PENERAPAN GABUNGAN METODE *ZERO CROSSING* DAN *VIRUS EVOLUTIONARY GENETIC ALGORITHM* (VEGA) PADA PENYELESAIAN PERSAMAAN NON-LINIER

SKRIPSI

Oleh

Zainul Anwar
NIM 121810101025

JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS JEMBER
2016

PENERAPAN GABUNGAN METODE *ZERO CROSSING* DAN *VIRUS EVOLUTIONARY GENETIC ALGORITHM* (VEGA) PADA PENYELESAIAN PERSAMAAN NON-LINIER

SKRIPSI

diajukan guna melengkapi tugas akhir dan memenuhi salah satu syarat untuk
menyelesaikan Program Studi Matematika (S1)
dan mencapai gelar Sarjana Sains

Oleh

Zainul Anwar
NIM 121810101025

JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS JEMBER
2016

PERSEMBAHAN

Skripsi ini saya persembahkan untuk:

1. Ayahanda Budiyono dan Ibunda Ma'idah tercinta yang senantiasa memberi doa, inspirasi, semangat dan kasih sayang;
2. Kakakku Edy Budi Santoso yang memberikan dukungan moril dan materil;
3. Seluruh guru dan dosen sejak taman kanak-kanak hingga perguruan tinggi yang telah memberi banyak ilmu dan membimbing dengan tulus;
4. Almamater Jurusan Matematika FMIPA Universitas Jember, SMA Negeri 2 Situbondo, SMP Negeri 1 Asembagus, SD Negeri 1 Sumberwaru dan TK Darmawanita Banyuputih;

MOTTO

”Sesungguhnya bersama kesulitan ada kemudahan. Maka apabila engkau telah selesai (dari suatu urusan), tetaplah bekerja keras untuk (urusan lain) dan hanya kepada Tuhanmu lah engkau berharap.”

(terjemahan Q.S. Al-Insyirah ayat 6-8)

PERNYATAAN

Saya yang bertanda tangan di bawah ini:

Nama : Zainul Anwar

NIM : 121810101025

menyatakan dengan sesungguhnya bahwa karya ilmiah yang berjudul ”Penerapan Gabungan Metode *Zero Crossing* dan *Virus Evolutionary Genetic Algorithm* Pada Penyelesaian Persamaan Non-Linier” adalah benar-benar hasil karya sendiri, kecuali kutipan yang sudah saya sebutkan sumbernya, belum pernah diajukan dalam institusi manapun dan juga bukan karya jiplakan. Saya bertanggung jawab atas keabsahan dan kebenaran isinya sesuai dengan sikap ilmiah yang harus dijunjung tinggi.

Demikian pernyataan ini saya buat dengan sebenarnya, tanpa ada tekanan dan paksaan dari pihak manapun serta bersedia mendapat sanksi akademik jika ternyata di kemudian hari pernyataan ini tidak benar.

Jember, Juni 2016

Yang menyatakan,

Zainul Anwar

NIM 121810101025

SKRIPSI

PENERAPAN GABUNGAN METODE *ZERO CROSSING* DAN *VIRUS EVOLUTIONARY GENETIC ALGORITHM* (VEGA) PADA PENYELESAIAN PERSAMAAN NON-LINIER

Oleh

Zainul Anwar

Pembimbing

Dosen Pembimbing Utama : M. Ziaul Arif, S.Si., M.Sc

Dosen Pembimbing Anggota : Ahmad Kamsyakawuni, S.Si., M.Kom

PENGESAHAN

Skripsi berjudul ” Penerapan Gabungan Metode *Zero Crossing* Dan *Virus Evolutionary Genetic Algorithm* (Vega) Pada Penyelesaian Persamaan Non-Linier” telah diuji dan disahkan pada

Hari, tanggal :

Tempat : Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Jember

Tim Penguji:

Ketua,	Sekretaris,
M. Ziaul Arif, S.Si., M.Sc	Ahmad Kamsyakawuni, S.Si., M.Kom
NIP. 198501112008121002	NIP. 197211291998021001
Anggota I,	Anggota II,
Drs. Rusli Hidayat, M.Sc	Kosala Dwidja Purnomo, S.Si., M.Si
NIP. 196610121993031001	NIP. 196908281998021001

Mengesahkan
Dekan,

Drs. Sujito, Ph.D
NIP.196102041987111001

RINGKASAN

Penerapan Gabungan Metode *Zero Crossing* dan *Virus Evolutionary Genetic Algorithm* (VEGA); Zainul Anwar; 121810101025; 2016; 59 halaman; Jurusan Matematika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Jember.

Persamaan non-linier adalah persamaan yang memiliki peubah dengan pangkat terkecil adalah 1 atau transenden. Solusi analitik adalah solusi terbaik dari persamaan non-linier, tetapi seringkali sulit didapatkan. Sehingga aproksimasi terhadap solusi eksak yang diperoleh melalui metode numerik layak dikatakan sebagai solusi. Selain metode numerik pada umumnya, persamaan non-linier saat ini juga dapat diselesaikan menggunakan metode optimasi seperti *Virus Evolutionary Genetic Algorithm* (VEGA). Tujuan dari penelitian ini adalah menyelesaikan persamaan non-linier menggunkan Gabungan Metode *Zero Crossing* dan *Virus Evolutionary Genetic Algorithm* (VEGA), serta membandingkan hasil yang diperoleh dengan hasil Metode Newton Raphson dan beberapa jurnal rujukan.

Permasalahan yang digunakan pada penelitian ini menggunakan beberapa persamaan dari jurnal rujukan. Terdapat 10 persamaan non-linier akar tunggal dan 5 persamaan non-linier akar ganda. Tujuan dari peneliti adalah mencari solusi persamaan non-linier dengan mencari nilai *fitness* $|f(x)|$ yang paling minimum dari diantara beberapa kandidat solusi.

Berdasarkan hasil dan pembahasan, diperoleh solusi yang mendekati dan sama dengan solusi eksak. Pada persamaan non-linier akar tunggal, Gabungan Metode *Zero Crossing* dan *Virus Evolutionary Genetic Algorithm* (VEGA) mampu memperoleh nilai *fitness* hingga $4.409e-016$ pada PNL 3 (akar tunggal). Begitu juga pada PNL 4 dan 7 (akar tunggal) juga menghasilkan solusi yang sama dengan solusi eksak seperti yang dihasilkan Metode Newton Raphson. Selain itu Gabungan Metode *Zero Crossing* dan *Virus Evolutionary Genetic Algorithm* (VEGA) mempunyai keunggulan yakni dapat menemukan 3 solusi pada PNL 10 (akar tunggal), hal ini lebih baik daripada Javidi (2007) yang telah

menemukan 2 solusi untuk persamaan yang sama. Pada persamaan non-linier akar ganda, Gabungan Metode *Zero Crossing* dan *Virus Evolutionary Genetic Algorithm* (VEGA) juga memberikan solusi yang layak dan lebih baik dari beberapa metode lain pada jurnal rujukan. Misalnya pada PNL 4 (akar ganda) memperoleh nilai *fitness* $|f(x)| = 0$, artinya solusi yang dihasilkan sama dengan solusi eksak. Di sisi lain, Gabungan Metode *Zero Crossing* dan *Virus Evolutionary Genetic Algorithm* (VEGA) memiliki kelemahan berupa waktu komputasi yang cukup lama dibandingkan metode numerik dan bergantung pada nilai parameter yang diinputkan. Pada penelitian ini, nilai parameter yang memberikan hasil optimal adalah *interinfection time* $= 3$, virus *pop size* $= 9$, probabilitas *crossover* (Pc)$= 0.8$, probabilitas mutasi (Pm)$= 0.1$ dan probabilitas infeksi virus (Pv)$= 0.4$. Kelemahan lain yakni tidak dapat menemukan jumlah akar ganda dari persamaan non-linier, tetapi hanya mampu menemukan solusi dari persamaan tersebut.

PRAKATA

Puji syukur ke hadirat Allah SWT yang telah melimpahkan rahmat, taufik dan karuniaNya sehingga skripsi yang berjudul ”Penerapan Gabungan Metode *Zero Crossing* dan *Virus Evolutionary Genetic Algorithm* (VEGA)” dapat terselesaikan. Skripsi ini disusun untuk memenuhi salah satu syarat dalam menyelesaikan pendidikan strata 1 (S1) di Jurusan Matematika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Jember. Sholawat dan salam semoga tercurahkan keharibaan Nabi Muhammad SAW yang telah menjadi pembawa rahmatan lil’alamin.

Penyusunan skripsi ini tidak terlepas dari bantuan berbagai pihak, baik secara langsung maupun tidak langsung. Oleh karena itu, penulis menyampaikan terima kasih kepada:

1. Drs. Sujito, Ph.D. selaku Dekan Fakultas MIPA Universitas Jember dan Kusbudiono S.Si., M.Si. selaku Ketua Jurusan Matematika Fakultas MIPA Universitas Jember yang telah memberikan fasilitas-fasilitas dalam tahap penelitian;
2. M. Ziaul Arif, S.Si., M.Sc. selaku Dosen Pembimbing Utama dan Ahmad Kamsyakawuni S.Si., M.Kom. selaku Dosen Pembimbing Anggota yang telah memberikan bimbingan dan bantuan untuk pengerjaan skripsi ini;
3. Drs. Rusli Hidayat, M.Sc. selaku Dosen Penguji I dan Kosala Dwidja Purnomo, S.Si., M.Si. selaku Dosen Penguji II yang telah memberikan kritik dan saran yang membangun untuk penyempurnaan skripsi ini;
4. Drs. Rusli Hidayat, M.Sc. selaku Dosen Pembimbing Akademik yang telah membimbing dalam pemilihan mata kuliah;
5. Seluruh dosen dan karyawan Jurusan Matematika Fakultas MIPA Universitas Jember yang telah memberikan ilmu serta membantu selama proses perkuliahan berlangsung;
6. Bapak Budiyono dan Ibu Ma’idah yang selalu memberi doa dan dukungan baik lahir maupun batin;
7. Kakak Edy Budi Santoso yang telah memberi semangat dan dukungan;

8. Halifatul Kamiliyah yang telah sabar dan memberi semangat, dukungan dan bantuan moral serta materi untuk pengerjaan skripsi ini;
9. Teman-teman Bathics'12 yang selalu memberikan dukungan dalam hal pembelajaran selama kuliah;
10. Kawan LPMM ALPHA serta seluruh LPM dalam naungan PPMI Kota Jember yang sudah memberikan wawasan dan menjadi teman diskusi;
11. Kawan-kawan CT yang telah memberi semangat dan dukungan moral dalam pengerjaan skripsi ini;
12. Semua pihak yang membantu terselesaikannya skripsi ini.

