Лабораторна робота №10

Матрична алгебра

Мета роботи: навчитися застосовувати оператори Python для прискорення операцій матричної алгебри.

Теоретичні відомості

Матрична алгебра. Нехай R – певна сукупність чисел. *Числовою матрицею* розміром $n \times m$ над R називається вектор A довжиною n, компоненти якого називаються *рядками* та є, у свою чергу, числовими векторами довжиною m над R. Для позначення елементів матриці використовують подвійну індексацію й пишуть $A = \left(a_{ij}\right)_{i=1\,j=1}^{n\;\;m}$, де a_{ij} означає j-й елемент у i-му рядку, або просто $A = \left(a_{ij}\right)$, якщо n та m фіксовані. Якщо $n = m$, то матриця називається *квадратною*. Основними операціями на матрицях є такі. Нехай $\lambda \in R$, $\bar{c} = \left(c_1,\ldots,c_n\right)$ – довільний вектор довжиною n на R, A,B,C – матриці розмірами відповідно $n \times m$, $n \times m$ та $m \times k$.

У результаті *множення матриці на скаляр і вектор* отримаємо матрицю $\lambda \circ A \overset{def}{=} \left(\lambda a_{ij}\right)_{i=1,\,j=1}^{n,m}$ того самого розміру $n \times m$ і вектор довжиною m $\bar{c} \times A \overset{def}{=} \left(d_j\right)_{j=1}^{m}$, $d_j = \sum_{l=1}^{n} c_l a_{lj}$, $j = \overline{1,m}$.

Додавання матриць повертає матрицю того самого розміру:

$$A + B \overset{def}{=} \left(a_{ij} + b_{ij}\right)_{i=1,\,j=1}^{n,m}.$$

Множення матриць повертає матрицю розміром $n \times k$:

$$A \times C \overset{def}{=} \left(d_{ij}\right)_{i=1\,j=1}^{n\;\;k}, \text{ де } d_{ij} = \sum_{l=1}^{m} a_{il} c_{lj}.$$

У матричній алгебрі є багато законів, що виконуються для чисел множини R. Наприклад, додавання цілих матриць асоціативне й комутативне, множення асоціативне тощо

АЛГОРИТМ ВИНОГРАДА ДЛЯ МНОЖЕННЯ МАТРИЦЬ

Кожен елемент добутку двох матриць є скалярним добутком відповідних рядка і стовпчика. Розглянемо скалярний добуток двох векторів – $V=(v_0,v_1,v_2,v_3)$ та $W=(w_0,w_1,w_2,w_3)$: $V\circ W=v_0w_0+v_1w_1+$ $+v_2w_2+v_3w_3=(v_0+w_1)(v_1+w_0)+(v_2+w_3)(v_3+w_2)-v_0v_1-v_2v_3-w_0w_1-$ $-w_2w_3$. Праву частину останнього виразу можна попередньо вирахувати й запам'ятати для кожного рядка першої матриці й кожного стовпчика другої. Це дозволяє виконувати для кожного елемента лише перші два множення та п'ять додавань, а також наступні два додавання.

Ш. Виноград для множення матриці G розміром $n\times m$ на матрицю H розміром $m\times k$ запропонував алгоритм, названий його іменем. Результат записується в матрицю R розміром $n\times k$:

$$a_i=\sum_{l=0}^{m/2} g_{i\,2l}g_{i\,2l+1};\ b_j=\sum_{l=0}^{m/2} h_{2l\,j}h_{(2l+1)\,j};$$

$$g_{ij}=\sum_{l=0}^{m/2}((g_{i\,2l+1}+h_{2l\,j})(g_{i\,2l}+h_{(2l+1)\,j}))-a_i-b_j+(g_{i\,m-1}h_{m-1\,j})(m-\text{непарне}).$$

Лістинг.

```
/*множення матриць методом Винограда*/
#define count1 5
#define count2 5
#define count3 5

double a[count1][count2];
double b[count2][count3];

double matrMult [count1][count3]; //результуюча матриця
double rowFactor[count1];  //сума добутків у кожному рядку
double columnFactors[count3]; //сума добутків у кожному стовпчику
//...
int i, j, k, d=(count2)/2;
```

```
//обчислення columnFactors для a
void calcRowFactors()
{
    for(i=0; i<count1; i++){
          rowFactor[i]=a[i][0] * a[i][1];
                for(j=1; j<d; j++){
                      rowFactor[i]+=a[i][2*j] * a[i][2*j+1];
          }
    }
}

//обчислення columnFactors для b
void calcColumnFactors()
{
    for(i=0; i<count3; i++){
          columnFactors[i]=b[0][i] * b[1][i];
          for(j=1; j<d; j++){
                columnFactors[i]+=b[2*j][i]* b[2*j+1][i];
          }

    }
}

//обчислення матриці matrMult
void mullMatr()
{
     for (i=0; i<count1; i++){
          for(j=0; j<count3; j++){
                matrMult[i][j]=-rowFactor[i] - columnFactors[j];
                for (k=0; k<d; k++){
                matrMult[i][j]+=(a[i][2*k+1]+b[2*k][j])*(a[i][2*k]+
b[2*k+1][j]);
                }
          }
     }
}

//обчислення матриці matrMult для випадку непарної розмірності
void mullMatr_1()
{
     if ((2 *((count2)/2))!=count2) {
          for(i=0; i<count1; i++){
                for(j=0; j<count3; j++){
                  matrMult[i][j]+=a[i][count2-1] * b[count2-1][j];
                }
          }
     }
}
```

```
int main()
{
  //... ініціалізація матриць a та b

  calcRowFactors();
  calcColumnFactors();
  mullMatr();
  mullMatr_1();
  // ...
  system("PAUSE");
  return 0;
}
```
■

