KAJIAN ALGORITMA KOMPUTASI
PERSAMAAN DIFFERENSIAL PARSIAL (PDP)

Agus Sujono *)

ABSTRACT

The method of numerical analysis is very useful to solve many problems that very difficult to hold by conventional analytical analysis. But its need much power of computing if the method is not good. Some of the popular methods are Gauss-Seidel, elimination Gauss and Doolittle, that have different algorithms of computing, that to be compared. The result of the compare of the algorithms is the time execution or the flops is depend on the degree of the element of the matrix, but in many cases the Gauss-Seidel method is the better to use.

1. PENDAHULUAN

Kemajuan komputer yang sangat cepat telah banyak memberi nuansa baru dalam bidang metoda komputasi, yang banyak mengarah pada metoda komputasi numeris. Sebab dengan metoda ini waktu komputasi dapat dipersingkat dan volume komputasi tidak terlalu dibatasi oleh karena kemampuan sumber daya yang ada. Selain itu juga banyak solusi bisa diperoleh dengan metoda ini dan bila dengan metode analitis terlalu sulit untuk memecahkannya.

Oleh karena itu untuk mendukung komputasi diberbagai bidang, metoda komputasi numeris perlu terus dikembangkan, yaitu dengan membuat algoritma komputasinya. Hal ini termasuk mencari metoda yang paling menguntungkan, agar dapat lebih bnyak hal dapat ditangani dan lebih cepat pengerjaannya.

Berikut ini kajian algoritma aplikasi dari permasalan perpindahan panas, mekanika fluida dan struktur / konstruksi baja, yang diselesaikan dengan metoda komputasi numeris.

1.1. Persoalan Perpindahan Panas.

Diambil contoh masalah perpindahan panas konduksi, dalam suatu bahan yang cukup besar pada kondisi transien, untuk koordinat kartesian x, y, z :

$$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}+\frac{\partial^2 T}{\partial z^2}+\frac{q'}{k}=\frac{1}{\alpha}\frac{\partial T}{\partial t}$$

dimana : T = suhu, t = waktu, q' = panas yang terjadi

*) Agus Sujono : staf pengajar Teknik Mesin, Fak.Teknik, UNS

1.2 Persoalan Mekanika Fluida

Aliran fluida secara umum mempunyai kecepatan yang tidak sama disetiap tempat, dan bila fluida tersebut adalah kompresibel, maka kondisinya berubah, lebih-lebih bila keadaan tidak ajeg (non steady state) maka akan selalu berubah terhadap waktu. Hal ini menimbulkan persamaan diferensial parsial multi-dimensional, sebagai berikut :

Rumus vektor bentuk :

$L(\rho_o) = \nabla \cdot \mathbf{u} = 0$

$$L(u) = \frac{\partial u}{\partial t} + (u.\nabla)u - \frac{1}{R_e}\nabla^2 u + \frac{1}{\rho_o}\nabla p + \frac{1}{F_r} b = 0$$

$$L(T) = \frac{\partial T}{\partial t} + (u.\nabla)T - \frac{1}{R_e P_r}\nabla^2 T + \sigma_{ij}\frac{\partial u_i}{\partial x_j} = 0$$

dimana : u = vektor kecepatan, T = suhu, R_e = Renold number

P_r = Prandtl number, F_r = Froudle number

1.3. Persoalan Struktur Konstruksi

Dalam struktur rangka ruang banyak digunakan metoda matrik, sebab komponen rangka batang berjumlah sangat banyak, sehingga bila diselesaikan secara manual akan sangat lama dan melelahkan.