Penulis menyadari bahwa dalam menyusun skripsi ini masih terdapat kekurangan baik isi maupun susunannya. Oleh sebab itu, penulis mengharapkan saran dan kritik demi penyempurnaan skripsi ini. Akhirnya penulis berharap semoga skripsi ini dapat memberi manfaat dan sumbangan pengetahuan bagi pembaca.

DAFTAR ISI

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

Halaman

BAB 1. PENDAHULUAN

1.1 Latar Belakang

Berbagai disiplin ilmu terutama bidang sains dan terapannya sering kali berhadapan dengan masalah yang berkaitan dengan solusi persamaan non-linier. Persamaan non-linier adalah persamaan yang memiliki peubah dengan pangkat terkecil adalah 1 atau transenden. Secara singkat pencarian solusi persamaan non-linier adalah menentukan nilai x yang memenuhi persamaan $f(x) = 0$, yakni nilai $x = s$ sedemikian hingga $f(s)$ sama dengan nol.

Solusi analitik dari persamaan non-linier adalah solusi terbaik untuk menyelesaikan permasalahan. Namun pada kenyataannya penyelesaian analitik dari persamaan non-linier tidak mudah ditemukan, kecuali pada beberapa kasus sederhana. Tidak jarang akan menemui kesulitan ketika melakukan perhitungan analitik meskipun persamaannya tampak sederhana. Hal inilah yang mendasari perhitungan secara numerik diperlukan untuk mencari penyelesaian persamaan non-linier dan memecahkan persoalan-persoalan dalam berbagai disiplin ilmu.

Perhitungan secara numerik menawarkan hasil estimasi yang mendekati solusi analitik dengan galat yang kecil. Solusi numerik dihasilkan dari perhitungan secara berulang (iteratif) hingga tercapai kekonvergenan. Sampai saat ini banyak metode yang umum digunakan untuk mencari penyelesaian persamaan non-linier, misalnya Metode Biseksi, Regulasi Falsi, Newton-Raphson dan lain sebagainya. Masing-masing metode tersebut memiliki kelebihan dan kelemahan dalam hal perhitungan numerik. Namun metode Newton-Raphson lebih banyak digunakan untuk menyelesaikan persamaan non-linier karena tingkat kecepatan perhitungannya lebih baik dari metode lain. Di sisi lain Metode Newton-Raphson tidak dapat digunakan ketika titik pendekatan awalnya berada pada titik ekstrim atau titik puncak karena di titik ini nilai $f'(x) = 0$ sehingga nilai penyebut $\frac{f(x)}{f\prime(x)}$ sama dengan nol.

Saat ini mulai dikembangkan beberapa metode untuk mencari solusi persamaan non-linier. Arif (2013) menggunakan modifikasi metode Chebyshev

orde tiga untuk mencari akar ganda tanpa menggunakan turunan, output yang dihasilkan adalah nilai dan jumlah akar persamaan non-linier. Chun (2005) mengguunakan metode iteratif untuk memperbaiki Metode Newton dengan Metode Dekomposisi. Metode iteratif lain juga telah ditemukan, misalnya oleh Javidi (2007), Wang (2011), Noor (2007) Liang (2015) dan Shengguo (2009).

Selain metode yang telah disebutkan di atas, dikembangkan pula metode penyelesaian persamaan non-linier dengan teknik stokastik atau yang dikenal sebagai metode metaheuristik. Optimasi fungsi menjadi dasar pengembangan metode tersebut. Indrianingsih (2010) menyatakan optimasi fungsi berkaitan dengan masalah pencarian solusi atas suatu himpunan melalui proses analisis dengan beberapa kendala dan kombinasi kendala sesuai tujuan optimasi. Untuk mendapatkan solusi optimum dibutuhkan proses perhitungan yang panjang dan tidak praktis sehingga untuk memecahkan kasus ini dibutuhkan metode metaheuristik. Beberapa metode metaheuristik diantaranya algoritma genetika, *Particle Swarm Optimation* (PSO), *Cat Swarm Optimation* (CSO), *Virus Evolutionary Genetic Algorithm* (VEGA) dan lain sebagainya. Meskipun selama ini metode metaheuristik banyak diterapkan pada optimasi dan penjadwalan *Operation Research* (OR), namun tidak menutup kemungkinan juga bisa diterapkan untuk mencari penyelesaian persamaan non-linier. Yusuf dan Soesanto (2012) dalam jurnalnya menyatakan algoritma genetika mampu memberikan penyelesaian akar persamaan sebuah fungsi dengan hasil yang sama dengan solusi analitiknya.

Algoritma lain yang efektif dalam hal optimasi adalah *Virus Evolutionary Genetic Algorithm* (VEGA). Algoritma ini dihasilkan dari penggabungan Algoritma Genetika dan infeksi virus (Fukuda, 1999). Beberapa penelitian sebelumnya tentang VEGA memberikan hasil yang baik. Kubota dkk(1996), mengaplikasikan VEGA untuk tiga jenis permasalahan optimasi yakni TSP (*Traveling Salesman Problem*), *Knapsack* dan optimasi fungsi yang memberikan hasil bahwa VEGA lebih baik daripada SSGA (*Steady State Genetic Algorithm*). Penelitian Amulyo (2013) mengenai *Hybrid Virus Evolutionary Genetic Algorithm* (VEGA) dan *Simulated Annealing* (SA) pada penjadwalan *flowshop*

menghasilkan bahwa semakin besar nilai parameter yang diberikan, maka nilai *makespan* yang diperoleh semakin minimal. Penelitian lain oleh Fountas (2013) tentang pengaplikasian VEGA untuk optimasi proses mesin pemahat menyatakan bahwa VEGA mampu mendapatkan hasil yang global optima, sedangkan algoritma genetika terjebak pada pencarian lokal optima.

Selain keunggulan di atas, masih terdapat kendala jika diaplikasikan pada persamaan non-linier. Yusuf dan Soesanto (2012) mengemukakan bahwa jika suatu populasi yang dibangkitkan sangat kecil, maka suatu kromosom dengan gen-gen yang mengarah ke solusi akan sangat menyebar ke kromosom-kromosom lainnya. Dengan kata lain populasi awal yang dibangkitkan belum tentu mengarah ke solusi. Oleh sebab itu dibutuhkan suatu teknik atau metode yang menjamin di dalam interval terdapat suatu solusi.

Metode *Zero* Crossing adalah metode yang memanfaatkan perubahan tanda pada setiap ujung interval untuk menjamin adanya solusi pada interval tersebut. Metode ini sudah sering dipakai pada metode numerik interval tertutup terutama Metode Biseksi. Akibatnya metode numerik dengan interval tertutup akan konvergen menuju satu titik.

Berdasarkan uraian di atas, menjadi bahan pertimbangan bagi penulis untuk melakukan penelitian tentang penerapan gabungan metode *Zero Crossing* dan *Virus Evolution Genetic Algorithm* (VEGA) pada penyelesaian persamaan non-linier.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, maka permasalahan yang akan dibahas dalam penelitian ini adalah

a. Bagaimana menyelesaikan persamaan non-linier berupa polinomial dan fungsi transenden menggunakan gabungan Metode *Zero Crossing* dan *Virus Evolution Genetic Algorithm* (VEGA)?

b. Bagaimana hasil gabungan Metode *Zero Crossing* dan *Virus Evolution Genetic Algorithm* (VEGA) jika dibandingkan dengan Metode Newton-Raphson serta metode numerik pada jurnal rujukan?

1.3 Batasan Masalah

Pada penelitian ini menggunakan batasan masalah sebagai berikut:

a. persamaan non-linier yang digunakan hanya persamaan non-linier satu peubah berupa polinomial dan fungsi transenden;
b. fungsi transenden yang digunakan adalah fungsi kontinu dan berupa fungsi trigonometri, eksponensial, maupun gabungannya.

1.4 Tujuan

Tujuan dari penelitian ini adalah

a. mengetahui cara menyelesaikan persamaan non-linier dengan menggunakan gabungan Metode *Zero Crossing* dan *Virus Evolution Genetic Algorithm* (VEGA);
b. mengetahui hasil penyelesaian persamaan non-linier dengan menggunakan gabungan Metode *Zero Crossing* dan *Virus Evolution Genetic Algorithm* (VEGA) jika dibandingkan dengan Metode Newton-Raphson.

1.5 Manfaat

Manfaat dari penelitian ini adalah mampu memberikan informasi mengenai metode terbaik untuk mengestimasi penyelesaian persamaan non-linier dan dapat dijadikan pertimbangan pada saat pemilihan metode yang akan digunakan untuk menyelesaikan persamaan non-linier secara numerik. Serta memberikan wawasan tentang penerapan metode metaheuristik dalam permasalahan numerik dari sudut pandang masalah optimasi.

