + f)
    result = np.zeros((2 * m, 2 * m), dtype = np.int32)
```

```
    result[: m, : m] = p5 + p4 - p2 + p6
    result[: m, m:] = p1 + p2
    result[m:, : m] = p3 + p4
    result[m:, m:] = p1 + p5 - p3 - p7

    return result[: n, : n]

if __name__ == "__main__":

    x = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
    y = np.array([[-1, 0, 0], [0, -1, 0], [0, 0, -1]])
    print('Matrix multiplication result: ')

  print(strassen_algorithm(x, y))
```

Naive algoritmi

Bu algoritm strassen algoritmidan ancha qulayroq. Yuqoridagi jadvalda buni yaqqol ko`rinishimiz mumkin.

Psevdokodi

Algorithm 1: The Naive Matrix Multiplication Algorithm

Data: $S[A][B]$, $P[G][H]$
Result: $Q[][]$
if $B == G$ **then**
    for $m = 0; m < A; m{+}{+}$ **do**
        for $r = 0; r < H; r{+}{+}$ **do**
            $Q[m][r] = 0;$
            for $k = 0; k < G; k{+}{+}$ **do**
                $Q[m][r] \mathrel{+}= S[m][k] * P[k][r];$
            end
        end
    end
end

```
void mul_matrix(matrix *result, matrix *mat1, matrix *mat2){
    int I = mat1->rows;
    int J = mat2->cols;
    int K = mat2->rows;
    #pragma omp parallel for
    for(int i = 0; i < I; i++){
        for(int k = 0; k < K; k++){
            _m256d vA = _mm256_set1_pd(mat1->data[i * K + k]);
            for(int j = 0; j < J / 4 * 4; j += 4){
                _m256d sum = _mm256_loadu_pd(result->data + i * J + j);
                _m256d vB = _mm256_loadu_pd(mat2->data + k * J + j);
                _m256d intermediate = _mm256_mul_pd(vA, vB);
                sum = _mm256_add_pd(sum, intermediate);
                _mm256_storeu_pd(result->data + i * J + j, sum);
             }
             for(int x = J / 4 * 4; x < J; x++){
                 result->data[i * J + x] += mat1 -> data[i * K + k] * mat2 ->
data[k * J + x];
             }
         }
     }
}
typedef struct matrix{
    int rows;
    int cols;
    double* data;
}matrix;
```

Algoritm S va P ning barcha yozuvlari bo‘ylab o‘tadi va eng tashqi sikl natijaviy Q matritsasini to‘ldiradi.

Time complexity: O(n^3)

Naive va strassen algoritmlari farqi

Parametrlar	**Naive**	**Strassen**
Time complexity	O(n^3)	O(n^2.8074)
Usul	Iterative	Divide and conquer
Ishlatilishi	Kvadrat va kvadrat bo`lmagan matritsalar	Faqat kvadrat matritsa

Xulosa

Xulosa qilib aytganda tenglamalar sistemasini yechishning Gauss usulida matrissa tuzilib , matrissaning diagonalining pastki qismi nolga aylantirilib yechiladi. Bu usul matrissaning bir qator yechimlari topishda yengillik yaratadi. Bular matrissaning rangi, determinant va unga teskari matritsa topish. Tenglamalar sistemasini yeching gauss usuli vaqtdan ancha yutqazganligi sabali turli xil muammolar keltirib chiqaradi. Internetdan Gauss elimination method is not optimal deb izlaganimda gauss usulidan optimalroq bo`lgan naive va Strassen algoritmlari kelib chiqdi. Strassen algoritmi Gauss usuliga qaraganda vaqt bo`yicha birozgina yutadi ya’ni taxminan 2.8 ga teng. Bu algoritm divide and conquer usulida (bo`lib tashlab hukmronlik qil) ishlaydi. Bunda berilgan matritsa 8ta bo`laklarga bo`linib yakunda 7ta matritsa ustida amalar bajaradi. Shu sababli time complexity=O(log(7))ga teng. Bu algoritmning kamchiligi esa faqatgina kvadrat matritsa uchun ishlaydi. Naive algoritmining time complexity=O($n^3$) ga teng. Lekin bu algoritm kvadrat bo`lmagan matrissalar uchun ham ishlaydi.

Foydalanilgan adabiyotlar va internet manzillari.

1. https://www.baeldung.com/cs/matrix-multiplication-algorithms#:~:text=The%20naive%20matrix%20multiplication%20algorithm%20contains%20three%20nested,of%20the%20matrix.%20Here%2C%20integer%20operations%20take%20time.
2. https://iq.opengenus.org/strassens-matrix-multiplication-algorithm/
3. https://staff.tiiame.uz/storage/users/436/presentations/uFM39RccLBWfgh1xe2zY6mAfAzmEfZ18KRw0jepJ.pdf
4. https://www.geeksforgeeks.org/gaussian-elimination/
5. https://www.sciencedirect.com/science/article/pii/S2352220816300529