Perhitungan gaya akan didasarkan pada persamaan kekakuan dari batang yang bersangkutan didalam sistim konstruksi terpadu sebagai berikut :

$\{P\} = [K]\{\delta\}$

dimana : P = gaya, K = kekakuan, δ = defleksi

$$\begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \\ P_7 \\ P_8 \\ P_9 \\ P_{10} \\ P_{11} \\ P_{12} \end{bmatrix} = \begin{bmatrix} \frac{EA}{L} & & & & & & & & & & & \\ 0 & \frac{12EI_z}{L^3} & & & & & & & \text{Simetris} & & & \\ 0 & 0 & \frac{12EI_y}{L^3} & & & & & & & & & \\ 0 & 0 & 0 & \frac{GJ}{L} & & & & & & & & \\ 0 & 0 & \frac{-6EI_y}{L^2} & 0 & \frac{4EI_y}{L} & & & & & & & \\ 0 & \frac{6EI_z}{L^3} & 0 & 0 & 0 & \frac{4EI_z}{L} & & & & & & \\ \frac{-EA}{L} & 0 & 0 & 0 & 0 & 0 & \frac{EA}{L} & & & & & \\ 0 & \frac{-12EI_z}{L^2} & 0 & 0 & 0 & \frac{-6EI_z}{L^2} & 0 & \frac{12EI_z}{L^3} & & & & \\ 0 & 0 & \frac{-12EI_y}{L^3} & 0 & \frac{6EI_y}{L^2} & 0 & 0 & 0 & \frac{12EI_y}{L^3} & & & \\ 0 & 0 & 0 & \frac{GJ}{L^2} & 0 & 0 & 0 & 0 & 0 & \frac{GJ}{L} & & \\ 0 & 0 & \frac{-6EI_y}{L^2} & 0 & \frac{2EI_y}{L} & 0 & 0 & 0 & \frac{6EI_y}{L^3} & 0 & \frac{4EI_y}{L} & \\ 0 & \frac{6EI_z}{L^3} & 0 & 0 & 0 & \frac{2EI_z}{L} & 0 & \frac{-6EI_z}{L^2} & 0 & 0 & 0 & \frac{4EI_z}{L} \end{bmatrix} \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \\ \delta_5 \\ \delta_6 \\ \delta_7 \\ \delta_8 \\ \delta_9 \\ \delta_{10} \\ \delta_{11} \\ \delta_{12} \end{bmatrix}$$

Matrik ini adalah hanya satu elemen saja, padahal kenyataannya berjumlah banyak, sehingga untuk penyelesaiannya tidak efisien bila dengan perhitungan matrik biasa, maka telah ada metoda perhitungan dengan matrik jarang (sparse matrix).

2. Teori Analisis Algotitma

Masalah di dalam analisis algoritma dibagi menjadi atas :

1. Masalah Numeris, yaitu membuat model matematika.
2. Masalah Simbol dan Teks yaitu string atau format grammars.

Keduanya akan menyusun suatu algoritma untuk membentuk sebuah program. Algoritma didefinisikan sebagai suatu barisan instruksi tersebut memiliki suatu arti yang jelas (clear meaning) dan dapat dijalankan dengan sejumlah usaha dalam jangka waktu tertentu.

Algorima terkadang berupa :

1. Solusi optimal
2. Baik tetapi bukan merupakan solusi optimal yang penting (heuristik).

Untuk penyelesaian masalah-masalah tersebut dapat diselesaikan dengan menggunakan beberapa algoritma, kemudian memilih algoritma yang tepat.

Ada 2 tujuan mengapa suatu algoritma dipilih yaitu :

1. Suatu algorima seharusnya mudah dimengerti, disandikan dan didebug. yang dipentingkan adalah penulisan suatu program yang akan digunakan sekali atau beberapa kali. Biaya untuk penulisan program lebih besar daripada biaya unuk running program.
2. Suatu algoritma seharusnya dapat menyebabkan penggunaan komputer menjadi lebih efisien, khususnya algoritma tersebut dapat berjalan secepat mungkin. Untuk suatu problem dengan solusi yang dipergunakan beberapa kali, biaya untuk running program lebih besar daripada biaya untuk menulis program.

Comment [A1]:

Mengukur *running time* dari suatu program tergantung dari beberapa faktor di bawah ini :

1. Masukan (*input*) program
2. Kualitas dari sandi (code) yang dihasilkan oleh kompiler yang dipergunakan untuk menulis (*create*) objek program.
3. Perlaku dan kecepatan instruksi-instruksi mesin yang dipergunakan untuk mengeksekusi program.
4. Kompleksitas waktu algoritma yang menjadi dasar suatu program.

