Jurnal Ilmiah KURSOR
Menuju Solusi Teknologi Informasi

Vol. 5, No. 1, Januari 2009 **ISSN 0216 - 0544**

ANALISA PEMANFAATAN FUNGSI PENALTI PADA KOMPUTASI PENYELESAIAN PERMASALAHAN OPTIMASI NONLINIER

Victor Hariadi
Jurusan Teknik Informatika Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember (ITS) Surabaya
Gedung Teknik Informatika, Kampus ITS Jl. Raya ITS, Sukolilo, Surabaya, 60111
E-Mail: victor@its-sby.edu

Abstract

The basis concept of penalty function at the optimum solving problem is changing the constrained problems to the unsconstrained problems by adding a penalty parameter to the objective function of the problems. The addition of penalty parameter will result in increasing the objective function as the limitation is approached. This research conducts an application of solving problem on nonlinear programmer (one of solving problems form) without function penalty inside. The penalty function chosen is Barrier Function, with its variants. Then, this research also conducts an analysis to the solving problems of nonlinear programmer. The research result shows that application function of Barrier Adaptive which is combined with Interior-Point Algorithm has a better result and time computation than other Barrier functions.

Key words: Nonlinear Programmer problems, Penalty Function, Barrier Function, Interior-Point Algorithm.

PENDAHULUAN

Permasalahan optimasi sering digunakan untuk mendapatkan suatu solusi yang ideal atau optimal dari permasalahan yang bersifat *linier* atau *nonlinier*. Pada pemrograman *nonlinier* mempunyai area yang lebih luas untuk dilakukan suatu penelitian, mengingat model-model *nonlinier* seringkali memiliki bentuk yang lebih kompleks dan dinamis. Pembagian pemrograman nonlinier dapat ditentukan dari bentuk fungsi tujuan/obyektif, dari karakteristik fungsi tujuan/obyektif, atau dari keberadaan dan bentuk fungsi-fungsi pembatasnya [1].

Sebagian besar dari permasalahan optimasi mempunyai suatu pembatas. Tetapi tidak ada perhitungan algoritma efektif yang tersedia untuk optimasi permasalahan nonlinear dengan pembatas, tetapi terdapat teknik baik untuk permasalahan optimasi tanpa pembatas. Maka digunakan metode penalti untuk mengubah kesulitan dari pemrograman *nonlinier* itu.

Metode penalti menggantikan suatu batasan permasalahan optimasi dengan suatu rangkaian permasalahan tanpa pembatas, dimana solusi tersebut harus memusat ke suatu solusi dari permasalahan dengan pembatas yang asli. Misalnya pada kasus minimasi dengan batasan pertidaksamaan, permasalahan minimasi tersebut dibentuk dengan menambahkan suatu penalti ke fungsi obyektif. Penalti tersebut akan digunakan ketika batasan dilanggar dan penalti tersebut menjadi 0 di daerah dimana batasan tidak dilanggar. Bentuk penalti umumnya terdiri dari suatu koefisien penalti dan fungsi penalti.

Penggunaan dari parameter penalti diperkenalkan pada model batasan dari sistem struktural untuk tujuan menghitung frekuensi alami yang menggunakan metode Rayleigh-Ritz [2]. Untuk permasalahan seperti itu, tanda dari kesalahan yang berkaitan dengan pelanggaran kondisi-kondisi pembatas tergantung pada tanda dari koefisien penalti itu. Dari sini, telah ditunjukkan bahwa kesalahan dalam kaitan dengan pelanggaran dari batasan dengan menggunakan metode penalti dapat ditentukan dan dikendalikan dengan menggunakan suatu kombinasi dari parameter penalti.

Pencarian solusi optimal untuk pemrograman *nonlinier* khususnya untuk pemrograman kuadratik memerlukan fungsi penalti yang digunakan untuk menangani pembatas dari permasalahan nonlinier [3]. Untuk mencari solusi dari permasalahan pemrograman *nonlinier* ini akan digunakan algoritma *interior-point* yang dipadukan dengan penggunaan fungsi penalti yang berupa fungsi *barrier* yang bertujuan untuk membatasi pergerakan nilai optimal yang dihasilkan pada setiap iterasinya dengan menjauhi batasan menuju interior dari ruang solusi [3][4]. Oleh karena itu dalam penelitian ini akan membahas tentang pemanfaatan fungsi penalti yang salah satunya adalah fungsi *barrier* pada komputasi penyelesaian permasalahan pemrograman *nonlinier*.

