RICARDO CERQUEIRA DE SOUZA

FERNANDO MOYA ORSATTI

9,10
(nove e um)

UTILIZAÇÃO DE ALGORITMOS EVOLUTIVOS NA MINIMIZAÇÃO DE VIBRAÇÕES EM UM BRAÇO ROBÓTICO

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia

São Paulo

2002

RICARDO CERQUEIRA DE SOUZA

FERNANDO MOYA ORSATTI

UTILIZAÇÃO DE ALGORITMOS EVOLUTIVOS NA MINIMIZAÇÃO DE VIBRAÇÕES EM UM BRAÇO ROBÓTICO

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia

Área de Concentração:
Engenharia Mecânica/Mecatrônica

Orientador:
Prof. Dr. Agenor de Toledo Fleury

São Paulo

2002

Às nossas famílias.

Agradecimentos

-Ao Prof. Dr. Agenor de Toledo Fleury, orientador deste trabalho, pela fundamental contribuição para nossa formação.

-Aos senhores Paulo Urbano, Ricardo Barbosa Damian e Felipe Ghellar pelas intermináveis discussões na fase mais inicial deste projeto; à William Manjud Maluf Filho pelo trabalho de revisão do texto, pelas sugestões e apoio; à Rodrigo Carareto pelas discussões sobre controle; á Heloisa Callegaro pela paciência e especialmente à senhora Ana Célia Moya Orsatti que manteve as reservas de coca-cola em níveis suficientes para a realização desse trabalho.

-Ao senhor Bar Xaréu pelas ajudas após os esforços; à senhora Regina Maria Cerqueira de Souza por acreditar no trabalho do filho que entrava a madrugada e à senhorita Marlene Klein Saito por tentar entender o escopo do trabalho.

Resumo

Atualmente a engenharia e outros ramos da ciência eventualmente se deparam com problemas que não tem solução ou cuja solução é extremamente complexa. Usualmente desenvolve-se para cada um desses problemas técnicas específicas para sua solução, porém muitas vezes essas soluções não são ótimas.

Os algoritmos evolutivos propõem-se a ser um método de otimização genérico, que busca a otimização de problemas complexos. Assim, esse trabalho é uma tentativa de utilizar a técnica no problema de vibração em um braço robótico cujo perfil laminar o sujeita a grandes vibrações durante movimentos.

Abstract

Nowadays, engineering and other science lines sometimes achieve problems which solutions are extremely complex or even it doesn't exist. Usually solutions to these problems are developed specifically, but in most cases those solutions aren't optimal.

The Evolutionary Algorithms are proposed to be a generic optimization method that search for optimization on complex problems. Thus, this text is an attempt to use the technique on the problem of vibration in a robotic arm whose thin profile subjects it to great vibrations during movements.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

AE – Algoritmos Evolutivos

AG – Algoritmos Genéticos

EE – Estratégias Evolutivas

ES – Evolution Strategies

Lista de Símbolos

x – Vetor de variáveis objetivas para um problema de otimização com EE.

f – Função objetiva do problema de otimização

M – Espaço de possíveis valores de **x** quando levadas em consideração as restrições do problema.

g – Restrições do problema de otimização

$\mathbf{P}^0$ – População inicial para uma EE.

σ – Vetor com os parâmetros estratégicos de uma EE.

m - Operador de mutação.

s - Operador de seleção.

c_d, c_i - Variáveis para controle da mutação.

t - Critério de parada.

N – Distribuição normal de média zero.

μ - Número de indivíduos da população.

Ψ - Número utilizado na operação de recombinação.

Φ - Número utilizado na operação de recombinação.

Δ - Passo da variação de σ na implementação de mutação linear.

λ – Número de descendentes em uma EE.

m - Número de iterações de isolamento das populações.

u - Número de populações desenvolvidas paralelamente

w - Número de populações descendentes das **u** populações originais

v - Número de populações que serão escolhidas para a geração das w populações descendentes.