Running time dari suatu program seharusnya didefinisikan sebagai suatu fungsi dari masukan, biasanya pada ukuran masukan *n* , ditulis T(n). *Running time* pada beberapa program, benar-benar merupakan suatu fungsi dari masukan tertentu (bukan hanya ukuran dari masukan). Dalam hal ini, jika T(n) adalah *Worst Case,* yaitu running time maksimal dari semua *running time* pada masukan dengan ukuran *n,* maka T(n) adalah *time complexity* dari program.

Pernyataan 1 dan 3, *running time* T(n) dapat dinyatakan dalam satuan standar waktu (misalnya detik).

Notasi Big-Oh dipergunakan pada *growth rate* suatu fungsi. Running time T(n) dari beberapa program adalah $O(n^2)$, dapat dibaca " big-oh dari n kwadrat". Pernyataan ini berarti bahwa ada konstanta positif c dan n_0 sedemikian hingga untuk setiap n sedemikian sehingga untuk setiap $n \geq n_o$, $T(n) \leq cn^2$.

Running time $T(n) \approx O(f(n))$ jika terdapat konstanta positif c sedemikian sehingga $T(n) \leq cf(n)$ untuk setiap $n \geq n_0$, program dikatakan memiliki *growth rate* f(n). Fungsi f(n) merupakan batas atas (upper bound) pada growth rate T(n). Untuk menspesifikasikan batas bawah (lower bound) pada growth rate T(n) dapat dipergunakan notasi W(g(n)), dibaca dengan "big-Omega dari g(n)", artinya terdapat suatu konstanta c sedemikian sehingga $T(n) \geq cg(n)$. Suatu prinsip

asimetrikantara big-oh dan big Omega seringkali cepat pada beberapa masalah algoritma dan tidak cepat pada semua masukkan.

Berpedoman pada fungsi-fungsi running time dari program, dengan mempertimbangkan program-program yang memiliki growth rate serendah mungkin sejak growth rate akhirnya menentukan berapa besarnya n , maslah dapat diselesaikan dengan komputer.

Growth rate dari suatu worst case running time bukanlah suatu kejadian tunggal atau perlu, yang terpenting adalah kriteria untuk mengevaluasi algoritma atau program yaitu :

1. Jika suatu program hanya dipergunakan beberapa kali saja, maka biaya untuk menulis dan debugging akan mendominasi keseluruhan biaya running time saat itu, dan sudah tentu akan berpengaruh pula pada biaya keseluruhan, sehingga perlu dipilih algoritma yang aksesnya dapat diimplementaskan dengan benar.
2. Jika suatu program hanya membutuhkan 'sedikit' masukan, maka growth rate dari running time akan lebih kecil, dan terlihat dari konstanta c pada rumusan di atas.
3. Algoritma yang sukar tetapi efisisen, biasanya tidak diinginkan, karena selain penulis program akan sukar untuk memahami algoritma tersebut.
4. Ada beberapa contoh algoritma efisien yang menggunakan begitu banyak ruang untuk diimplementasikan tanpa menggunakan pengingat sekunder yang berjalan lambat, hal ini lebih banyak dijumpai dibandingkan dengan harus membatalkan algoritma tersebut.
5. Pada algoritma numeris, ketepatan dan stabilitasnya sama pentingnya dengan efisiensi algoritma.

Menghitung *running time* dari berbagai macam program merupakan masalah yang sangat kompleks terutama dalam hal matematikanya. Andaikan $T_1(n)$ dan $T_2(n)$ adalah *running time* dari 2 fragmen program yaitu P_1 dan P_2. $T_1(n)$ dinyatakan sebagai O(f(n)) dan $T_2(n)$ sebagai O(g(n)), maka $T_1(n) + T_2(n)$ adalah *running time* dari P_1 diikuti oleh P_2. Dalam hal ini $T_1(n) + T_2(n)$ dapat ditulis sebagai O(max(f(n),g(n))). Untuk setiap konstanta c_1 dan c_2, n_1 dan n_2, jika $n \geq n1$, maka $T_1(n) \leq c_1\ f(n)$ dan jika $n \geq n_2$ maka $T_2(n) \leq c_2 g(n)$. Diambil $n_0 = \max(n_1,n_2)$, jika $n \geq n0$ maka $T_1(n) + T_2(n) \leq c_1 f(n) + C_2 g(n)$. Oleh karena itu $T_1(n) + T_2(n)$ adlah O(max(f(n),g(n))) yang kemudian dikenal sebagai *rule of sum.*