BAB 2. TINJAUAN PUSTAKA

2.1 Persamaan Non-linier

Persamaan non-linier adalah semua persamaan yang bukan persamaan linier dengan peubah berderajat terkecil sama dengan satu atau transenden dan apabila digambarkan bukan garis lurus. Beberapa fungsi yang termasuk persamaan non-linier adalah fungsi transenden seperti

$\sin 2x - sec^{-1}x = 5$

$I = 10e^{-t} \sin 2\pi t$

dan fungsi non transenden yang merupakan polinomial

$a_n x^n + a_{n-1}x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0 = 0.$

Penyelesaian persamaan non-linier adalah penentuan akar-akar persamaan non-linier, dengan menentukan nilai-nilai x yang menyebabkan $f(x) = 0$. Dengan kata lain, akar persamaan non-linier adalah titik potong $f(x)$ dengan sumbu X. Akar-akar persamaan non-linier dapat diperoleh secara analitis dan numerik (pendekatan). Dalam beberapa kasus sederhana metode analitis menjadi pilihan utama, misalnya pada polinomial kuadrat menggunakan rumus

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Namun rumus tersebut tidak bisa digunakan untuk mencari penyelesaian polinomial derajat tinggi ataupun fungsi transenden. Oleh sebab itu digunakan pendekatan untuk mendapatkan penyelesaian non-linier. Pada umumnya penyelesaian numerik diperoleh melalui metode iteratif atau perulangan.

Menurut Triatmodjo (1996), penyelesaian numerik dilakukan dengan perkiraan yang berurutan (iterasi), sedemikian sehingga setiap hasil adalah lebih teliti dari perkiraan sebelumnya. Dengan melakukan sejumlah prosedur iterasi yang dianggap cukup, akhirnya diperoleh hasil perkiraan yang mendekati hasil eksak (hasil yang benar) dengan toleransi kesalahan yang diijinkan.

2.2 Metode Numerik

Tidak sedikit suatu persamaan non-linier yang sulit untuk dikerjakan secara analitis, sehingga diperlukan metode numerik untuk mencari penyelesaiannya. Triatmodjo (1996) mengemukakan bahwa metode numerik adalah teknik untuk menyelesaikan permasalahan-permasalahan yang diinformasikan secara matematis dengan cara operasi hitungan (*arithmetic*). Metode numerik mampu menyelesaikan suatu system persamaan yang besar, non-linier dan sangat kompleks yang tidak mungkin diselesaikan secara anlitis.

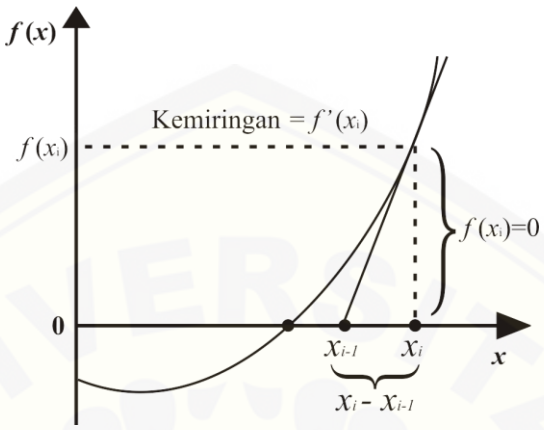
Penyelesaian yang diberikan melalui metode numerik adalah aproksimasi atau hampiran mendekati solusi eksak dengan tingkat kesalahan tertentu. Besarnya tingkat kesalahan dapat dinyatakan dalam bentuk kesalahan relatif, yaitu membandingkan kesalahan yang terjadi dengan nilai eksak.

Metode numerik untuk pencarian akar-akar persamaan non-linier dapat dibagi menjadi dua, yakni metode tertutup dan terbuka. Metode tertutup melakukan pencarian akar persamaan secara berulang di dalam interval $[a, b]$ yang berisi minimal satu penyelesaian, sehingga iterasinya selalu konvergen ke akar persamaan. Beberapa metode tertutup diantaranya Biseksi, Posisi Palsu dan Grafis. Sedangkan metode terbuka tidak memerlukan interval $[a, b]$ yang berisi akar persamaan, namun hanya memerlukan tebakan atau hampiran awal untuk memulai proses iterasi misalnya Metode Iterasi Satu-Titik Tetap, Newton Raphson dan Secant. Akibatnya metode terbuka tidak selalu konvergen menuju akar persamaan. Dari kedua metode tersebut terdapat satu kesamaan ciri yaitu dilakukan secara iteratif dan perkiraan sekarang dibuat berdasarkan perkiraan sebelumnya, sehingga terdapat kesalahan yang diperoleh dari perbedaan antara perkiraan sebelumnya dengan perkiraan sekarang

2.3 Metode Newton Raphson

Metode numerik yang populer digunakan untuk penyelesaian persamaan non-linier adalah Metode Newton-Raphson. Metode ini memanfaatkan garis singgung untuk mencari aproksimasi akar persamaan. Menurut Pujiyanta (2007), jika perkiraan awal dari akar-akar persamaan non-linier adalah x_i, suatu garis singgung dapat dibuat dari titik $(x_i, f(x_i))$ dimana garis singgung tersebut

memotong sumbu X yang biasanya memberikan perkiraan yang lebih dekat dengan nilai akar. Secara grafis, Metode Newton Raphson digambarkan pada Gambar 2.1.

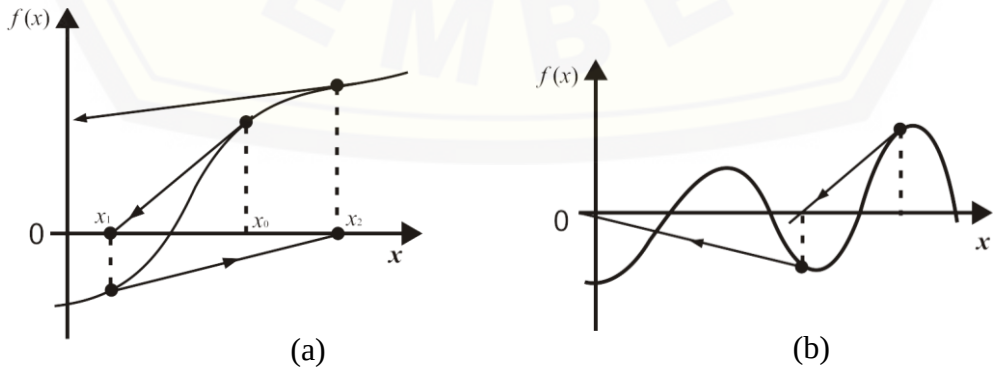


Gambar 2.1 Aproksimasi Metode Newton Raphson

Berdasarkan Gambar 2.1 diperoleh persamaan matematis untuk menentukan hampiran akar di x_{i+1}, yakni

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Walaupun Metode Newton Raphson pada umumnya sangat efisien, namun terdapat beberapa permasalahan yang menyebabkan performanya menjadi buruk. Permasalahan tersebut diantaranya titik balik ($f''(x) = 0$) terjadi di sekitar suatu akar yang mengakibatkan nilai hampiran akan menjauhi akar (Gambar 2.2a). Permasalahan kedua terjadi ketika nilai hampiran awal dekat dengan salah satu akar dapat meloncat jauh ke suatu nilai yang jauh dengan akar lainnya seperti yang ditunjukkan pada gambar 2.2b.



(a) (b)

Gambar 2.2 Permasalahan Metode Newton Raphson (sumber: Chapra,1988)

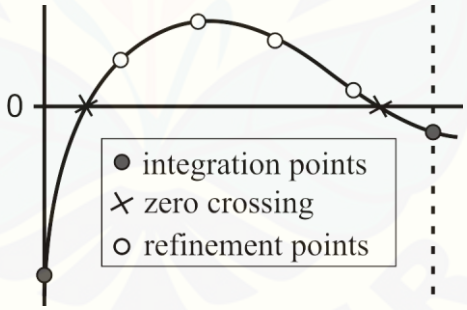
2.4 *Zero Crossing*

Zero crossing adalah keadaan suatu fungsi bernilai nol atau mengalami perpindahan nilai dari positif menjadi negatif. Metode ini seringkali digunakan untuk syarat metode tertutup karena *Zero Crossing* akan mengevaluasi tanda positif atau negatif dari nilai $f(x)$ pada ujung-ujung interval ($sign(f(a)) \neq sign(f(b))$). Jika nilai $f(x)$ pada ujung-ujung interval berbeda tanda, maka interval tersebut mengindikasikan terdapat nilai x yang menyebabkan nilai $f(x) = 0$ untuk $f(x)$ kontinu pada interval tersebut.

Menurut Zhang (2008), terdapat beberapa permasalahan yang ditemukan dalam *Zero Crossing* yakni :

a. Banyak akar

Pada Metode *Zero Crossing* sederhana, pendeteksian dilakukan dengan membandingkan tanda nilai fungsi dan jika tandanya berbeda maka dapat dinyatakan melewati $f(x) = 0$. Cara ini akan salah jika fungsi tersebut memiliki akar lebih dari satu diantara dua titik yang dievaluasi, ditunjukkan pada Gambar 2.3



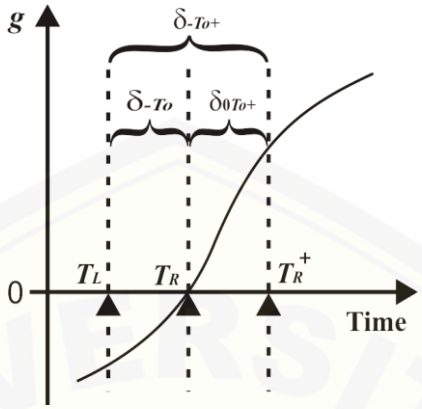
Gambar 2.3 Permasalahan akar ganda (sumber : Zhang,2008)

Pada Gambar 2.1 tampak bahwa titik evaluasi menyebabkan tanda nilai $f(x)$ akan sama. Cara alternatif untuk menyelesaikan permasalahan akar ganda adalah membagi interval awal menjadi beberapa interval kecil sehingga akar-akar persamaan non-linier terdeteksi sebagian atau semuanya (Esposito dkk, 2001).