PEMROGRAMAN KUADRATIK

Permasalahan pada pemrograman kuadratik mempunyai bentuk yang berbeda dengan permasalahan pada pemrograman linier. Perbedaannya adalah pada fungsi tujuannya (*objective function*), dimana pada pemrograman kuadratik terdapat xi^2 dan $xixj$ ($i \neq j$). Kondisi yang harus dipenuhi untuk pemrograman kuadratik adalah bahwa fungsi obyektif berbentuk nonlinear, dimana didalamnya terdapat bentuk linear dan kuadrat serta semua pembatasnya berbentuk linier.

Bentuk umum pemrograman kuadratik adalah sebagai berikut [5][6][7]:

Minimize : $f(x) = \frac{1}{2} x^T Q x + c^T x$ (1)

Subject to : $Ax = b$ *dan* $x \geq 0$

Dimana $A \in R^{mXn}, b \in R^m, c \in R^n$ *dan* $H \in R^{nXn}$ yang simetrik dan *positive semidefinite*.

Kondisi *Convex*

Sebuah permasalahan harus memenuhi syarat konveksitas agar dapat diselesaikan dan mempunyai solusi yang optimal. Berikut ini akan diberikan penjelasan mengenai syarat konveksitas yang harus dipenuhi, yaitu bagaimana kondisi *convex* secara umum dan hubungan defferensial (*derivatives*) dengan kondisi *convex* (*convexity*).

Kondisi *Convex (convexcity)* secara umum

Sebuah permasalahan mempunyai nilai solusi optimalnya jika memenuhi beberapa persyaratan, antara lain [8]:

- Fungsi *Objective*nya harus merupakan fungsi *convex*.
- Ruang solusinya harus *convex set*.

Differential (Derivative) dan *Convexity*

Jika suatu fungsi satu dimensi (*one-dimensional*) f dapat dilakukan dua kali penurunan (*derivative*) secara berkesinambungan, maka pengujian kondisi *convex* dapat dilakukan dengan mudah. Sebuah fungsi memenuhi kondisi *convex* jika dan hanya jika:

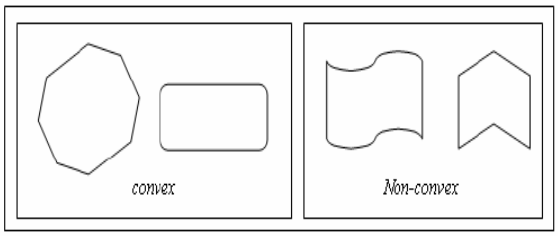
$$f''(x) \geq 0 \text{ untuk semua } x \in S \qquad (2)$$

Kondisi *KKT*

Kondisi KKT adalah kondisi yang diperlukan bagi penyelesaian permasalahan optimasi nonlinier. Dan menjadi syarat cukup bagi permasalahan pemrograman kuadratik [9]. Jaminan akan diperoleh solusi optimal jika kondisi KKT terpenuhi.

Rumusan KKT merupakan generalisasi fungsi Lagrange [1][9]:

$$L(x, \mu) = \frac{1}{2} x^T Q x + c^T x - \mu(Ax - b) \qquad (3)$$



Gambar 1. *Convex* dan *Non convex set*.

Pemrograman kuadratik terpenuhi kondisi KKT nya jika memenuhi beberapa kondisi:

1. $\frac{\partial L}{\partial x} \geq 0 \qquad c + x^T Q + \mu A \geq 0$

$$(c + x^T Q + \mu A + z = 0)$$

2. $x\frac{\partial L}{\partial x}=0 \qquad x^T\left(c+x^TQ+\mu A\right)=0$
3. $\frac{\partial L}{\partial \mu}\le 0 \qquad Ax-b\le 0$

   $(Ax-b+v=0)$
4. $\mu g(x)=0 \quad \mu(Ax-b)=0$
5. $x\ge 0 \quad x\ge 0$
6. $\mu\ge 0 \quad x\ge 0\mu$ (4)

Dari dua persamaan awal, untuk menyederhanakannya dengan memberikan nilai x = 0 atau z = 0, dari Persamaan (3) dan (4) digunakan pertimbangan $Ax-b=0$, jika μ = 0 maka persamaan yang digunakan adalah $Ax-b+v=0$. Sehingga kondisi KKT terpenuhi jika $x^Tz=0$. Jadi persamaan kondisi KKT yang digunakan adalah:

$$c+x^TQ+\mu A+z=0$$
$$Ax-b=0$$
$$x^Tz=0 \qquad (5)$$

Selanjutnya variabel μ akan diganti dengan variabel *y* (untuk memudahkan implementasi metode *primal-dual*).