Увага! Дана програма діє некоректно при `count2=1` ►

Верхня оцінка часової складності алгоритму Винограда для квадратних матриць – $O\left(n^{2.375477}\right)$.

АЛГОРИТМ ШТРАССЕНА ДЛЯ МНОЖЕННЯ МАТРИЦЬ

Розглянемо відкритий Штрассеном рекурсивний алгоритм, що множить дві $(n \times n)$-матриці за час $O\left(n^{\log_2 7}\right) = O\left(n^{2.81}\right)$.

Алгоритм Штрассена діє за принципом "розділяй і володій". Він працює з квадратними матрицями, розмір яких є степенем двійки, але настільки ефективний, що іноді доцільно розширити матриці до квадратних (додаванням нульових рядків чи стовпчиків), і при цьому він все одно дає виграш.

Нехай потрібно обчислити добуток двох матриць $(n \times n)$: $C = AB$. Нехай n є точним степенем двійки, тоді розділимо кожну з матриць A, B та C на 4 блоки розміром $(n/2 \times n/2)$. Перепишемо рівняння $C = AB$ таким чином:

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} * \begin{pmatrix} e & g \\ f & h \end{pmatrix},$$

де $r = ae + bf$, $s = ag + bh$, $t = ce + df$, $u = cg + dh$.

Тут відбувається вісім множень матриць розміром $n/2$, тому в такому вигляді алгоритм, як і послідовний алгоритм, працює за час $O\left(n^{\log_2 8}\right)$. Алгоритм Штрассена пропонує схему, за якої в рекурсивному алгоритмі можна обійтися лише сімома множеннями матриць розміром $n/2$, тим самим скоротивши час до $O\left(n^{\log_2 7}\right) = O\left(n^{2.81}\right)$.

Алгоритм Штрассена множить дві матриці $(n \times n)$ A та B таким чином:

$x_1 = ag - ah$;
$x_2 = ah + bh$;
$x_3 = ce + de$;
$x_4 = df - de$;

$x_5 = ae + ah + de + dh$;
$x_6 = bf + bh - df - dh$;
$x_7 = ae + ag - ce - cg$.
$r = x_5 + x_4 - x_2 + x_6 = ae + de - d + bf$;
$s = x_1 + x_2 = ag + bh$;
$t = x_3 + x_4 = ce + df$;
$u = x_5 + x_1 - x_3 - x_7 = cg + dh$.

Отже, при множенні двох матриць 2×2 алгоритм виконує 7 множень і 18 додавань. На практиці алгоритм Штрассена застосовується порівняно рідко через суттєву величину константи, що міститься в асимптотичному виразі для часу його роботи. Його застосування виправдане для великих матриць (від 45×45) з малою кількістю нулів.

ПАРАЛЕЛЬНІ АЛГОРИТМИ МНОЖЕННЯ МАТРИЦІ НА ВЕКТОР

http://www.intuit.ru/studies/courses/1156/190/lecture/4952

ПАРАЛЕЛЬНІ АЛГОРИТМИ МНОЖЕННЯ МАТРИЦЬ

http://www.intuit.ru/studies/courses/1156/190/lecture/4954

Завдання: **Розробити алгоритм**, **який здійснює операцію над матрицями або векторами відповідно до варіанта**. **Показати**, **що зазначена операція виконується швидше**, **ніж звичайна операція множення матриць** (**або відповідна операція множення матриці на вектор**).

Варіант 1. Алгоритм Винограда для множення матриць.

Варіант 2. Алгоритм Штрассена для множення матриць

Варіант 3. Алгоритм множення матриці на вектор при розподілі даних за рядками

Варіант 4. Алгоритм множення матриці на вектор при розподілі даних за стовпцями

Варіант 5. Множення матриці на вектор при блочному розподілі даних

Варіант 6. Алгоритм множення матриць при стрічкової схемі розподілу даних

Варіант 7. Алгоритм Фокса множення матриць при блочному розподілі даних

Варіант 8. Алгоритм Кеннона множення матриць при блочному розподілі даних