b. *Double detection* (deteksi ganda)

Permasalahan lain adalah pendeteksian ganda pada *Zero Crossing*. Hal ini terjadi ketika nilai $f(x) = 0$ terdeteksi pada ujung kanan suatu interval (T_R)

dan juga terdeteksi pada ujung kiri inteval lain (T_L). Hal ini ditunjukkan pada Gambar 2.4



Gambar 2.4 Permasalahan deteksi ganda
(sumber : Zhang,2008)

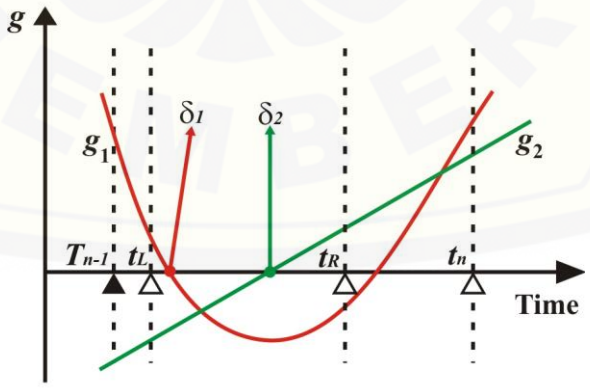
Berdasarkan gambar tersebut apabila dihitung menggunakan komputer maka nilai $f(x) = 0$ akan dianggap dua nilai yang berbeda. Untuk menyelesaikan permasalahan tersebut, $\delta_{-T_{0+}}$ dapat didefinisikan menjadi

$$\delta_{-T_{0+}} = \delta_{-T_{0+}} = \delta_{-T_0} \mid \delta_{0T_{0+}}$$

dengan | adalah operator logika "*or*".

c. Akar ganda

Masalah akar yang tertutup ditemukan jika akar persamaan lebih dari satu. Beberapa akar persamaan akan terdeteksi, namun ada akar lain yang tidak terdeteksi. Permasalahan ini dapat digambarkan pada gambar di bawah ini



Gambar 2.5 Permasalahan akar ganda
(sumber : Zhang,2008)

Penyelesaian kasus ini dengan cara menggeser level ke atas sumbu X sejauh L sehingga sehingga terdapat perpotongan dengan sumbu X. Pada Gambar 2.3, nilai $f(x) = 0$ yang sebenarnya adalah pada δ_2 yang kemudian levelnya digeser ke atas sehingga terdapat dua titik potong. Cara ini dinamakan *Level Crossing* yang merupakan pengembangan dari *Zero Crossing.* Kasus akar kembar juga dapat diselesaikan menggunakan *Level Crossing*.

2.5 Algoritma Genetika

Menurut Chipperfield, dkk (1994), Algoritma Genetika (AG) adalah metode stokastik pencarian global. Sejak awal diperkenalkan oleh John Holland pada 1975, algoritma ini dirancang untuk menyelesaikan masalah-masalah komputasi melalui komputer yang bisa melakukan seperti sifat dasar makhluk hidup.

Pada awalnya, algoritma genetika dimulai dengan sekumpulan kandidat solusi (individu atau kromosom) yang disebut populasi. Setiap kromosom menyatakan satu solusi. Populasi awal diperoleh dengan membangkitkan bilangan *random* pada suatu interval kandidat solusi. Populasi awal akan ber-"evolusi" menjadi populasi baru melalui serangkaian iterasi (generasi). Di akhir generasi, satu individu anggota populasi terbaik ditetapkan sebagai solusi dari permasalahan awal.

Menurut Suyanto (2010), proses evolusi yang terjadi pada AG adalah sebagai berikut :

a. dua individu dipilih sebagai orang tua (*parent*) menggunakan mekanisme tertentu. Kedua orang tua ini kemudian direkombinasi (*crossover*) untuk menghasilkan dua individu anak (*offspring*);
b. dengan probabilitas tertentu, dua individu anak ini mungkin mengalami perubahan gen melalui operator mutasi.
c. suatu *replacement scheme* (skema penggantian individu) atau *survivor selection* (seleksi individu-individu yang bertahan hidup) tertentu diterapkan sehingga menghasilkan populasi baru;

d. proses ini akan terus berulang sampai kondisi berhenti tertentu. Kondisi berhenti bisa berupa jumlah generasi, jumlah individu, waktu proses tertentu, atau kriteria lainnya.

2.4.1 Pengkodean Individu

Tahap awal dimulai dengan pembangkitan populasi yang berisi kromosom sebagai kandidat solusi. Penyajian kromosom yang berisi informasi dari solusi bisa dilakukan dalam beberapa cara seperti yang dijelaskan menurut Obitko (1998) dan Gen and Cheng (1997) :

a. Pengkodean biner

Setiap kromosom disimbolkan dengan angka 0 dan 1 (digit biner).

b. Pengkodean nilai

Setiap kromosom direpresentasikan sebagai nilai (angka, bilangan real, atau karakter).

c. Pengkodean bilangan permutasi

Setiap kromosom direpresentasikan sebagai urutan bilangan.

2.4.2 Nilai *Fitness*

Teori evolusi menyatakan bahwa suatu individu yang memiliki ketahanan lebih baik akan mampu bertahan hidup. Analogi dari teori tersebut, suatu kromosom yang menyatakan kandidat solusi akan dievaluasi menggunakan fungsi tertentu sebagai ukuran kemampuan untuk bertahan hidup. Fungsi tertentu yang digunakan untuk mengevaluasi disebut nilai *fitness*. Nilai ini bergantung pada jenis permasalahan yang dikerjakan, misalnya pada penelitian Amulyo (2013) menggunakan invers nilai *makespan* ($\frac{1}{M}$) sebagai nilai *fitness* pada permasalahan *flowshop*.

Algoritma genetika bertujuan untuk mencari kromosom dengan nilai *fitness* terbaik, sehingga nilai *fitness* akan dijadikan ukuran kualitas kromosom. Pada permasalahan penyelesaian persamaan non-linier, nilai *fitness* yang digunakan adalah mutlak dari nilai fungsi ($|f(x)|$). Nilai *fitness* terbaik adalah nilai yang paling minimum atau sama dengan nol.

2.4.3 Seleksi

Seleksi adalah proses pemilihan kromosom terbaik yang diharapkan mampu bertahan dan membentuk keturunan baru (*offspring*). Proses ini dianalogikan sebagai perkawinan antar dua individu di dunia nyata. Pada umumnya individu berkualitas tinggi akan melakukan perkawinan dengan sesame individu yang berkualitas tinggi, namun tidak menutup kemungkinan individu berkualitas tinggi melakukan perkawinan dengan individu berkualitas rendah. Berdasarkan uraian tersebut, penyeleksian kromosom orang tua terjadi berdasarkan probabilitas yang berbeda-beda dan bergantung secara proporsional terhadap nilai *fitness*.

Metode seleksi yang paling sederhana dan sering digunakan adalah *roulette wheel*. Menurut Yusuf dan Soesanto (2012), metode *roulette wheel* menirukan permainan *roulette wheel* dengan masing-masing kromosom menempati luasan lingkaran pada *roulette* berdasarkan nilai *fitness*. Kusumadewi, dkk (2005) menyatakan bahwa cara kerja metode ini adalah :

a. hitung total *fitness* (f)

   Total *fitness* = $\sum f_k$; $k = 1,2, \dots, PanjangKromosom$; (2.1)

b. hitung probabilitas masing-masing kromosom dari *fitness* setiap kromosom dibagi total *fitness* semua kromosom

   $$p_k = \frac{f_k}{Totalfitness} \quad (2.2)$$

c. hitung *fitness* kumulatif

   $q_1 = p_1$

   $q_k = q_{k-1} + p_k \,; k = 2,3,4, \dots, PanjangKromosom$ (2.3)

d. pilih induk yang akan di *crossover* dengan cara :

   1. bangkitkan bilangan acak $r[0,1]$;
   2. jika $r \leq q_1$, maka kromosom pertama terpilih. Jika $q_k \leq r \leq q_{k+1}$ untuk maka pilih kromosom ke (k+1) sebagai induk.

Metode seleksi yang lain adalah *tournament selection* (seleksi turnamen), merupakan metode seleksi yang didasarkan pada “pertarungan” minimal 2 kromosom yang akan menghasilkan satu pemenang (Miller,1995). Secara umum

metode ini diawali dengan pengelompokan beberapa kromosom yang akan "bertarung", pengelompokan ini dilakukan secara acak dengan jumlah kromosom pada masing-masing kelompok adalah sama. Penentuan pemenang dari setiap kelompok berdasarkan nilai *fitness*-nya, kromosom dengan nilai *fitness* terbaik di kelompoknya akan menjadi pemenang (calon induk). Pada kasus pencarian solusi persamaan non linier, nilai *fitness* terbaik adalah nilai yang mendekati 0 (nol).

2.4.4 Pindah Silang (*Crossover*)

Pindah silang pada algoritma genetika merupakan hasil pengadopsian secara sederhana dari kehidupan nyata yakni perkawinan dua individu akan menghasilkan anak yang memiliki gen-gen orang tuanya. Pada algoritma genetika juga terjadi hal yang serupa melalui pindah silang atau *crossover*, menurut Suyanto (2010) penggabungan gen-gen dari kedua kromosom orang tua akan menghasilkan dua kromosom anak (*offspring*) yang mungkin berbeda dengan kedua orang tuanya.

Tujuan dari *crossover* untuk mendapatkan kromosom-kromosom yang bervariasi. Sedangkan penentuan posisi gen-gen yang di-*crossover* dilakukan secara acak, sehingga *offspring* yang dihasilkan bisa berkualitas lebih baik, lebih buruk, atau sama dengan orang tuanya.