Kw – Parâmetro do modelo do motor de corrente contínua.

T – Parâmetro do modelo do motor de corrente contínua.

K – Ganho proporcional do controlador PI.

Ti – Constante de tempo integrativa do controlador PI.

Mp – Sobressinal para resposta a degrau.

ts – Tempo de acomodação para resposta a degrau.

y – Deslocamento de um ponto do braço em relação ao referencial flutuante.

θ – Posição angular do braço.

η_1 – coordenada modal do primeiro modo de vibração do braço.

η_2 – coordenada modal do segundo modo de vibração do braço.

ϕ_1 – Função de forma do primeiro modo de vibração.

ϕ_2 – Função de forma do segundo modo de vibração.

$\mathbf{I_{total}}$ – Momento de inércia total do conjunto cubo + eixo.

ω_1 – freqüência natural do primeiro modo de vibração.

ω_2 – freqüência natural do segundo modo de vibração.

T(t) – Torque aplicado pelo motor.

L – Comprimento do braço.

K – Matriz de ganhos do controlador por alocação de pólos.

dm – diferença entre o sobressinal obtido e o desejado.

dt – diferença entre o tempo de acomodação obtido e o desejado.

pico – pico de deslocamento da ponta do braço relativo ao referencial flutuante.

df – diferença entre o ângulo θ obtido e o desejado.

des – deslocamento médio da ponta do braço relativo ao referencial flutuante.

Kp – Ganho proporcional do controlador PID.

Ki – Ganho integrativo do controlador PID.

Kd – Ganho derivativo do controlador PID.

fo – Função objetiva.

Mp – Sobressinal.

ts – Tempo de acomodação.

dp - pico de deslocamento da ponta durante uma manobra completa.

θf – Ângulo final para o braço após movimento completado.

1 INTRODUÇÃO

1.1 Objetivo do Trabalho

Algoritmos Evolutivos (AE) são algoritmos usados, principalmente, para solução de problemas de otimização para os quais não se possa determinar solução com métodos determinísticos normais.

Em geral métodos determinísticos não são suficientes para solução de problemas com funções objetivas não lineares e com muitas variáveis reais [1], ou mesmo para problemas combinatórios, caso o espaço de soluções seja exageradamente grande, como, por exemplo, no problema de corte de placas 2D.

Os Algoritmos Evolutivos tem como princípio fundamental a idéia de imitação da natureza, do ponto de vista evolutivo. Por apresentarem uma série de processos randômicos, como mutação, os AE são um método estocástico de otimização, que em geral são mais efetivos para a solução de problemas do tipo dos citados anteriormente.

Para o estudo, e verificação das possibilidades dos AE, realizar-se-á o controle ótimo de um braço robótico flexível. Para esse problema já existem modelos e resultados obtidos por outros métodos e assim será possível verificar-se a qualidade das respostas encontradas.

1.2 Definição do problema

Tendo em vista que o objetivo do trabalho é o estudo e implementação das teorias de Algoritmos Evolutivos, um problema real foi escolhido para que pudéssemos implementar a solução encontrada através dos algoritmos.

O problema escolhido é a minimização das vibrações na ponta de um braço robótico (fig. 1). Esse braço executa um movimento simples de rotação em torno do eixo de seu cubo. Por ter uma espessura extremamente pequena (de forma a poder ser considerado uma lamina), a vibração causada em sua ponta é considerável. Dessa forma, uma técnica de controle das vibrações se faz necessária. Dado que já existem soluções para o problema (obtidas por técnicas diversas àquela que utilizaremos), os resultados

obtidos poderão ser comparados com os preexistentes. Assim, a cada etapa do desenvolvimento do projeto, teremos em que nos basear de forma a refinarmos nossa solução.