Bentuk *Primal* dan *Dual*

Permasalahan asli dapat disebut sebagai bentuk *primal*. Dan setiap bentuk primal pasti akan memiliki ”bayangan” yang disebut bentuk *dual*.

Penyelesaian permasalahan optimasi dapat dilakukan tanpa melibatkan bentuk *dual*. Namun dalam banyak kasus, justru kehadiran *dual* dapat mempermudah penyelesaian masalah (karena bentuk *dual* biasanya lebih sederhana daripada *primal*-nya).

Solusi optimal dari permasalahan optimasi telah diperoleh jika solusi *dual* menghasilkan nilai yang sama dengan solusi *primal* [5].

Pembuatan *dual* untuk permasalahan *primal linier* dan *nonlinier* dapat mudah dilakukan. Masing-masing memiliki cara tersendiri.

Dampak hubungan antara *primal* dan *dual* untuk perhitungan dan teori:

1. Umumnya permasalahan dalam bentuk *dual* lebih mudah untuk diselesaikan, sehingga jika solusi optimal bagi permasalahan *dual* diketahui, maka solusi optimal dari permasalahan *primal* dapat diperoleh dengan mudah.
2. Karena *Lagrange multiplier* merupakan variabel *dual*, maka jika kondisi optimal dari *Lagrange multiplier* diketahui, akan mempermudah penyelesaian permasalahan optimasi.
3. Solusi optimal untuk fungsi obyektif akan diperoleh jika nilai fungsi obyektif *primal* dan *dual* adalah sama (walaupun pada banyak kasus kesamaan ini tidak diikuti oleh nila solusi optimal untuk masing-masing *primal* dan *dual*).

Fungsi obyektif bentuk *primal*:

$$ZP=c^Tx+\frac{1}{2}x^TQx \qquad (6)$$

Fungsi pembatas bentuk *primal*:

$$Ax-b \qquad (7)$$

Fungsi obyektif bentuk *dual*:

$$ZD=b^Ty-\frac{1}{2}x^TQx \qquad (8)$$

Fungsi pembatas bentuk *dual*:

$$A^Ty+z-Qx-c \qquad (9)$$

Fungsi Penalti

Prinsip dasar fungsi penalti adalah mengubah suatu permasalahan yang dibatasi (*constrained*) menjadi suatu permasalahan yang tidak dibatasi (*unconstrained*) dengan menambahkan suatu parameter penalti ke dalam fungsi obyektif. Dengan menambahkan parameter penalti maka akan meningkatkan nilai dari fungsi obyektif ketika batasan dilanggar atau bahkan ketika batasan didekati. Permasalahan yang tidak dibatasi dapat dipecahkan dengan menyesuaikan parameter penalti sehingga pemusatan/*convergence* dapat dicapai.

Terdapat tiga tingkatan fungsi penalti:

1. Metode *barrier* dimana tidak perlu mempertimbangkan solusi *infeasible* (tidak mungkin).
2. Fungsi penalti parsial (sebagian) dimana penalti diterapkan dekat batas kelayakan.
3. Fungsi penalti global (keseluruhan) yang diterapkan di seluruh daerah *infeasible.*

Fungsi *Barrier*

Fungsi *barrier* digunakan untuk membatasi pergerakan fungsi obyektif agar tidak keluar dari daerah kelayakan.

Bentuk umum fungsi *barrier* adalah [8][10]:

$\beta(x,y,\mu)$ = fungsi *lagrange* + *barrier term*

Bentuk umum fungsi logaritmik barrier:

$$\beta(x,y,\mu)=\frac{1}{2}x^TQx+c^Tx-\mu\sum\log x-y(Ax-b) \quad (10)$$

Bentuk umum fungsi adaptif *barrier*:

$$\beta(x,y,\mu)=\frac{1}{2}x^TQx+c^Tx-\mu\sum x_n\log x-y(Ax-b)^{\mathrm{X_n}} \quad (11)$$

adalah variabel penalti dan selanjutnya akan ditulis sebagai *pe*.