Crossover hanya bisa dilakukan apabila bilangan acak yang dibangkitkan berada di bawah probabilitas (*Pc*) tertentu yang ditentukan di awal. Nilai P_c antara 0,8 sampai 0,95, tapi untuk beberapa permasalahan $P_c = 0{,}6$ memiliki hasil yang paling baik (Obitko,1998). Terdapat beberapa cara *crossover*, yang paling sederhana dan sering digunakan adalah *one point cut crossover*. Titik potong dipilih secara acak, kemudian bagian pertama dari orang tua 1 digabungkan dengan bagian kedua dari orang tua 2 dan begitu pula sebaliknya.

titik potong

Orang tua 1	2	1	3	4	2	3	4	1	1	4	2
Orang tua 2	3	1	4	2	4	3	3	1	2	3	4
Anak 1	2	1	3	4	2	3	3	1	2	3	4
Anak 2	3	1	4	2	4	3	4	1	1	4	2

Gambar 2.6 *one point cut crossover*

Crossover lainnya adalah *flat crossover*, merupakan cara perkawinan antara dua induk yang memungkinkan terjadinya perubahan gen-gen secara keseluruhan tanpa mengubah sifat yang diturunkan induknya. Berbeda halnya dengan *one cut point crossover* yang menggunakan titik potong, *flat crossover* menggunakan bilangan random untuk melakukan *crossover*. Induk 1 dan induk 2 dinyatakan dalam bentuk vektor dan bilangan random yang dibangkitkan juga dalam bentuk vektor yang sama panjang dengan vektor induk. Mendes (2013) menjelaskan bahwa anak 1 (*offspring*) diperoleh dari vektor kombinasi *linear* sesuai aturan berikut

$$x_i^1 = r_i x_i + (1 - r_i) x_{2,i} \qquad i = 1 \dots n \qquad (2.4)$$

dan anak 2 juga dianalogikan seperti anak 1.

Parent 1	0.32	0.22	0.34	0.89	0.23	0.76	0.78	0.45
Parent 2	0.12	0.65	0.38	0.47	0.31	0.56	0.88	0.95
Random 1	0.32	0.22	0.34	0.89	0.23	0.76	0.78	0.45
Offspring 1	0.18	0.56	0.37	0.84	0.29	0.71	0.8	0.73
Random 2	0.16	0.34	0.92	0.54	0.65	0.76	0.98	0.32
Offspring 2	0.15	0.5	0.34	0.7	0.26	0.71	0.78	0.79

Gambar 2.7 *flat crossover*

(sumber : Mendes, 2013)

2.4.5 Mutasi

Operasi pendukung untuk menghasilkan kromosom baru yang dihasilkan secara acak adalah operator mutasi. Diadopsi dari dunia nyata bahwa adanya kemungkinan individu mengalami mutasi gen secara stokastik. Proses mutasi bisa menghasilkan kromosom baru yang lebih baik, lebih buruk, atau sama dengan kromosom lama. Proses stokastik pada langkah ini dilakukan dengan memilih secara acak posisi gen yang akan dimutasi.

Proses mutasi pada algoritma genetika berguna untuk menggantikan kromosom yang hilang selama proses seleksi sehingga bisa diujikan pada suatu kondisi yang baru. Selain itu, mutasi berguna untuk menyediakan kromosom yang

tidak ditampilkan pada populasi awal. Untuk mengetahui jumlah kromosom yang akan dihasilkan dari proses mutasi diberikan parameter laju mutasi (P_m), yakni rasio kromosom yang mengalami mutasi dari total populasi. Pada umumnya nilai Pm diantara 0,05 sampai dengan 0,1 (Obitko,1998).

2.6 *Virus Evolutionary Genetic Algorithm* (VEGA)

Teori evolusi yang dikemukakan oleh Charless Darwin memberikan inspirasi bagi peneliti-peneliti lain untuk menyempurnakannya. Sebuah pengembangan dari teori evolusi Darwin yang menyatakan adanya seleksi alam, Kubota dkk. mengajukan algoritma baru bernama *Virus Evolutionary Genetic Algorithm* (VEGA). Algoritma ini adalah penggabungan antara algoritma genetika dan infeksi virus (Fukuda dkk, 1999). VEGA disusun dari dua populasi yakni populasi *host* dan populasi virus. Populasi *host* sama dengan populasi yang ada pada algoritma genetika yakni kandidat solusi. Sedangkan populasi virus adalah *substring* dari populasi *host* yang akan menginfeksi populasi *host*.

Menurut Ling (2006), ide terpenting dari VEGA adalah populasi virus yang memiliki dua operator, operator *reserve transcription* dan proses *transduction*. Di biologi, *reserve transcription* adalah proses dari virus yang menggunakan enzim untuk mengganti RNA mereka menjadi DNA. Operator *reserve transcription* juga demikian, mengganti gen kromosom pada *substring* kromosom *host*. Pada setiap generasi, beberapa virus memiliki kemungkinan *substring*-nya menjadi kromosom populasi *host*. Sedangkan operator transduksi adalah proses mengambil DNA/gen dari individu *host* untuk dijadikan virus baru.

Menurut Fukuda (1999), beberapa elemen yang terdapat pada VEGA adalah sebagai berikut :

a. *Interinfection time* : satu waktu interval iterasi dari infeksi virus;
b. $host_j$: individu *host* ke - *j* sebelum mengalami *reverse transcription;*
c. $host_j'$: individu *host* ke - *j* setelah mengalami *reverse transcription;*
d. $fithost_j$: nilai *fitness* individu *host* sebelum mengalami *reverse transcription*;

e. $fithost_j'$: nilai *fitness* individu *host* setelah mengalami *reverse transcription*;

f. $fitvirus_{i,j}$: perbedaan antara $fithost_j$ *dan* $fithost_j'$

$$fitvirus_{i,j} = fithost_j' - fithost_j \quad (2.5)$$

g. $fitvirus_i$: kekuatan infeksi virus

$$fitvirus_i = \sum_{j \in s} fitvirus_{i,j} \quad (2.6)$$

h. S : himpunan individu *host* yang diinfeksi virus ke I;

i. $Life_i$: kekuatan hidup virus

$$Life_i = r \times Life_{i,t-1} + fitvirus_i \quad (2.7)$$

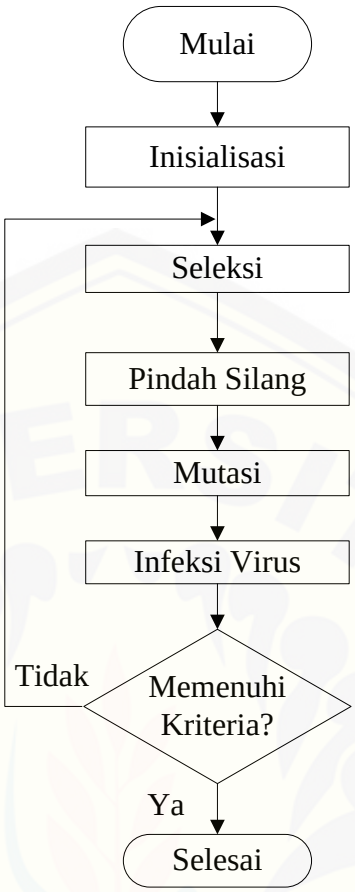
j. r : tingkat penurunan daya hidup virus (dengan nilai [0,1]);

k. t : generasi virus.

Berdasarkan uraian di atas, parameter yang terdapat pada VEGA adalah P_c (*Probability Crossover*), P_m (*Probability Mutation*), r (tingkat penuruan daya hidup virus), *interinfection time* (jumlah maksimal proses GA sebelum diinfeksi), *max gen* (jumlah maksimal generasi), *host pop size* (jumlah individu *host* dalam populasi), P_v (Probabilitas individu *host* yang akan diinfeksi virus) dan virus *pop size* (jumlah individu virus dalam populasi). Nilai *fitness* virus menyatakan pengaruh secara keseluruhan dari virus terhadap *host* yang diinfeksi. Jika bernilai positif maka virus memberikan efek positif pada individu *host* dan sebaliknya jika bernilai negatif maka virus tidak mampu memperbaiki individu *host* (Ling, 2006).

Nilai kekuatan hidup virus ($Life_i$) akan bernilai positif atau negatif. Apabila $Life_i$ bernilai negatif maka virus tidak lagi memberikan pengaruh positif terhadap individu *host* sehingga operator *transduction* akan memperbarui kromosom virus dengan mengambil *substring* individu *host* secara acak. Jika $Life_i$ bernilai positif maka individu virus mendapatkan sebagian *substrings* dari individu *host* yang diinfeksi.