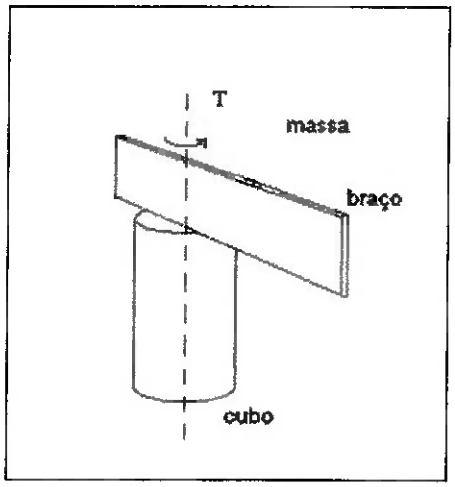


Figura 1: Braço Robótico a ser controlado

O problema consiste na determinação da curva de torque aplicada ao braço, para que esse se movimente, num dado deslocamento, com a menor vibração possível em sua extremidade.

2 ALGORITMOS EVOLUTIVOS

Desde o princípio do desenvolvimento da computação, além da inovadora capacidade computacional, a possibilidade de dotar uma máquina de inteligência sempre instigou os pesquisadores da área de computação. Ao enxergar a possível capacidade destas máquinas de imitar habilidades vivas, como auto-replicação, capacidades de aprendizado, adaptação e controle do meio ambiente surgiram então linhas de desenvolvimento baseadas em atividades biológicas.

Assim, desde os primeiros passos da criação dos computadores até o começo dos anos 80 algumas linhas de atividade em computação seguiram princípios biológicos. A partir de então, estas linhas subdividiram-se e consolidaram-se em "machine learning", redes neurais e computação evolucionária.

A computação evolucionária teve início por volta das décadas de 50 e 60, quando diversos cientistas estudaram a evolução como um método prático de otimização para problemas em engenharia. A idéia consistia em criar-se soluções para um problema específico e evoluí-las, utilizando-se algoritmos e operadores inspirados em métodos naturais de seleção e variação.

Inicialmente três correntes, dentro da computação evolucionária, surgiram de forma independente. Em 1960 Rechenberg introduziu as Estratégias Evolutivas, um método que era utilizado para a otimização de parâmetros em valores reais para dispositivos como os aerofólios. Em 1966 Fogel, Owens e Walsh desenvolveram a "programação evolucionária", onde as soluções para o problema pedido eram representadas por máquinas de estados finitos, cuja evolução consistia em mudar aleatoriamente seus diagramas estado-transição.

Os algoritmos genéticos foram inventados por John Holland também nos anos 60, sendo desenvolvido por ele e seus alunos, na Universidade de Michigan, durante as décadas de 60 e 70. A grande diferença entre a técnica desenvolvida por Holland em comparação com as outras dentro da computação evolucionaria consiste no fato de que Holland buscava estudar formalmente o fenômeno de adaptação e a partir disso desenvolver maneiras para que os mecanismos adaptativos naturais pudessem ser

importados em sistemas computacionais, e não desenvolver um algoritmo que solucionasse um problema específico.

Em 1975 Holland lançou seu livro *Adaptions in Natural and Artifficial Systems*, no qual apresentou os algoritmos genéticos como uma abstração da evolução biológica e deu o embasamento teórico para sua implementação, e mesmo para os outros tipos de AE [2].

Uma última forma de algoritmos evolutivos se desenvolveu mais tarde, chama programação genética, desenvolvida por Koza, porém essa não tem por objetivo a solução de problemas de otimização, e sim a manipulação da sintaxe de programas de computador.

Em uma implementação típica dos Algoritmos Genéticos, os indivíduos são representados por Strings ou Vetores. Todo indivíduo contém informações, que se forem decodificadas, correspondem a uma solução para o problema em questão. O conjunto desses indivíduos é considerado uma população, a qual será submetida ao processo de Evolução. Em outras palavras, esses indivíduos serão selecionados, de forma que os mais aptos para critérios pré-estabelecidos permaneçam na população. Em seguida esses gerarão descendentes, processo no qual é possível a ocorrência de mutações. Os descendentes serão novamente selecionados e assim sucessivamente, até que se obtenha soluções satisfatórias para o problema.