2.1. Rule of Sum

Rule of sum dapat dipergunakan untuk menentukan *running time* dari suatu barisan langkah-langkah program dimana masing-masing langkah mungkin merupakan suatu fragmen program sembarang dengan kalang (*loop*) dan cabang (*branch*). Andaikan ada 3 langkah program dengan *running time* $0(n^2)$, $O(n^3)$ dan (n log n), maka *running time* dari 2 langkah pertama yang dieksekusi secara berurutan adalah $O(\max(n^2,n^3))$ yaitu $O(n^3)$. *Running time* dari ketiga langkah bersamaan adalah $O(\max(n^2,n^3,n\log n))$ yaitu $O(n^3)$.

Secara umum, *running time* dari suatu barisan berhingga langkah-langkah dengan menggunakan faktor konstan adalah sama dengan *running time* dari langkah yang memiliki *running time* terbesar.

2.2. Rule of Product

Jika $T_1(n)$ dan $T_2(n)$ memiliki *running time* O(f(n)) dan O(g(n)), maka $T_1(n).T_2(n)$ memiliki *running time* O(f(n),g(n)) yang kemudian disebut sebagai ***rule of product.***

O(cf(n)) sama dengan O(f(n)) untuk sembarang bilangan positif c. Bila *running time* = $O(n^2)$. Program tersebut membutuhkan waktu yang sebanding dengan kwadrat anggota dari item yang diurutkan.

Secara umum, aturan untuk menganalisa suatu program adalah sebagai berikut:

1. *Running time* dari tiap-tiap tugas, membaca dan menulis ungkapan sebesar O(1).
2. *Running time* dari suatu barisan ungkapan (*statement*) ditentukan oleh ***rule for sums***. *Running time* dari beberapa barisan akan menjadi *running time* terbesar dari barisan-barisan yang ada di dalammnya.
3. *Running time* dari ungkapan ***IF*** adalah biaya dari ungkapan-ungkapan yang mengkondisikan apakah instruksi akan dieksekusi atau tidak, ditambah dengan waktu eksekusi dari kondisi tersebut. Waktu yang dibutuhkan untuk ungkapan ***IF-THEN-ELSE*** adalah waktu untuk mengevaluasi kondisi ditambah waktu terbesar yang dibutuhkan untuk mengeksekusi ungkapan jika kondisi tersebut benar dan waktu untuk mengeksekusi ungkapan jika kondisi tersebut salah.
4. Waktu untuk mengeksekusi suatu kalang (*loop*) adalah jumlah semua waktu yang dibutuhkan untuk mengeksekusi badan kalang (*body loop*) dan waktu untuk mengeksekusi kondisi agar kalang berhenti.

2. PENERAPAN PADA PDP

Persamaan diferensial parsial (PDP) linier orde dua (*Partial Diferential Equation /PDE*), seringkali mengacu kepada, seperti tertulis di atas, sebagai elliptik, hiperbolik atau parabolik. Klasifikasi seperti hal tersebut di atas, bisa jika persamaannya telah dikurangi dengan transformasi yang cocok dari variabel-variabel indenpenden, ke bentuk

$$\sum_{i=1}^{n} A_i \frac{\partial^2 u}{\partial x_i^2} + \sum_{i=1}^{n} B_i \frac{\partial u}{\partial x_i} + cu + D = 0 \qquad (1\text{-}1)$$

dimana koefisien A_i , dihitung pada titik (x_1, x_2, ...x_n) bisa 1, -1 atau nol. Disini *u* adalah variabel mandiri, dan x_i adalah variabel bebas. Catatlah ketiadaan turunan campuran dalam bentuk $\partial^2 u/\partial x_i \partial x_j \ (i \neq j)$.