Langkah-langkah pada VEGA dapat diilustrasikan pada *flowchart* di bawah ini :

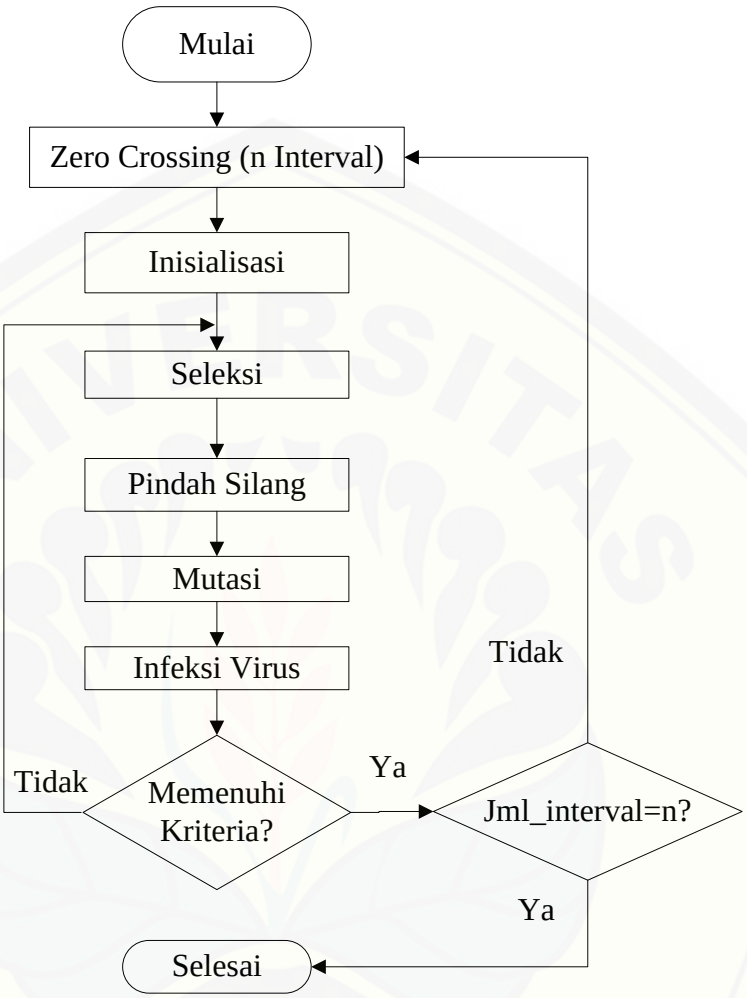


Gambar 2.8 *Flowchart* VEGA

2.7 Gabungan Metode *Zero Crossing* dan *Virus Evolutionary Genetic Algorithm* (VEGA)

Gabungan Metode *Zero Crossing* dan *Virus Evolutionary Genetic Algorithm* (VEGA) dilakukan dengan menempatkan metode *Zero Crossing* di awal iterasi sebelum pembangkitan populasi awal. Input berupa interval sebagai kandidat solusi akan dipecah menjadi beberapa interval kecil untuk dievaluasi. Jika terdapat perbedaan tanda pada masing-masing batas interval, maka populasi akan dibangkitkan pada interval tersebut. Sedangkan interval-interval yang tidak memberikan perubahan tanda pada batas-batas intervalnya tidak akan digunakan kecuali interval yang hasil evaluasinya sama dengan nol, sehingga proses iterasi akan lebih cepat karena hanya dilakukan pada interval yang kecil dan dipastikan memiliki akar.

Langkah-langkah gabungan Metode *Zero Crossing* dan VEGA dapat diilustrasikan pada *flowchart* di bawah ini :



Gambar 2.9 *Flowchart* gabungan Metode *Zero Crossing* dan VEGA

2.8 MATLAB

MATLAB (*Matrix Laboratory*) adalah aplikasi untuk menganalisis dan melakukan proses komputasi data numerik dengan menggunakan bahasa pemrograman matematika lanjutan, yang dibentuk dengan sifat dan bentuk matriks. Bahasa dalam MATLAB mampu mengintegrasikan kemampuan visual, komputasi dan pemrograman dalam suatu lingkungan yang tunggal dan mudah digunakan. Array atau matriks menjadi standar variabel pada MATLAB.

Program MATLAB yang dikembangkan oleh Mathworks Inc. digunakan untuk menyelesaikan masalah matematis yang kerap ditemui pada bidang teknis.

MATLAB juga menyediakan beberapa pilihan untuk dipelajari, misalnya visualisasi, pemrograman, atau keduanya. Dalam bukunya, Widiarsono (2005) menyatakan bahwa salah satu aspek yang sangat berguna dari MATLAB adalah kemampuannya untuk menggambarkan berbagai jenis grafik sehingga data dan fungsi yang kompleks dapat tervisualisasi dengan baik. Selain itu, MATLAB juga memberikan keuntungan bagi *programmer* untuk menjadi program pembanding.

BAB 3. METODE PENELITIAN

Pada penelitian ini, penulis akan menerapkan gabungan Metode *Zero crossing* dan *Virus Evolutionary Genetic Algorithm* (VEGA) untuk menyelesaikan persamaan non-linier. Langkah-langkah yang akan dilakukan dalam penelitian ini dapat digambarkan pada diagram alir berikut:

Gambar 3.1 Skema metode penelitian

Penjelasan skema pada Gambar 3.1 untuk memperoleh hasil yang diinginkan adalah sebagai berikut:

a. Studi literatur

Studi literatur dilakukan dengan mengumpulkan dan mempelajari berbagai teori yang menunjang penelitian ini. Teori yang dipelajari meliputi Metode *Zero Crossing*, *Virus Evolutionary Genetic Algorithm* (VEGA), penyelesaian persamaan non-linier dan teori-teori penunjang lainnya. Studi literatur dilakukan sebagai pedoman penelitian baik pengaplikasian dan analisis yang

akan dilakukan. Selain itu studi literatur juga bertujuan agar lebih memahami teori-teori yang digunakan dalam penelitian.

b. Menentukan masalah

Masalah yang akan diteliti adalah penyelesaian persamaan non-linier. Persamaan non-linier yang digunakan disesuaikan dengan batasan masalah pada penelitian ini yakni persamaan non-linier satu variabel berupa persamaan polinomial dan fungsi transenden kontinu berupa trigonometri, eksponensial, maupun gabungannya. Persamaan-persamaan tersebut merujuk pada beberapa jurnal. Dari beberapa jurnal rujukan sebagian besar menggunakan polinomial dan eksponensial karena bentuk tersebut lebih mudah untuk ditemukannya sebuah solusi, sehingga dapat dianalisis tingkat akurasi dari solusi yang diperoleh menggunakan suatu metode.

c. Membuat Program

Pembuatan program dilakukan menggunakan *software* MATLAB R2009a. Langkah-langkah yang diperlukan adalah sebagai berikut:

1) Input

Peneliti akan memberikan input berupa persamaan non-linier (dalam bentuk fungsi) yang akan dicari penyelesaiannya, parameter-parameter yang digunakan dan kriteria pemberhentian. Parameter yang digunakan adalah panjang interval, P_c, P_m, *interinfection time*, *max gen*, *host pop size*, P_v dan virus *pop size*. Kriteria pemberhentian berupa nilai *fitness* dan iterasi. apabila nilai *fitness* dari solusi sama dengan 0 (nol) maka iterasi akan dihentikan. Kriteria kedua adalah iterasi maksimum seperti metode numerik, namun tidak menutup kemungkinan pada *Zero Crossing* dapat ditemukan solusi sehingga tidak perlu melanjutkan pada langkah berikutnya.

2) Proses

Input berupa interval akan dibagi menjadi interval yang lebih kecil untuk dievaluasi pada setiap ujung interval menggunakan *Zero Crossing* dan hasilnya berupa interval yang memuat solusi. Pada interval kecil tersebut akan dibangkitkan kromosom (kandidat solusi) sebanyak *host pop*

size dan direpresentasikan dalam bentuk biner. Selanjutnya dilakukan evaluasi nilai *fitness*, seleksi, pindah silang, mutasi, transkripsi dan transduksi sesuai langkah-langkah yang terdapat pada VEGA.

3) Output

Output program berupa kromosom atau individu dengan nilai *fitness* terbaik serta akan ditampilkan grafik berupa pergerakan gabungan metode *Zero Crossing* dan VEGA dimana sumbu *X* merupakan iterasi dan sumbu *Y* adalah nilai *fitness*.

d. Implementasi gabungan Metode *Zero Crossing* dan VEGA

Peneliti akan menyelesaikan persamaan non-linier dengan program gabungan Metode *Zero Crossing* dan VEGA yang telah dibuat pada langkah c. Simulasi program dilakukan menggunakan persamaan yang diperoleh pada langkah b.

e. Analisis hasil

Peneliti akan menganalisis hasil yang diperoleh menggunakan program. Gabungan Metode *Zero Crossing* dan VEGA akan dianggap baik jika output mendekati atau sama dengan solusi eksak. Apabila tidak ditemukan solusi eksak, maka dilakukan perbandingan terhadap hasil yang terdapat pada jurnal rujukan di langkah b. Hal ini dilakukan guna menentukan akurasi hasil yang diperoleh oleh gabungan Metode *Zero Crossing* dan VEGA. Kriteria keberhasilan berupa tingkat akurasi hasil yang diperoleh dan waktu komputasi. Selain itu peneliti akan menganalisis pengaruh beberapa nilai parameter terhadap solusi maupun waktu komputasi.

f. Kesimpulan

Pengambilan kesimpulan dilakukan dengan memberikan jawaban dari tujuan penelitian ini. Saran-saran juga diberikan untuk perbaikan bagi penelitian selanjutnya.

BAB 5. PENUTUP

5.1 Kesimpulan

Berdasarkan hasil analisis dan pembahasan pada bab sebelumnya, maka diperoleh kesimpulan dari penerapan gabungan Metode *Zero Crossing* dan VEGA pada penyelesaian persamaan non-linier sebagai berikut :

a. gabungan Metode *Zero Crossing* dan VEGA dapat menyelesaikan persamaan non-linier baik akar tunggal maupun akar ganda dengan hasil yang mendekati atau sama dengan solusi eksak. Solusi diperoleh dengan input berupa fungsi, interval, iterasi $= 200$, *interinfection time* $= 3$, *pop size* $= 20$, virus *pop size*$= 9$, $Pc = 0{,}8$, $Pm = 0{,}1$ dan $Pv = 0{,}4$;
b. solusi yang diperoleh menggunakan gabungan Metode *Zero Crossing* dan VEGA memberikan hasil yang lebih baik dibandingkan dengan Metode Newton-Raphson dan beberapa metode pada jurnal rujukan. Pada PNL 4 dan 7 (akar tunggal) mampu memberikan solusi yang sama dengan solusi eksak seperti hasil Newton Raphson. Pada PNL 10 (akar tunggal), gabungan Metode *Zero Crossing* dan VEGA mampu mendapatkan 3 solusi sedangkan Javidi (2007) hanya memperoleh 2 solusi. Pada PNL 4 (akar ganda) juga mampu memperoleh hasil yang sama dengan solusi eksak dan nilai *fitness* $|f(x)| = 0$.