Bentuk umum fungsi *barrier* yang dimodifkasi:

$$\beta(x,y,\mu)=\frac{1}{2}x^TQx+c^Tx-\mu\sum_j\lambda\ln\left(\frac{g(x)}{\mu}+1\right)-y(Ax-b) \quad (12)$$

Algoritma *Interior-Point*

Pencarian solusi optimal dengan menggunakan algoritma *Interior-Point* tidak membutuhkan pencarian *extreme point* [3]. Algoritma ini mempercepat pencarian solusi optimal dengan melakukan pemotongan pencarian dari bagian interior daerah solusi menuju titik solusi optimal. Dimana dimulai dari titik awal pencarian, kemudian dengan bantuan *move direction* ditemukan titik baru yang lebih mendekati titik solusi optimal.

Algoritma *interior-point* dalam pencarian solusi optimal menerapkan konsep dasar seperti berikut:

1. Dimulai dari pemilihan sembarang titik awal yang berada di daerah *interior* dari ruang solusi, dimana titik ini akan menjadi titik awal pencarian titik optimal (*starting point*).
2. Perpindahan titik tersebut diarahkan oleh *move direction*, dimana dapat meningkatkan fungsi nilai dari tujuan atau lebih cepat tercapainya titik solusi optimal.
3. Melakukan transformasi dari ruang solusi yang layak untuk meletakkan titik-titik solusi yang telah ditemukan.
4. Perubahan (*update*) terhadap nilai solusi untuk pencarian titik solusi baru.

Konsep tersebut dapat pula divisualisasikan seperti pada Gambar 2.

APLIKASI ALGORITMA *INTERIOR-POINT*

Langkah-langkah dalam aplikasi algortima *Interior Point*:

1. Pemilihan titik awal pencarian solusi (*starting point*), dimana titik tersebut harus berada di daerah interior pada ruang solusi yang layak dengan nilai yang layak untuk bentuk primal $x^{(0)} > 0$ dan dual $(y^{(0)}, z^{(0)})$ dengan $z^{(0)} > 0$.
2. Penentuan *move direction*.
   Setiap variabel yang digunakan pada persamaan fungsi *barrier* ditambah dengan move direction (Δ), sehingga menjadi:
   (x + Δx), (y + Δy),dan (z + Δz)
   Move direction akan ditentukan berdasarkan fungsi *barrier* yang akan digunakan:

   1. Tanpa fungsi *barrier*
   $$\Delta y=\left(A(Z+XQ)^{-1}XA^T\right)^{-1}A(Z+XQ)^{-1}Xz$$
   $$\Delta x=(Z+XQ)^{-1}\left(XA^T\Delta y-Xz\right)$$
   $$\Delta z=Q\Delta x-A^T\Delta y \quad (13)$$

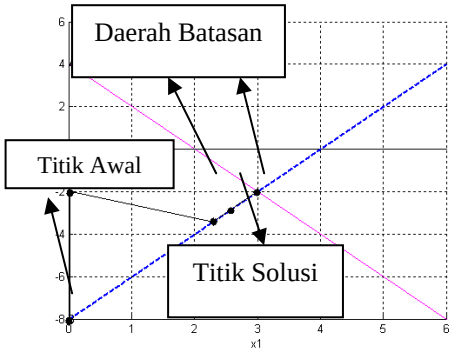
   2. Logaritmik *barrier*
   $$\Delta y=\left(A(Z+XQ)^{-1}XA^T\right)^{-1}A(Z+XQ)^{-1}(\mu e-XZe)$$
   $$\Delta x=(Z+XQ)^{-1}\left(XA^T\Delta y+(\mu e-XZe)\right)$$
   $$\Delta z=Q\Delta x-A^T\Delta y \quad (14)$$

   3. Adaptif barrier
   $$\Delta y=\left(A(Z+XQ)^{-1}XA^T\right)^{-1}A(Z+XQ)^{-1}(\mu epe-XZe)$$
   $$\Delta x=(Z+XQ)^{-1}\left(XA^T\Delta y+(\mu epe-XZe)\right)$$
   $$\Delta z=Q\Delta x-A^T\Delta y \quad (15)$$