A primeira população é gerada estocasticamente. Essa população é avaliada através da utilização de uma função de adaptação. Ocorre, então, a seleção, de tal forma que os indivíduos mais bem adaptados "sobrevivam", e os menos adaptados "morram"; ou seja, sejam eliminados da população. Depois da seleção os indivíduos selecionados são indicados como pais, cuja replicação gerará uma população de descendentes. Em princípio essa nova população é mais bem adaptada que a anterior, ou seja, os valores da função de adaptação para essa nova população devem ser maiores que os valores para a população anterior. Esse processo deve ser repetido até que um critério de parada seja atingido, ou um limite máximo para o tempo de simulação seja alcançado. A representação típica de um Algoritmo Evolutivo está esquematizada na figura 2 [3].

1. *Escolha dos parâmetros estratégicos*
2. *Inicialização da população inicial P(0)*
3. *t --> 0*
4. *Avaliação da população inicial*
5. *Repetição (Ciclo de gerações)*
6. *t --> t + 1*
7. *Seleção (Escolha dos pais)*
8. *Replicação (geração dos descendentes)*
9. *Mutação*
10. *Avaliação dos descendentes*
11. *Formação da nova população (P(t))*
12. *Até que $t_{máx}$ seja alcançado ou o critério de parada seja satisfeito*
13. *Solução*
14. *Fim*

Figura 2- Implementação típica de AE

Dessa forma Holland introduziu na computação evolucionaria uma técnica com uma formulação teórica consistente, o que serviu a partir de então como base para a maioria dos desenvolvimentos na área.

Os dois segmentos mais importantes de desenvolvimento dos algoritmos evolutivo são os algoritmos genéticos e as estratégias evolutivas, sendo essa resultado de estudos feitos principalmente na Alemanha e aqueles desenvolvidos nos EUA, sendo que esse desenvolvimento ocorreu de forma quase simultânea e praticamente independente. Nas próximas seções serão feitas descrições detalhadas desses dois tipos de AE.

3 ESTRATÉGIAS EVOLUTIVAS[1, 4, 5]

As Estratégias Evolutivas são o resultado do desenvolvimento alemão da utilização do modelo evolutivo da natureza em problemas técnicos. Como mencionado anteriormente, as EE foram desenvolvidas inicialmente por Ingo Rechenberg nos anos 60 na TU Berlin. Mais tarde foram aperfeiçoados, principalmente, por Hans-Paul Schwefel.

A primeira diferença fundamental entre as Estratégias Evolutivas e os Algoritmos Genéticos reside no grau de formalismo da imitação da natureza. Ao contrário dos AG, as EE, segundo o próprio Rechenberg, estão baseadas na idéia de que uma cópia fiel dos fenômenos naturais não é estritamente necessária para que se obtenham resultados satisfatórios em problemas técnicos.

Como conseqüência foram desenvolvidas várias formas para a implementação das EE, que levam em consideração diferentes detalhes do processo evolutivo. A descrição dessas formas principais será feita a seguir.

A segunda grande diferença entre os AG e as EE é conseqüência da primeira. Os AG necessitam que se faça uma codificação, em geral binária, dos parâmetros do problema. Em geral é necessário fazer-se uma aproximação por números inteiros para variáveis contínuas, o que causa perda de resolução na resposta, além de aumentar bastante o esforço computacional. Essa exigência dos AG está atrelada ao formalismo com o qual esse copia a natureza. Para as EE, em geral, codifica-se variáveis contínuas com números reais. Além de utilizar os recursos do computador de maneira mais eficiente, essa opção faz com que a entrada e saída de dados sejam mais claras [1].

3.1 Descrição do problema de otimização

Em geral, o objetivo de um problema de otimização é encontrar um vetor $\mathbf{x}^*$ tal que:

$$\forall x \in M : f(x) \geq f(x^*) = f^* \qquad (1)$$

x: Vetor de variáveis objetivas do problema. **x** é um vetor de n componentes, sendo n o número de dimensões do problema.

f: Função objetiva do problema. Função que deve ser minimizada.