5.2 Saran

Gabungan Metode Zero Crossing dan VEGA pada penelitian ini telah memberikan hasil yang baik, tetapi juga memiliki beberapa batasan dan kelemahan seperti yang telah disebutkan sebelumnya. Disarankan kepada peneliti selanjutnya untuk mengaplikasikan gabungan Metode Zero Crossing dan VEGA pada kasus numerik yang lain guna menambah khazanah baru ketika suatu permasalahan numerik diselesaikan dari sudut pandang masalah optimasi, misalnya penyelesaian persamaan non-linier akar ganda dengan *multiplicity*, persamaan diferensial, kasus integrasi, penyelesaian persamaan non-linier multivariabel dan lainnya.

DAFTAR PUSTAKA

Amulyo, R., Suprajitno, H. & Miswanto. 2013. Hybrid Virus Evolutionary Genetic Algorithm dan Simulated Annealing Pada Penjadwalan Flowshop. *Jurnal Matematika Departemen Matematika Universitas Airlangga* **2** (2) : 077 – 085.

Arif, M. Z. & Julianto, B. 2013. Modifikasi Metode Chebyshev Orde Tiga untuk Mencari Akar Ganda Tanpa Menggunakan Turunan. *Majalah Ilmiah Matematika dan Terapan* **13** : 070 – 079.

Chapra, S. C. & Canale, R. P., 1989. *Metode Numerik* (terjemahan). Jakarta : Erlangga.

Chipperfield, A., Flemingm, P., Pohlheim, H., & Fonseca, C. 1994. *Genetic Algorithm Toolbox for Use with MATLAB.* Sheffield : University of Sheffield.

Chun, C., 2005. Iterative Methods Improving Newton's Method by Decomposition Method. *Computer and Mathematics with Aplication* **50** :1559-1568.

Esposito, J. M., Kumar, V. & Pappas, G. J. 2001. Accurate Event Detection for Simulating Hybrid Systems. *Lecture Notes in Computer Science* (204 - 217). University of Pennsylvania.

Fountas, N. A. & Vaxevandis, N. M.. 2013. A Modified Virus Evolutionary Genetic Algorithm For Rough Machining Optimization of Sculptured Surfaces. *International Journal of Engineering Tome XI* **3** : 283 – 288.

Fukuda, T., Shimojima, K., & Kubota, N. 1996. *The Role of Virus Infection in Virus Evolutionary Genetic Algorithm*. Nagoya : Nagoya University.

Fukuda, T., Shimojima, K., & Kubota, N. 1999. *Virus Evolutionary Genetic Algorithm and Its Applications To Traveling Salesman Problem*. Editor : Xin Yao. Singapura : World Scientific.

Gen, M. & R. Cheng, 1997. *Genetic Algorithm and Engineering Design*. New york.: John Wiley & Sons.

Indrianingsih, Y. 2010. Algoritma Genetika Untuk Menyelesaikan Masalah Optimasi Fungsi Berkendala Dengan Pengkodean Bilangan Bulat. *Jurnal Angkasa Sekolah Tinggi Teknologi Adisutjipto (STTA)* **2** (1) : 067 – 076.

Javidi, M., & Golbabai, A. 2007. A Third-Order Newton Type Method for Nonlinear Equations Based on Modified Hemotopy Peturbation Method. *Aplied Mathematics and Computation* **191** : 199-205.

Kusumadewi, S., & Purnomo, H. 2005. *Penyelesaian Masalah Optimasi dengan Teknik-Teknik Heuristik*. Yogyakarta : Graha Ilmu.

Liang, J., *et al*. 2015. Fifth-Order Iterative Method for Solving Multiple Roots of the Highest Multiplicity of Nonlinear Equation. *Algorithms* **8** : 656-668.

Ling, W. X. 2006. *Virus Transmision Genetic Algorithm.* Thesis. Athens : The University of Georgia.

Mendes, J. M. 2013. *A Comparative Study of Crossover Operators for Genetic Algorithms to Solve the Job Shop Scheduling Problem*. Portugal : School of Engineering – Polytechnic of Porto.

Noor, M. A. 2007. New Iterative Schemes for Nonlinear Equations. *Aplied Mathematics and Computation* **187** : 937-943.

Obitko, M., 1998. *Introduction To Genetic Algorithm*. Prague : Czech Technical University.

Pujiyanta, A. 2007. *Komputasi Numerik dengan MATLAB*. Yogyakarta : Graha Ilmu.

Shengguo, L., Xiangke, L., & Lizhi, C. 2009. A New-Fourth Order Iterative Method for Finding Multiple Roots of Nonlinear Equations. *Aplied Mathematics and Computation* **215** : 1288-1292.

Suyanto. 2010. *Algoritma Optimasi (Deterministik atau Probabilistik).* Yogyakarta : Graha Ilmu.

Triatmodjo, B. 1996. *Metode Numerik*. Yogyakarta : Peta Offset.

Wang, P. 2011. A Third-Order Family Of Newton Like-Iteration Methods for Solving Nonlinear Equations. *Journal of Mathematics and Stochastics* **3** (1) : 13-19.

Widiarsono, T., 2005. *Tutorial Praktis Belajar MATLAB*. Jakarta.

Wu, X.Y. & Fu, D.S. 2001. New high-order convergence iteration methods without employing derivatives for solving nonlinear equations. *Comput. Math. Appl*. **41** : 489-495.

Yusuf, A. & Soesanto, O. 2012. Algoritma Genetika Pada Penyelesaian Akar Persamaan Sebuah Fungsi. *Jurnal Matematika Murni dan Terapan* **6** (2) : 047 – 056.

Zhang, F., Yeddanapudi, M., & Mosterman, P.J. 2008. *Zero Crossing Location and Detection Algorithm for Hybrid Simulation*. USA : The Mathworks, Inc.