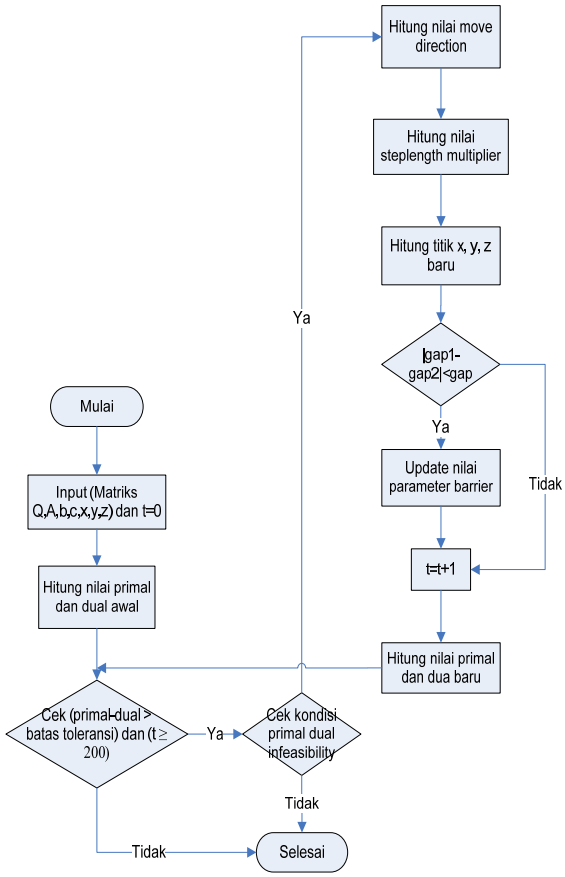
   4. Fungsi *barrier* yang dimodifikasi
   $$\Delta y=-\left(A(Z+XQ)^{-1}XA^T\right)^{-1}\left(A(Z+XQ)^{-1}\left(\lambda\frac{A}{(\mu(Ax-b)+1)}X-Xz\right)\right)$$
   $$\Delta x=(Z+XQ)^{-1}\left(XA^T\Delta y+\left(\lambda\frac{A}{(\mu(Ax-b)+1)}X-Xz\right)\right)$$
   $$\Delta z=Q\Delta x-A^T\Delta y \quad (16)$$

   5. Penentuan *steplength multiplier*.
   Steplength multiplier diperlukan untuk menjaga agar nilai *(x,z)* tetap positif, karena merupakan variabel *slack*.
   Steplength multiplier ditentukan dengan menggunakan persamaan :
   $$\alpha_x=\beta\left[\max(1,-\Delta x/x)\right]^{-1}$$
   $$\alpha_z=\beta\left[\max(1,-\Delta z/z)\right]^{-1} \quad (17)$$

Rentang nilai β antara 0 dan 1; $\alpha_x=\alpha_y=\alpha_z=\alpha=\min(\alpha_x,\alpha_z)$.



Gambar 2. Pencarian Solusi dengan Menggunakan Metode *Interior Point*.



Gambar 3. Alur Penyelesaian Permasalahan Optimasi *Nonlinier* dengan Algoritma *Interior-point*.

6. *Update* variabel *x, y, z*
   Variabel *x, y,* dan *z* yang digunakan pada perhitungan selanjutnya adalah :
   $x = x + \alpha\Delta x$
   $y = y + \alpha\Delta y$
   $z = y + \alpha\Delta z$ (18)
7. Menghitung nilai fungsi obyektif primal (*ZP*) dan dual (*ZD*) yang baru.

$$ZP = c^T x + \frac{1}{2} x^T Qx$$

$$ZD = b^T y - \frac{1}{2} x^T Qx \quad (19)$$

8. Menghitung *gap*.
   $gap = (ZP - ZD) / n$ (dihitung di awal iterasi)
   $gap\_1_i = (ZP_{i-1} - ZD_{i-1})$
   $gap\_2_i = (ZP_i - ZD_i)$ (20)
   Jika $|gap\_1 - gap\_2| < gap$, maka menghitung nilai *pe* yang baru & iterasi dilanjutkan. Iterasi berhenti jika *jumlah_iterasi* = 200 atau $gap <= 10^{-6}$.

HASIL DAN PEMBAHASAN

Dalam proses uji coba sebagai masukan digunakan permasalahan optimasi *nonlinier* yang telah termodelkan ke dalam bentuk persamaan/fungsi obyektif dengan fungsi-fungsi pembatasnya. Dan kemudian fungsi-fungsi di atas ditulis dalam format matriks (untuk memudahkan entri ke dalam aplikasi Matlab yang telah dibuat) [11].

Aplikasi Matlab untuk penyelesaian permasalahan optimasi dibuat dengan empat varian, yaitu tanpa menggunakan fungsi penalti, dengan fungsi penalti logaritmik *barrier*, *adaptif barrier*, serta fungsi *barrier* yang telah dimodifikasi [12].