Selama koefisien *Ai , Bi , C,* dan *D* adalah fungsi dari variabel mandiri x_1, x_2,.....,x_n, klasifikasi PDE dapat bermacam-macam sesuai dengan titik yang tertentu yang di diperhatikan pada ruang (x_1, x_2,.....x_n). Sangat sering, salah satu variabel mandiri bisa jadi waktu *t* dan sisanya adalah sebuah jarak koordinat *x, y,* dan *z*.

Dapat ditunjukan bahwa, untuk dua dimensi bidang xy dengan *u* sebagai sebuah variabel mandiri kita mempunyai bentuk persamaan :

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \nabla^2 u = 0$$ disebut persamaan Laplace.

Dapat ditunjukan dalam bidang xy, jika kita mempunyai fungsi f(x,y) kemudian mengikuti :

$$\nabla^2 u = f(x,y)$$ disebut persamaan Poisson.

Metode numerik untuk menyelesaikan persamaan diferensial elliptik dipergunakan sama pada kedua persamaan Laplace dan Poisson. Persamaan menjelaskan bagaimana *u* bervariasi dibagian dalam sebuah wilayah tertutup. Fungsi ditetapkan pada batas wilayah oleh kondisi

batas. Kondisi batas bisa menspesifikasikan nilai *u* pada semua titik pada batas atau beberapa kombinasi.

Metode memecahkan masalah persamaan diferensial parsial. Diantaranya, adalah metode Gauss-Seidel, Eliminasi Gauss, dan Doolittle.

Persamaan linear simultan dari bentuk :

$$A * \underline{v} = \underline{b}$$

Beberapa metode faktorisasi matrik *A* yang ada, diantaranya adalah Doolittle, Metode Gauss-Seidel, Eliminasi Gauss, yang akan dikaji.

3. KAJIAN ALGORITMA

Untuk menganalisa permasalahan program PDP, dipergunakan pendekatan dengan membetuk kisi-kisi diskrit dimana jarak h sejajar dengan sumbu x dan sejajar sumbu y, dan sejajar sumbu z adalah sama.

Dengan : $x_i = ih_1$, $y_j = jh_2$, $z_k = kh_3$

$h_1 = \frac{q}{nx+1} \longrightarrow$ *nx* = banyaknya titik pada tiap baris sumbu x

$h_2 = \frac{b}{ny+1} \longrightarrow$ *ny* = banyaknya titik pada tiap baris sumbu y

$h_3 = \frac{c}{nz+1} \longrightarrow$ *nz* = banyaknya titik pada tiap baris sumbu z

Sehingga setiap titik $u_{ijk} \approx u(x_i,y_j,z_k)$, PDE dapat diwakili oleh bentuk persamaan linear

$$f(x_i, y_j, z_k) = \frac{u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}}{{h_1}^2} + \frac{u_{i,j+1,k} - 2u_{i,j,k} + u_{i,j-1,k}}{{h_2}^2} + \frac{u_{i,j,k-1} - 2u_{i,j,k} + u_{i,j,k+1}}{{h_3}^2}$$

$= -2(h_1^2 + h_2^2 + h_3^3)u_{i,j,k} + h_2^2h_3^2u_{i-1,j,k} + h_2^2h_3^2u_{i+1,j,k} + h_1^2h_3^2u_{i,j,k-1} + h_1^2h_2^2u_{i,j+1,k}$

$=(h_1^2h_2^2h_3^2)f(x_i,y_j,z_k)$

Di sini terlihat bahwa setiap $u_{i,j,k}$ selalu didukung oleh 6 titik tetangganya yaitu $u_{i-1,j,k}$, $u_{i+1,j,k}$, $u_{i,j-1,k}$, $u_{i,j+1,k}$, $u_{i,j,k+1}$, and $u_{i,j,k-1}$

Kemudian dibentuk Matrik A (matrik penthadiagonal) dengan elemen-elemen :

$$\begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix} \begin{bmatrix} v(1) \\ v(2) \\ . \\ . \\ v(nn) \end{bmatrix} = \begin{bmatrix} B(1) \\ B(2) \\ . \\ . \\ B(nn) \end{bmatrix}$$

