Perbandingan kinerja algoritma dalam beberapa bahasa pemrograman dengan paradigma pemrograman berbeda

Dimas Angga Saputra / 13510046
Program Studi Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13510046@std.stei.itb.ac.id

***Abstract*—Algoritma *Gaussian-Elimination* merupakan algoritma untuk menyelesaikan sistem persamaan linear dari suatu matriks. Meskipun dengan alur metode yang sama, namun bahasa pemrograman berbeda akan menghasilkan kinerja yang berbeda pula, Makalah ini akan membandingkan kinerja beberapa bahasa pemrograman untuk mengaplikasikan salah satu algoritma Aljabar Geometri.**

***Keywords*— Fungsional, Imparatif, Python, Lisp, Java, Gaussian**

I. PENDAHULUAN

. Algoritma *Gaussian-Elimination* merupakan algoritma untuk menyelesaikan sistem persamaan linear dari suatu matriks. Meskipun dengan alur metode yang sama, namun bahasa pemrograman berbeda akan menghasilkan kinerja yang berbeda pula, faktor perbedaan paradigma, fungsi primitif, dan kemampuan bahasa pemrograman menjadi faktor yang diduga menyebabkan peredaan kinerja tersebut.

Makalah ini akan membandingkan kinerja beberapa bahasa pemrograman untuk mengaplikasikan salah satu algoritma Aljabar Geometri.

II. DASAR TEORI

A. Pseudo Code

Pseudo code Algoritma yang akan digunakan adalah sebagai berikut :

Algorithm 1 Gaussian elimination

```
function GAUSS(A ∈ ℝ^{n×n+1})
  for (i = 1; i ≤ n; i++) do
    // Search for maximum in this column
    maxEl = |A_{i,i}|
    maxRow = i
    for (k = i + 1; k ≤ n; k++) do
      if |A_{k,i}| > maxEl then
        maxEl = A_{k,i}
        maxRow = k
      end if
    end for

    // Swap maximum row with current row
    for (k = i; k ≤ n; k++) do
      tmp = A_{maxRow,k}
      A_{maxRow,k} = A_{i,k}
      A_{i,k} = tmp
    end for

    // Make all rows below this one 0 in current column
    for (k = i + 1; k ≤ n; k++) do
      c = -A_{k,i}/A_{i,i}
      for (j = i; j ≤ n; j++) do
        if i == j then
          A_{k,j} = 0
        else
          A_{k,j} += c · A_{i,j}
        end if
      end for
    end for
  end for

  // Solve equation for an upper triangular matrix
  x = { 0 } ∈ ℝ^n
  for (i = n; i ≥ 1; i--) do
    x_i = A_{i,n+1}/A_{i,i}
    for (k = i - 1; k ≥ 1; k--) do
      A_{k,n+1} -= A_{k,i} · x_i
    end for
  end for

  return x
end function
```

B. Paradigma pemrograman

Bahasa-bahasa yang dipilih untuk dibandingkan adalah bahasa Python, CommonLisp, Haskell, dan Java. Keempatnya dipilih karena mewakili paradigma berbeda, Python mewakili prosedural imparatif, CommonLisp mewakili fungsional list, Haskell mewakili fungsional non lisp, dan Java mewakili imparatif berorientasi objek.

Perbedaan imparatif dan fungsional adalah sebagai berikut

Characteristic	Imperative approach	Functional approach
Programmer focus	How to perform tasks (algorithms) and how to track changes in state.	What information is desired and what transformations are required.
State changes	Important.	Non-existent.
Order of execution	Important.	Low importance.
Primary flow control	Loops, conditionals, and function (method) calls.	Function calls, including recursion.
Primary manipulation unit	Instances of structures or classes.	Functions as first-class objects and data collections.

III. Percobaan

A. Kode dalam Python

```
def pprint(A):
    n = len(A)
    for i in range(0, n):
        line = ""
        for j in range(0, n+1):
            line += str(A[i][j]) +
"\t"
            if j == n-1:
                line += "| "
        print(line)
    print("")


def gauss(A):
    n = len(A)

    for i in range(0, n):
        # Search for maximum in this
column
        maxEl = abs(A[i][i])
        maxRow = i
        for k in range(i+1, n):
            if abs(A[k][i]) > maxEl:
                maxEl = abs(A[k][i])
                maxRow = k

        # Swap maximum row with
current row (column by column)
        for k in range(i, n+1):
            tmp = A[maxRow][k]
            A[maxRow][k] = A[i][k]
            A[i][k] = tmp

        # Make all rows below this one
0 in current column
        for k in range(i+1, n):
            c = -A[k][i]/A[i][i]
            for j in range(i, n+1):
                if i == j:
                    A[k][j] = 0
                else:
                    A[k][j] += c *
A[i][j]

    # Solve equation Ax=b for an upper
triangular matrix A
    x = [0 for i in range(n)]
    for i in range(n-1, -1, -1):
        x[i] = A[i][n]/A[i][i]
        for k in range(i-1, -1, -1):
            A[k][n] -= A[k][i] * x[i]
    return x


if __name__ == "__main__":
    A =
[[1.00,0.00,0.00,0.00,0.00,0.00,-0.01]
,
[1.00,0.63,0.39,0.25,0.16,0.10,0.61],
[1.00,1.26,1.58,1.98,2.49,3.13,0.91],
[1.00,1.88,3.55,6.70,12.62,23.80,0.99]
,
[1.00,2.51,6.32,15.88,39.90,100.28,0.6
0 ],
[1.00,3.14,9.87,31.01,97.41,306.02,0.0
2]]
    #b = [[-0.01], [0.61], [0.91],
[0.99],[0.60],[0.02]]

    # Print input
    pprint(A)

    # Calculate solution
    x = gauss(A)

    print (x)
```