Format/bentuk masukan :

```
A = [1 0 0 1 0 0;0 1 0 0 1 0;0 0 1 0 0 1];
c  = [-15 -12 -9 0 0 0]';
b  = [1 1 1]';
Q = [18 0 0 0 0 0;0 18 0 0 0 0;0 0 18 0 0 0;0
     0 0 0 0 0;0 0 0 0 0 0;0 0 0 0 0 0];
x  = [0.0001 0.0001 0.5 0.9999 0.9999 0.5]';
y  = [-14.9983  -11.9983  -0.0001]';
z  = [0.0001 0.0001 0.0001 14.9983 11.9983
     0.0001]';
```

Hasil Uji Coba Tanpa Fungsi *Barrier*

Gambar 3 menunjukkan bahwa nilai optimal komputasi untuk bentuk *primal* dan *dual* mulai menuju titik konvergensi pada iterasi ke-10. Dan semakin menajam pada sekitar iterasi ke-20.

Pada solusi yang dihasilkan seperti dalam Tabel 1 dapat dilihat bahwa kedua nilai optimasi *primal* dan *dual* telah mendekati

konvergensi pada nilai -12.50000 dengan *gap* < 10^{-6}.

Hasil Uji Coba dengan Fungsi Logaritmik *Barrier*

Gambar 4 menunjukkan bahwa nilai optimal komputasi untuk bentuk primal cenderung stabil dan konvergensi diperoleh melalui mendekatnya nilai dual (walaupun pada awal-awal iterasi sempat menunjukkan perilaku divergensi) pada nilai primal. Titik konvergensi mulai terlihat pada iterasi ke-10, dan semakin menajam pada sekitar iterasi ke-12.

Pada solusi yang dihasilkan dalam Tabel 2 dapat dilihat bahwa kedua nilai optimasi *primal* dan *dual* telah mendekati konvergensi pada nilai -12.50000 dengan *gap* < 10^{-6}.

Hasil Uji Coba dengan Fungsi *Adaptif Barrier*

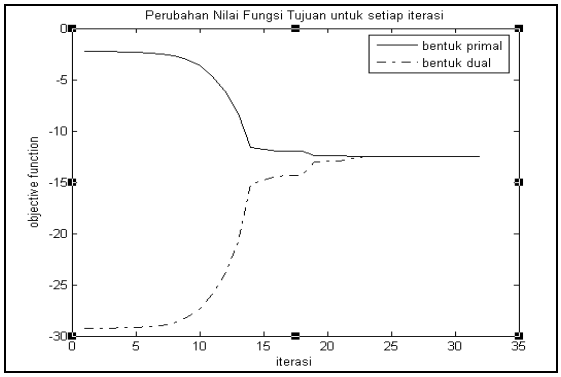
Gambar 5 menunjukkan bahwa nilai optimal komputasi untuk bentuk *primal* cenderung stabil dan konvergensi diperoleh melalui mendekatnya nilai *dual* (walaupun pada awal-awal iterasi sempat menunjukkan perilaku divergensi) pada nilai primal. Titik konvergensi mulai terlihat pada iterasi ke-4, dan semakin menajam pada sekitar iterasi ke-5.
Di sini terlihat sebuah perbedaan yang cukup signifikan dibanding dua uji coba sebelumnya, dan perilaku ini juga muncul pada kasus-kasus uji coba yang lain.

Pada solusi yang dihasilkan seperti dalam Tabel 3 dapat dilihat bahwa kedua nilai optimasi *primal* dan *dual* telah mendekati konvergensi pada nilai -12.50000 dengan *gap* < 10^{-6}.

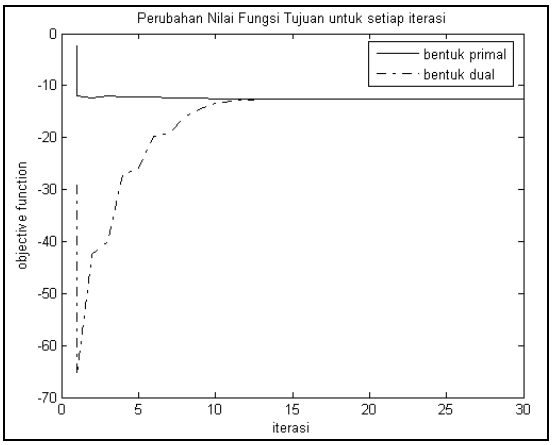
Hasil Uji Coba dengan Fungsi *Barrier* yang Dimodifikasi