LAMPIRAN

A. *Script* Program Gabungan Metode *Zero Crossing* dan *Virus Evolutionary Genetic Algorithm* (VEGA)

```
fungsi=str2func(get(handles.editfung,'string'));%@(x)(fu
ngsi)
interval=str2num(get(handles.editint,'string'));%inputan
berupa vektor a b
pop_size=str2double(get(handles.editpop,'string'))';
vpop_size=str2double(get(handles.editvpop,'string'));
max_gen=str2double(get(handles.edititer,'string'));
int_time=str2double(get(handles.editinttime,'string'));
pc=str2double(get(handles.editpc,'string'));
pm=str2double(get(handles.editpm,'string'));
pv=str2double(get(handles.editpv,'string'));

waktu=tic;
%Zero Crossing
pjg_interval=interval(2)-interval(1);
ind_ter=0;
inter=[];
for i=1:pjg_interval
    calon_interval(i,1)=interval(:,1);
    calon_interval(i,2)=interval(:,1)+1;

zero=feval(fungsi,calon_interval(i,1))*feval(fungsi,calo
n_interval(i,2));
    if zero<0
        ind_ter=ind_ter+1;
        inter(ind_ter,1)=calon_interval(i,1);
        inter(ind_ter,2)=calon_interval(i,2);
    elseif zero==0
        ind_ter=ind_ter+1;
        if feval(fungsi,calon_interval(i,1))==0
            solution_bulat(ind_ter)=calon_interval(i,1);
        elseif feval(fungsi,calon_interval(i,2))==0
            solution_bulat(ind_ter)=calon_interval(i,2);
        end
    end
    interval(1)=interval(:,1)+1;
end
[panjang_inter kolom_inter]=size(inter);
```

```
if panjang_inter~=0
    set(handles.listbox3,'string','Iterasi     Solusi
Fitness');
    for i=1:panjang_inter
    %Inisialisasi Populasi
        [bar kol]=size(inter);
        pop=(inter(i,2)-
inter(i,1))*rand(pop_size,1)+inter(i,1);
    %Inisialisasi Virus
        pop_virus=(inter(i,2)-
inter(i,1))*rand(vpop_size,1)+inter(i,1);
    %nilai harapan hidup awal virus
        life=zeros(1,vpop_size);
    for j=1:max_gen
            %Pengkodean Pop Host Biner
            for b=1:pop_size

host_bin_integer(b,:)=conv2bin(pop(b,:));
                host_bin_float(b,:)=float2bin(pop(b,:));
            end
            %Pengkodean Pop Virus Biner
            for c=1:vpop_size

virus_bin(c,:)=float2binvirus(pop_virus(c));
            end
            %Evaluasi Nilai Fitness
            for d=1:pop_size
                fitness(d,:)=abs(fungsi(pop(d,:)));
            end
            for k=1:int_time
            %Seleksi Turnamen
            host_induk=tour(fitness);
            %Crossover
            pan=length(host_induk);
            acak_cross=rand(pan,1);
            ind=find(acak_cross<pc);
            panjang_ind=length(ind);
            for e=1:panjang_ind
                induk_bin_float
(e,:)=host_bin_float(host_induk(ind(e)),:);
            end
            [offs_cross
offs_bin_cross]=flatcross(induk_bin_float,inter(i,1),int
er(i,2));
            jml_cross=length(offs_cross);
```

```
            %Evaluasi nilai fitness crossover
            for r=1:jml_cross

fitness_cross(r,:)=abs(fungsi(offs_cross(r,:)));
            end
            %Mutasi
            jml_mut=round(pm*pop_size);
            acak_mut=randperm(pop_size);
            rand_mut=acak_mut(1:jml_mut);
            for f=1:jml_mut
                [offs_mut(f,:)
offs_bin_mut(f,:)]=mutation(host_bin_float(rand_mut(f),:
),inter(i,1),inter(i,2));

fitness_mut(f,:)=abs(fungsi(offs_mut(f,:)));
            end
            %Penggabungan semua host
            pop_dummy(1:pop_size,:)=pop;

pop_dummy(pop_size+1:pop_size+jml_cross,:)=offs_cross;
pop_dummy(pop_size+jml_cross+1:pop_size+jml_cross+jml_mu
t,:)=offs_mut; fitness_dummy(1:pop_size,:)=fitness;

fitness_dummy(pop_size+1:pop_size+jml_cross,:)=fitness_c
ross;
fitness_dummy(pop_size+jml_cross+1:pop_size+jml_cross+jm
l_mut,:)=fitness_mut;

host_bin_float_dummy(1:pop_size,:)=host_bin_float;
host_bin_float_dummy(pop_size+1:pop_size+jml_cross,:)=of
fs_bin_cross;
host_bin_float_dummy(pop_size+jml_cross+1:pop_size+jml_c
ross+jml_mut,:)=offs_bin_mut;
            pilih=randperm(length(pop_dummy));
            for u=1:pop_size
                pop(u,:)=pop_dummy(pilih(u));
                fitness(u,:)=fitness_dummy(pilih(u));

host_bin_float(u,i)=host_bin_float_dummy(pilih(u));
            end
            pop_dummy=[ ];fitness_dummy=[
];host_bin_float_dummy=[ ];
        end
        %Infeksi virus
        %Transcription & Transduction
```

```
        jml_infeksi=round(pv*pop_size);
        for g=1:vpop_size
            acak_infek=randperm(pop_size);
            rand_inf=acak_infek(:,1:jml_infeksi);
            for h=1:jml_infeksi

pop_infek(h,:)=host_bin_float(rand_inf(h),:);
            end
            [hasil_inf
inf_bin]=transcription(virus_bin(g),pop_infek,inter(i,1)
,inter(i,2));
            %evaluasi nilai fitness setelah dan sebelum
infeksi
            for l=1:jml_infeksi

fitness_inf(l,:)=abs(fungsi(hasil_inf(l)));

fitness_host_inf(l,:)=fitness(rand_inf(l));
                fit_virus(l,:)=fitness_inf(l)-
fitness_host_inf(l);
                if fit_virus(l)<0
                    fitness(rand_inf(l))=fitness_inf(l);
                    pop(rand_inf(l))=hasil_inf(l);

host_bin_float(rand_inf(l))=inf_bin(l);
                end
            end
            %evaluasi nilai harapan hidup virus
            fitvirus_total=sum(fit_virus);
            rand_life=rand(1,1);

life(:,g)=life(:,g)*rand_life+fitvirus_total;
            if life(:,g)<0
                %Transduction
                off_transduc=randi([1 jml_infeksi],1,1);
                pop_transduc=randi([1 41],1,1);
                for m=1:10

virus_bin(g,m)=inf_bin(off_transduc,pop_transduc);
                    pop_transduc=pop_transduc+1;
                end
            end
            pop_virus(g,:)=bin2float(virus_bin(g,:));
        end
        [urut id_fit]=sort(fitness);
```

```
        indeks_solusi=find(id_fit==1);
        solusi(j,:)=pop(indeks_solusi);
        fitness_solusi(j,:)=fitness(indeks_solusi);
        ss=get(handles.listbox3,'string');
        ss1=sprintf('%4.0f % 18.15f % 18.15f
\n',j,solusi(j,:),fitness_solusi(j,:));
        set(handles.listbox3,'string',char({ss;ss1}));
        guidata(hObject,handles);
        axes(handles.axesgrafik);

figure_plot(j)=plot(j,fitness_solusi(j,:),'r',...
            'LineWidth',6);
        hold on
    end
    %Solusi Terbaik
    [sol urut_sol]=sort(fitness_solusi);
    indeks_sol=urut_sol(1);
    solution(:,i)=solusi(indeks_sol);
    fit_sol(:,i)=fitness_solusi(indeks_sol);
    end
    running_time=toc(waktu);
    tampil={['solusi: ' sprintf('%0.15f ',solution)];
        ['fitness: ' sprintf('%0.15f ',fit_sol)];
        ['running time: ' sprintf('%0.15f
',running_time)]};
    set(handles.listsol,'string',char(tampil));
    h=figure('visible','off');
    plot(1:max_gen,fitness_solusi,'r-','LineWidth',0.5);

line(urut_sol(1),fit_sol,'Marker','s','MarkerEdgeColor',
'k','MarkerFaceColor','g','MarkerSize',5);
    text(urut_sol(1),fit_sol,sprintf(['x = '
num2str(urut_sol(1)) '\ny = ' num2str(fit_sol) '\n \n
\n']));
    xlabel('Iterasi');ylabel('Nilai Fitness');
    saveas(h,'fungsi.jpg');
elseif panjang_inter==0
    cek=length(solution_bulat);
    if cek~=0
        s=1;
        for i=1:cek-1
            for j=i+1:cek
                if solution_bulat(i)==solution_bulat(j)
                    solution_bulat(i)=solution_bulat(i);
                    s=s+1;
```

```
                else
solution_bulat(s)=solution_bulat(i+1);
                    s=s+1;
                end
            end
        end
        fit_sol=0;running_time=toc(waktu);
        tampil={['solusi: ' sprintf('%0.15f
',solution_bulat)];
            ['fitness: ' sprintf('%0.15f ',fit_sol)];
            ['running time: ' sprintf('%0.15f
',running_time)]};
        set(handles.listsol,'string',char(tampil));
    else
        errordlg('Coba Interval Lain','Error Interval');
    end
end
```

B. *Script* Program Pengkodean Biner Populasi *host* (float2bin.m)

```
function floatbin=float2bin(pop)
%Pengkodean Host Biner Desimal
kosong=zeros(1,53);
z=0;
integer=floor(abs(pop));
desimal=abs(pop)-integer;
temp=kosong;
if desimal<1
    while desimal~=0
        desimal=desimal*2;
        kurang=desimal-1;
        if kurang<1 && kurang>=0
           z=z+1;
           temp(z)=1;
           desimal=kurang;
        elseif kurang<0
            z=z+1;
            temp(z)=0;
        end
    end
end
  floatbin=temp;
end
```

C. *Script* Program Seleksi Turnamen (tour.m)

```
function win=tour(fitnes)
tournSize=2;
n=length(fitnes);
ind=1;
for i=1:tournSize
competitors = reshape(randperm(n), tournSize,
n/tournSize)';
[bar kol]=size(competitors);
for j=1:bar

a(j,1)=min(fitnes(competitors(j,1)),fitnes(competitors(j
,2)));
    for k=1:n
        cocok=fitnes(k,:);
        if a(j,1)==cocok
           hasil(ind,1)=k;
           ind=ind+1;
           break
        end
    end
end
win=hasil;
end
```

D. *Script* Program *crossover* (flatcross.m)

```
 %Flatcrossover
function
[crossover,offs_bin_cross]=flatcross(biner_float,inter_b
awah,inter_atas)
[n col]=size(biner_float);
a=1;
for i=1:n-1
    for j=i+1:n
        for l=1:2
            r_flat=randi([0 1],1,53);
            bin_off(a,:)=r_flat.*biner_float(i,:)+(1-
r_flat).*biner_float(j,:);
            jumlah=0;
            for k=1:53
                jumlah=jumlah+(bin_off(a,k)*(1/2)^(k));
            end
```

```
            if inter_bawah<0 && inter_atas<0
                offs(a,:)=inter_atas-jumlah;
            elseif inter_bawah<0 && inter_atas==0
                offs(a,:)=inter_atas-jumlah;
            elseif inter_bawah>=0 && inter_atas>0
                offs(a,:)=inter_bawah+jumlah;
            end
            a=a+1;
        end
    end
    crossover=offs;
    offs_bin_cross=bin_off;
end
end
```

E. *Script* Program Mutasi (mutation.m)

```
 %Mutasi Pembalikan
function
[mutasi,offs_bin_mut]=mutation(biner_mut,inter_bawah,int
er_atas)
[baris kolom]=size(biner_mut);
for i=1:baris
    pop_mut(i,:)=biner_mut(i,:);
    acak=randperm(kolom);
    if acak(1)>acak(2)
        pos1=acak(2);
        pos2=acak(1);
    else
        pos1=acak(1);
        pos2=acak(2);
    end
    tengah=floor((pos2-pos1)/2);
    for k=pos1:tengah
        dummy=pop_mut(i,k);
        pop_mut(i,k)=pop_mut(i,pos2);
        pop_mut(i,pos2)=dummy;
        pos2=pos2-1;
    end
    offs_mut(i,:)=0;
    for j=1:kolom

offs_mut(i,:)=offs_mut(i,:)+(pop_mut(i,j))*(1/2)^(j);
    end
```

```
    if inter_bawah<0 && inter_atas<0
        jumlah(i,:)=inter_atas-offs_mut(i,:);
    elseif inter_bawah<0 && inter_atas==0
        jumlah(i,:)=inter_atas-offs_mut(i,:);
    elseif inter_bawah>=0 && inter_atas>0
        jumlah(i,:)=inter_bawah+offs_mut(i,:);
    end
end
mutasi=jumlah;
offs_bin_mut=pop_mut;
```

F. *Script* Program *Transcription* (tour.m)

```
 %Transcription
function
[off_trans,trans_bin]=transcription(v_bin,pop_trans,inte
r_bawah,inter_atas)
[row col]=size(v_bin);
[bar kol]=size(pop_trans);
batas=kol-col;
b=1;
for j=1:bar
    pos=randi([1 batas],1,1);
    a=1;
    dummy(j,:)=0;
    for k=pos:pos+col-1
        pop_trans(j,k)=v_bin(:,a);
        a=a+1;
    end
    for l=1:kol

dummy(j,:)=dummy(j,:)+(pop_trans(j,l))*(1/2)^(l);
    end
    if inter_bawah<0 && inter_atas<0
        off_trans(j,:)=inter_atas-dummy(j,:);
    elseif inter_bawah<0 && inter_atas==0
        off_trans(j,:)=inter_atas-dummy(j,:);
    elseif inter_bawah>=0 && inter_atas>0
        off_trans(j,:)=inter_bawah+dummy(j,:);
    end
    trans_bin(j,:)=pop_trans(j,:);
end
end
```