B. Kode dalam Common Lisp

```
(defmacro mapcar-1 (fn n list)
  "Maps a function of two parameters where the first one
is fixed, over a list"
    `(mapcar #'(lambda (l) (funcall ,fn ,n l)) ,list) )


(defun gauss (m)
  (labels
    ((redc (m) ; Reduce to triangular form
      (if (null (cdr m))
        m
          (cons (car m) (mapcar-1 #'cons 0 (redc (mapcar
#'cdr (mapcar #'(lambda (r) (mapcar #'- (mapcar-1 #'*
```

```
(caar m) r)

(mapcar-1 #'* (car r) (car m)))) (cdr m)))))) ))
       (rev (m) ; Reverse each row except the last element
             (reverse (mapcar #'(lambda (r) (append (reverse (butlast r)) (last r))) m)) ))
      (catch 'result
        (let ((m1 (redc (rev (redc m)))))
          (reverse (mapcar #'(lambda (r) (let ((pivot (find-if-not #'zerop r))) (if pivot (/ (car (last r)) pivot) (throw 'result 'singular)))) m1)) ))))
```

C. Kode dalam Haskell

```
type Row = [Float]
type Matrix = [Row]

gaussianReduce :: Matrix -> Matrix
gaussianReduce matrix = fixlastrow $ foldl reduceRow matrix [0..length matrix-1] where

 --swaps element at position a with element at position b.
 swap xs a b
  | a > b = swap xs b a
  | a == b = xs
  | a < b = let
  (p1,p2) = splitAt a xs
  (p3,p4) = splitAt (b-a-1) (tail p2)
  in p1 ++ [xs!!b] ++ p3 ++ [xs!!a] ++ (tail p4)

 reduceRow matrix1 r = let
  --first non-zero element on or below (r,r).
  firstnonzero = head $ filter (\x -> matrix1 !! x !! r /= 0) [r..length matrix1-1]

  --matrix with row swapped (if needed)
  matrix2 = swap matrix1 r firstnonzero

  --row we're working with
  row = matrix2 !! r

  --make it have 1 as the leading coefficient
  row1 = map (\x -> x / (row !! r)) row

  --subtract nr from row1 while multiplying
  subrow nr = let k = nr!!r in zipWith (\a b -> k*a - b) row1 nr

  --apply subrow to all rows below
  nextrows = map subrow $ drop (r+1) matrix2

  --concat the lists and repeat
  in take r matrix2 ++ [row1] ++ nextrows

 fixlastrow matrix' = let
  a = init matrix'; row = last matrix'; z = last row; nz = last (init row)
  in a ++ [init (init row) ++ [1, z / nz]]
```

D. Kode dalam JAVA

GaussianElimination.java

Below is the syntax highlighted version of GaussianElimination.java from §9.5 Numerical Linear Algebra.

```
/*************************************************************************
 *  Compilation:  javac GaussianElimination.java
 *  Execution:    java GaussianElimination
 *
 *  Gaussian elimination with partial pivoting.
 *
 *  % java GaussianElimination
 *  -1.0
 *  2.0
 *  2.0
 *
 *************************************************************************/

public class GaussianElimination {
    private static final double EPSILON = 1e-10;

    // Gaussian elimination with partial pivoting
    public static double[] lsolve(double[][] A, double[] b) {
        int N  = b.length;

        for (int p = 0; p < N; p++) {

            // find pivot row and swap
            int max = p;
            for (int i = p + 1; i < N; i++) {
                if (Math.abs(A[i][p]) > Math.abs(A[max][p])) {
                    max = i;
                }
            }
            double[] temp = A[p]; A[p] = A[max]; A[max] = temp;
            double   t    = b[p]; b[p] = b[max]; b[max] = t;

            // singular or nearly singular
            if (Math.abs(A[p][p]) <= EPSILON) {
                throw new RuntimeException("Matrix is singular or nearly singular");
            }

            // pivot within A and b
            for (int i = p + 1; i < N; i++) {
                double alpha = A[i][p] / A[p][p];
                b[i] -= alpha * b[p];
                for (int j = p; j < N; j++) {
                    A[i][j] -= alpha * A[p][j];
```