Gambar 6 menunjukkan bahwa nilai optimal komputasi untuk bentuk *primal* cenderung stabil dan konvergensi diperoleh melalui mendekatnya nilai dual pada nilai primal. Titik konvergensi mulai terlihat pada iterasi ke-3, dan semakin menajam pada sekitar iterasi ke-5.
Namun pola solusi dengan fungsi *Barrier* yang dimodifikasi ternyata tidak stabil. Ini dapat dilihat pada pola solusi untuk kasus-kasus lain. Pada beberapa kasus, pemakaian fungsi *Barrier* yang dimodifikasi ternyata memberikan solusi dengan jumlah iterasi yang justru lebih banyak ketimbang fungsi logaritmik *Barrier.*

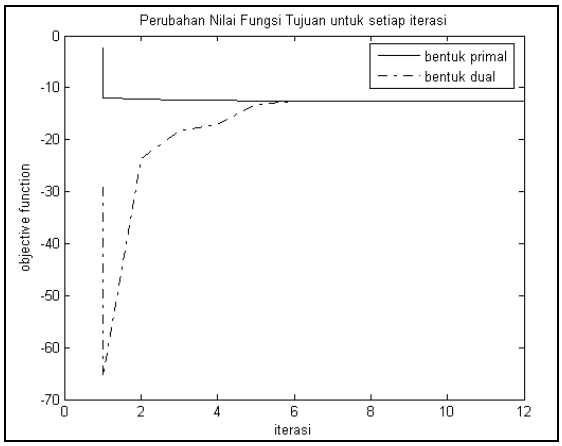
Solusi yang dihasilkan pada Tabel 4 dapat dilihat bahwa kedua nilai optimasi *primal* dan *dual* telah mendekati konvergensi pada nilai -12.50000 dengan *gap* < 10^{-6}.
Sebagai catatan, grafik-grafik pada gambar merupakan salah satu keluaran dari aplikasi yang dikembangkan. Dan grafik tersebut memberikan informasi mengenai proses konvergensi antara nilai optimal dari persamaan *primal* dan *dual*. Karena seperti diketahui bahwa tolak ukur optimalitas dari penyelesaian permasalahan optimasi *nonlinier* salah satunya adalah dengan tercapainya konvergensi antara nilai *primal* dan *dual* [5].



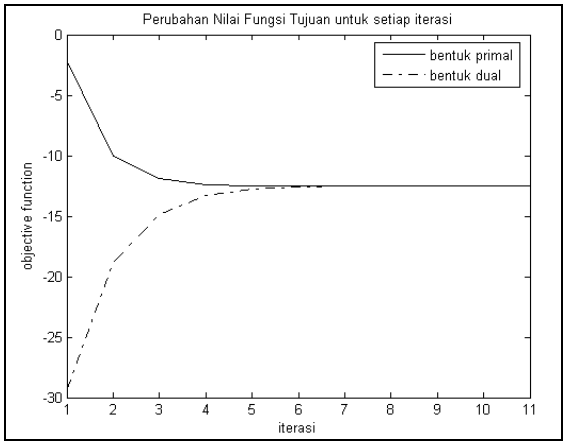
Gambar 3. Grafik Perubahan Nilai Fungsi Obyektif.



Gambar 4. Grafik Perubahan Nilai Fungsi Tujuan.



Gambar 5. Grafik Perubahan Nilai Fungsi Tujuan.



Gambar 6. Grafik Perubahan Nilai Fungsi Tujuan.

Tabel 1. Ringkasan Hasil Komputasi tanpa Menggunakan Fungsi *Barrier* pada Iterasi Ke-32.

iterasiOptimal = 32			
*x*Optimal =	*y*Optimal =	*z*Optimal =	*ZP*Optimal=
0.8333	1.0e-006 *	1.0e-006 *	-12.5000
0.6667			
0.5000	-0.8350	0.0000	*ZD*Optimal=
0.1667	-0.1171	0.0000	-12.5000
0.3333	-0.0000	0.0000	
0.5000		0.8350	*gap*Optimal=
		0.1171	1.7818e-007
		0.0000	
Waktu = 0.500000 seconds			

Tabel 2. Ringkasan Hasil Komputasi tanpa Menggunakan Fungsi *Barrier* pada Iterasi Ke-30.