A (label(j,i) , label(j,i)) = $-2(h_1^2 + h_2^2 + h_3^2)$

A (label(j,i) , label(j - 1,i)) = $h_1^2 h_2^2$

A (label(j,i) , label(j + 1,i)) = $h_1^2 h_3^2$

A (label(j,i) , label(j,i - 1)) = $h_2^2 h_1^2$

A (label(j,i) , label(j,i + 1)) = $h_2^2 h_3^2$

dimana label(j,i) = matrik yang elemen-elemennya berisi nomor label misal, label(j,i) = c, then v(c) is $u(x_i,y_j,z_k)$ at c.

Sehingga dapat tersusun bentuk matrik sebagai berikut :

Terlihat bahwa akan terbentuk nn seperti persamaan linear simultan :

$$A * \underline{v} = \underline{b}$$

3.1. Kajian Algoritma Gauss Seidel

```
function [v] = GSeidel(A,B);
global nx ny nz u;
nn = nx*ny*nz;
x = zeros(nn,1);
1.  for i=1:nn
2.      v(i) = B(i);
3.          for j=i:i-1
4.              v(i) = v(i) - A(i,j)*v(j);
5.          end;
6.         for j=i+1:nn
7.              v(i) = v(i) - A(i,j)*x(j);
8.          end;
9.       v(i) = v(i)/A(i,i);
10.  end;
```

Misal n = nn

1. Baris 3 hingga 5 memiliki maksimum kalang sebesar (n-1) sehingga worst case running time sebesar O(n).
2. Baris 6 hingga 8 memiliki maksimum kalang sebesar (n-1) sehingga worst case running time sebesar O(n).

3. Baris 1 hingga 10 memiliki kalang sebesar (n) sehingga waktu yang dibutuhkan sama dengan n
4. Worst case running time dari baris 1 sampai dengan baris 10 adalah :

$$\text{Total} = n.\left[(n-1)+(n-1)\right]$$
$$= n.\left[2n-2\right]$$
$$= 2n^2 - 2n$$
$$= O(n^2)$$

3.2. Kajian Algoritma Gauss

```
function [v] = Gauss(A,B);
global nx ny nz;
nn = nx*ny*nz;
1.  for j=1:nn-1
2.      for i=j+1:nn
3.         m(i,j)= A(i,j)/A(j,j);
4.          A(i,j)= 0;
5.          B(i)= B(i) - m(i,j)*B(j);
6.              for k=j+1:nn
7.                 A(i,k)= A(i,k) - m(i,j)*A(j,k);
8.              end;
9.       end;
10. end;

11. for i=nn:-1:1
12.     v(i)= B(i);
13.         for j=i+1:nn
14.          v(i)= v(i) - A(i,j)*v(j);
15. end;
16. v(i)= v(i)/A(i,i);
17. end;
```

Misal n = nn

1. Baris 6 hingga 8 memiliki maksimum kalang sebesar (n-1) sehingga worst case running time sebesar O(n).

2. Baris 2 hingga 9 memiliki maksimum kalang sebesar (n-1) sehingga worst case running time sebesar O(n).
3. Baris 1 hingga 10 memiliki maksimum kalang sebesar (n-1) sehingga worst case running time sebesar O(n).
4. Worst case running time dari baris 1 sampai dengan baris 10 adalah :

$$\text{Total} = \left[(n-1)(n-1)(n-1)\right]$$
$$= n^3 - 3n + 3n - 1$$
$$= O(n^3)$$

5. Baris 13 hingga 15 memiliki maksimum kalang sebesar (n-1) sehingga worst case running time sebesar O(n).
6. Baris 11 hingga 17 memiliki maksimum kalang sebesar (n) sehingga worst case running time sebesar O(n).
7. Worst case running time dari baris 1 sampai dengan baris 10 adalah :

$$\text{Total} = n(n-1)$$
$$= n^2 - n$$
$$= O(n^2)$$

8. Total Worst case running time dari baris 1 sampai dengan baris 17 adalah :

$$\text{Total} = \max(n^3, n^2)$$
$$= O(n^3)$$

3.3. Kajian Algoritma Doolittle

```
function [v] = dlitlle(A,B);
global nx ny nz;
nn = nx*ny*nz;
% Triangularization ----
1. for j=1:nn-1
2.   for i=j+1:nn
3.       A(i,j) = A(i,j)/A(j,j);
4.          for k=j+1:nn
5.         A(i,k) = A(i,k) - A(i,j)*A(j,k);
6.         end;
7.    end;
8.  end;
```

