Calculating the determinant using Kraut method in $O(N^3)$

Table of Contents

In this article, we'll describe how to find the determinant of the matrix using Kraut method, which works in $O(N^3)$.

The Kraut algorithm finds decomposition of matrix A as $A = LU$ where L is lower triangular and U is upper triangular matrix. Without loss of generality, we can assume that all the diagonal elements of L are equal to 1. Once we know these matrices, it is easy to calculate the determinant of A: it is equal to the product of all the elements on the main diagonal of the matrix U.

There is a theorem stating that any invertible matrix has a LU-decomposition, and it is unique, if and only if all its principle minors are non-zero. We consider only such decomposition in which the diagonal of matrix L consists of ones.

Let A be the matrix and N - its size. We will find the elements of the matrices L and U using the following steps:

1. Let $L_{ii} = 1$ for $i = 1, 2, \dots, N$.
2. For each $j = 1, 2, \dots, N$ perform:
   - For $i = 1, 2, \dots, j$ find values

$$U_{ij} = A_{ij} - \sum_{k=1}^{i-1} L_{ik} \cdot U_{kj}$$

   - Next, for $i = j+1, j+2, \dots, N$ find values

$$L_{ij} = \frac{1}{U_{jj}} \left(A_{ij} - \sum_{k=1}^{j-1} L_{ik} \cdot U_{kj} \right)$$

.

Implementation

```
static BigInteger det (BigDecimal a [][], int
    try {

        for (int i=0; i<n; i++) {
            boolean nonzero = false;
            for (int j=0; j<n; j++)
                if (a[i][j].compareTo (new BigDeci
                    nonzero = true;
            if (!nonzero)
                return BigInteger.ZERO;
        }
```

```
BigDecimal scaling [] = new BigDecimal [n]
for (int i=0; i<n; i++) {
    BigDecimal big = new BigDecimal (BigIn
    for (int j=0; j<n; j++)
        if (a[i][j].abs().compareTo (big)
            big = a[i][j].abs();
    scaling[i] = (new BigDecimal (BigInteg
        (big, 100, BigDecimal.ROUND_HALF_E
}

int sign = 1;

for (int j=0; j<n; j++) {
    for (int i=0; i<j; i++) {
        BigDecimal sum = a[i][j];
        for (int k=0; k<i; k++)
            sum = sum.subtract (a[i][k].mu
        a[i][j] = sum;
    }

    BigDecimal big = new BigDecimal (BigIn
    int imax = -1;
    for (int i=j; i<n; i++) {
        BigDecimal sum = a[i][j];
        for (int k=0; k<j; k++)
            sum = sum.subtract (a[i][k].mu
        a[i][j] = sum;
        BigDecimal cur = sum.abs();
        cur = cur.multiply (scaling[i]);
        if (cur.compareTo (big) >= 0) {
            big = cur;
            imax = i;
```

```java
            }
        }

        if (j != imax) {
            for (int k=0; k<n; k++) {
                BigDecimal t = a[j][k];
                a[j][k] = a[imax][k];
                a[imax][k] = t;
            }

            BigDecimal t = scaling[imax];
            scaling[imax] = scaling[j];
            scaling[j] = t;

            sign = -sign;
        }

        if (j != n-1)
            for (int i=j+1; i<n; i++)
                a[i][j] = a[i][j].divide
                    (a[j][j], 100, BigDecimal.

    }

    BigDecimal result = new BigDecimal (1);
    if (sign == -1)
        result = result.negate();
    for (int i=0; i<n; i++)
        result = result.multiply (a[i][i]);

    return result.divide
        (BigDecimal.valueOf(1), 0, BigDecimal.
    }
```

```
    catch (Exception e) {
        return BigInteger.ZERO;
    }
}
```