iterasiOptimal = 30			
*x*Optimal =	*y*Optimal =	*z*Optimal=	
0.8333	1.0e-006 *	1.0e-006 *	*ZP*Optimal=
0.6667			-12.5000
0.5000	-0.9050	0.1810	
0.1667	-0.4525	0.2263	*ZD*Optimal=
0.3333	-0.3017	0.3017	-12.5000
0.5000		0.9050	
		0.4525	
		0.3017	*gap*Optimal=
			9.0504e-007
Waktu = 0.406000 seconds			

Tabel 3. Ringkasan Hasil Komputasi tanpa Menggunakan Fungsi *Barrier* pada Iterasi Ke-12.

iterasiOptimal = 12			
*x*Optimal =	*y*Optimal =	*z*Optimal =	*ZP*Optimal =
0.8333	1.0e-006 *	1.0e-006 *	-12.5000
0.6667			*ZD*Optimal =
0.5000			
0.1667	-0.1751	0.0149	-12.5000
0.3333	-0.0977	0.0205	
0.5000	-0.0544	0.0100	*gap*Optimal=
		0.1751	1.2009e-007
		0.0977	
		0.0544	
Waktu = 0.297000 seconds			

Tabel 4. Ringkasan Hasil Komputasi tanpa Menggunakan Fungsi *Barrier* pada Iterasi Ke-11.

iterasiOptimal = 11			
*x*Optimal =	*y*Optimal =	*z*Optimal =	*ZP*Optimal=
0.8333	1.0e-005 *	1.0e-005 *	-12.5000
0.6667			
0.5000	-0.2324	0.0138	*ZD*Optimal=
0.1667	-0.0364	0.0149	-12.5000
0.3333	-0.0158	0.0159	
0.5000		0.2324	*gap*Optimal=
		0.0364	8.8094e-007
		0.0158	
Waktu = 0.312000 seconds			

SIMPULAN

Berdasarkan aplikasi yang telah dibuat beserta uji coba yang telah dilakukan, maka dapat ditarik simpulan sebagai berikut:

1. Algoritma *interior point* dengan menggunakan fungsi *barrier* maupun tanpa menggunakan fungsi *barrier* dapat digunakan untuk menyelesaikan permasalahan pemrograman kuadratik.
2. Secara umum penggunaan algoritma *interior point* dengan fungsi *adaptif barrier* akan mendapatkan hasil yang lebih baik dibanding fungsi *barrier* yang lain karena mempunyai jumlah iterasi yang lebih kecil. Sedangkan untuk penggunaan logaritmik barrier secara umum memberikan hasil yang kurang baik karena jumlah iterasinya selalu lebih besar dibandingkan dengan fungsi *barrier* yang lain.

DAFTAR PUSTAKA

[1] Nash SG and Sofer A. *Linear and NonLinear Programming*. New York: The Mc. Graw-Hill Companies. 1994.

[2] Ilanko S and Dickinson SM. Asymptotic Modelling of Rigid Boundaries and Connections in the Rayleigh-Ritz Method. *Journal of Sound and Vibration*. 4: 59-66. 1999.

[3] Freud RM. *Penalty and Barrier Methods for Constrained Optimization.* Lecture Notes. Massachusetts: Massachusetts Institute of Technology. 2004.

[4] Polyak R. Modified Barrier Function (Theory and Method). *Math Programming*. 54: 177-222, 1992.

[5] Taha HA. *Operations Research: An Introduction.* Saddle River, NJ: Prentice Hall. 2003.

[6] Rardin RL. *Optimization in Operations Research*. New York: Prentice-Hall. Inc. 1998.

[7] Hiller FS and Lieberman GJ. *Introduction to Operations Research 8th edition.* New York: The Mc. Graw-Hill Companies Inc. 2005.

[8] Lange K. An Adaptive Barrier Method for Convex Programming. *Methods and Applications of Analysis*. 32: 392-402, 1994.

[9] Nocedal J and Wright SJ. *Numerical Optimization.* Beijing: Science Press. 2006.

[10] Vassiliadis VS. Application of the Modified Barrier Method in Large Scale Quadratic Programming Problem. *Comput. Chem. Eng.*, 20: S243-S248. 1996.

[11] Venkataraman P. *Applied Optimization with MATLAB Programming*. New York: John Wiley & Sons. 2002.

[12] Smith AE and Coit DW. *Penalty Functions.* Handbook of Evolutionary Computation. UK: IOP Publishing Ltd and Oxford University Press. 1997