}

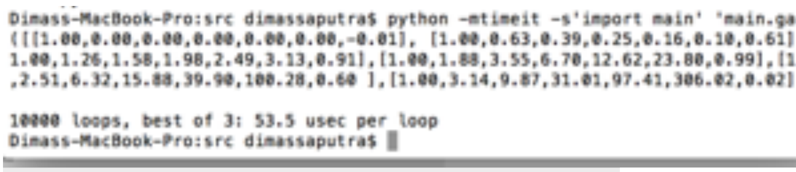

```
substitution
        double[] x = new double[N];
        for (int i = N - 1; i >= 0; i--) {
            double sum = 0.0;
            for (int j = i + 1; j < N; j++) {
                sum += A[i][j] * x[j];
            }
            x[i] = (b[i] - sum) / A[i][i];
        }
        return x;
    }


    // sample client
    public static void main(String[] args) {
        int N = 3;
        double[][] A = { { 0, 1,  1 },
                         { 2, 4, -2 },
                         { 0, 3, 15 }
                       };
        double[] b = { 4, 2, 36 };
        double[] x = lsolve(A, b);


        // print results
        for (int i = 0; i < N; i++) {
            System.out.println(x[i]);
```

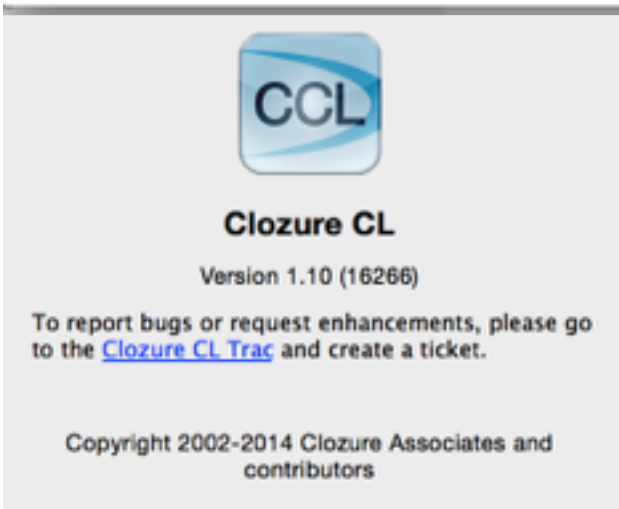

```
        }

    }
```

```
    }

}
    }

    // back
```

IV. HASIL PERCOBAAN

Seluruh percobaan dilakukan dengan spesifikasi :

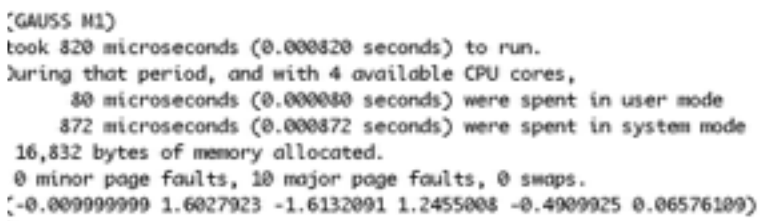




Mesin :

IDE :

GHCI versi 7.10.1



Bahasa :
Java 1.8
Python 2.7

Matriks uji :

```
[1.00,0.00,0.00,0.00,0.00,0.00,-0.01],
[1.00,0.63,0.39,0.25,0.16,0.10,0.61],
[1.00,1.26,1.58,1.98,2.49,3.13,0.91],
[1.00,1.88,3.55,6.70,12.62,23.80,0.99]
,
[1.00,2.51,6.32,15.88,39.90,100.28,0.6
0 ],
[1.00,3.14,9.87,31.01,97.41,306.02,0.0
2]
```

```
Pengukuran :

Dengan commonlisp

Dengan Python

Dengan Java
```

```
64321
-0.01
1.6027903945020974
-1.6132030599054943
```

```
Dengan haskell
```

```
*Main> gaussianReduce [[1.00,0.00,0.00,0.00
,2.51,6.32,15.88,39.90,100.28,0.60 ],[1.00,
[[1.0,0.0,0.0,0.0,0.0,0.0,-1.0e-2],[-0.0,1.
3,9.588836,6.775985e-2],[0.0,0.0,0.0,0.0,1.
(0.03 secs, 10,361,376 bytes)
```

Jika satuan waktu diubah ke milisecond maka :

Haskell : 30 ms
Java : 0.006431 ms
Python : 0.0535ms
CommonLisp : 0.82 ms

V. Simpulan

Java adalah bahasa pemrograman yang tercepat untuk membandingkan kinerja algoritma Gaussian elimination, namun dari segi jumlah baris, CommonLisp adalah yang tersepat

References

1. http://martin-thoma.com/solving-linear-equations-with-gaussian-elimination/ Wakti akses 16 desember 2015 11:00
2. https://msdn.microsoft.com/en-us/library/bb669144.aspx Wakti akses 16 desember 2015 11:00
3. http://introcs.cs.princeton.edu/java/95linear/GaussianElimination.java.html Wakti akses 16 desember 2015 11:00
4. https://luckytoilet.wordpress.com/2010/02/21/solving-systems-of-linear-equations-in-haskell/ Wakti akses 16 desember 2015 11:00

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 27 November 2013

ttd

Nama dan NIM