Misal n = nn

1. Baris 4 hingga 6 memiliki maksimum kalang sebesar (n-1) sehingga worst case running time sebesar O(n).

2. Baris 2 hingga 7 memiliki maksimum kalang sebesar (n-1) sehingga worst case running time sebesar O(n).
3. Baris 3 hingga 8 memiliki maksimum kalang sebesar (n-1) sehingga worst case running time sebesar O(n).
4. Worst case running time dari baris 1 sampai dengan baris 10 adalah :

Total = $\left[(n-1)(n-1)(n-1)\right]$

= $n^3 - 3n + 3n - 1$

= $O(n^3)$

```
% Forward Elimination ----
1. for i=1:nn
2.   z(i) = B(i);
3.     for j=1:i-1
4.       z(i) = z(i) - A(i,j)*z(j);
5.     end;
6. end;
```

Misal n = nn

1. Baris 3 hingga 5 memiliki maksimum kalang sebesar (n-1) sehingga worst case running time sebesar O(n).
2. Baris 1 hingga 6 memiliki maksimum kalang sebesar (n) sehingga worst case running time sebesar O(n).
3. Worst case running time dari baris 1 sampai dengan baris 6 adalah :

Total = $\left[(n-1)n\right]$

= $n^2 - n$

= $O(n^2)$

```
% Back Substitution ----
1. for i=nn:-1:1
2.     v(i) = z(i);
3.       for j=i+1:nn
4.           v(i) = v(i) - A(i,j)*v(j);
5.       end;
6.     v(i) = v(i)/A(i,i);
7. end;
```

Misal n = nn

1. Baris 3 hingga 5 memiliki maksimum kalang sebesar (n-1) sehingga worst case running time sebesar O(n).
2. Baris 1 hingga 7 memiliki maksimum kalang sebesar (n-1) sehingga worst case running time sebesar O(n).
3. Worst case running time dari baris 1 sampai dengan baris 7 adalah :

$$\text{Total} = \left[(n-1)(n-1)\right]$$
$$= n^2 - 2n + 1$$
$$= O(n^2)$$

Jumlah seluruh worst case running time untuk program di atas adalah :

$$T(n) = \max(n^3, n^2, n^2)$$
$$= O(n^3)$$

4. HASIL DAN KESIMPULAN

Metoda	2x2x2		3x3x3		4x4x4		5x5x5	
	Flops	Time	Flops	Time	Flops	Time	Flops	Time
Gauss-Seidel	115	0,6	4.226	0,6	12.199	1,0	134.887	93,2
Gauss	1.547	0,7	18.032	1,7	190.951	13,7	1.348.750	90,9
Doolittle	1.555	0.27	18.059	1,7	191.015	13,3	1.348.870	93,2

Flops : adalah jumlah proses komputasi pada eksekusi pada program Matlab.

Time : adalah jumlah waktu yang diperlukan untuk eksekusi.

Dari hasil esekusi pada program MATLAB dapat ditarik kesimpulan bahwa secara umum metoda Gauss-Seidel merupakan algoritma komputasi numeris yang memberikan waktu eksekusi terbaik.

5. DAFTAR PUSTAKA

Chapra SC, Canale RR, 1990, *Numerical Methode for Engineers*, McGraw Hill, New York

Etter DM, 1993, *Engineering Problem Solving with Matlab*, Prentice Hall, New Jersey

Kingston JH, 1991, *Algorithms and Data Structures*, Addison-Wesley Publishing Co, Sydney

Penny J, Lindfield, 1995, *Numerical Methods Using Matlab*, Ellis Horwood, New York