$\mathbf{f}^*$: Mínimo global da função **f**.

M: Sub-espaço de $\mathbf{R}^n$, que representa todos os valores que **x** pode assumir. **M** pode ser definido da seguinte maneira:

$$M = \{x \in R^n / g_j(x) \geq 0 \forall j \in \{1,...,q\}\} \quad (2)$$

M é um conjunto de pontos limitado pelas restrições do problema, que podem ser expressas na forma de inequações.

É importante ressaltar que a definição do problema como um problema de minimização não causa perda alguma de generalidade, pois este pode, facilmente, ser transformado em um problema de maximização. E, por isso, não haverá mais preocupação a esse respeito no texto.

3.2 Estratégias Evolutivas com dois indivíduos

Esse é o tipo mais simples de implementação de EE, e foi o primeiro a ser utilizado para solução de problemas, já na década de 60. A notação normalmente utilizada para representação desse tipo de implementação é: (1+1)-ES (utiliza-se a sigla ES nesse ponto para conformidade com a nomenclatura mais comum). O motivo dessa notação ficará claro ao longo do texto.

Em um (1+1)-ES, tem-se apenas um indivíduo (vetor de valores reais) que compõe a população. Esse indivíduo é então duplicado e essa cópia sofre mutações, obtendo-se assim um segundo indivíduo (descendente). Esses dois indivíduos são então avaliados pela função objetiva, e o melhor deles é mantido para a próxima população, sendo o outro descartado. Em seguida reinicia-se o ciclo: o indivíduo selecionado é novamente duplicado, a cópia sofre mutação e assim sucessivamente até que se atinja um critério de parada.

Uma possível representação matemática para essa implementação pode ser feita da seguinte maneira:

$(1+1)\text{-ES} = (P^0, m, s, c_d, c_i, f, g, t)$

$\mathbf{P^0} = (\mathbf{x^0}, \sigma^0) \in I$: População inicial. $I = R^n \times R^n$

x: variável objetiva do problema

σ: parâmetro estratégico do método

m: Operador de mutação. $I \rightarrow I$

s: Operador de seleção. $I \times I \rightarrow I$

$c_d, c_i \in R$: Variáveis para controle da mutação.

f: Função objetiva. $R^n \rightarrow R$

$\mathbf{g_j}$: Funções de restrições. $R^n \rightarrow R$. $j \in \{1,...,q\}$

t: Critério de parada. $I \times I \rightarrow \{0,1\}$

É importante notar que **n** é o número de variáveis a serem ajustados no problema, ou seja, o número de variáveis objetivas do problema de otimização.

P^0 representa a população inicial, que pode ser gerada aleatoriamente pelo próprio algoritmo, ou definida pelo operador como um chute inicial. A geração do segundo indivíduo (descendente) a partir do primeiro é feita da seguinte maneira:

$$\begin{aligned} P'^t &= (a_1'^t, a_2'^t) \in I \times I \\ a_1'^t &= P^t = (x^t, \sigma^t) \\ a_2'^t &= m(P^t) = (x'^t, \sigma'^t) \end{aligned} \qquad (3)$$

O conjunto das equações (3) representa a formação de uma população com dois indivíduos (P'^t), no instante t, a partir da população original no mesmo instante t (P^t), composta por apenas um indivíduo. A população P'^t é temporária, ela é formada apenas para que sobre ela possa ser aplicado operador de seleção.

Sabe-se que na evolução natural, alterações pequenas causadas por mutação, em geral, são mais eficientes para obtenção de melhorias, que grandes alterações. Dessa forma, o operador de mutação é definido de tal maneira, que normalmente as mutações causem pequenas alterações nos indivíduos originais.

$$x_i'^{t} = x_i'^{t} + N(0, \sigma_i'^{t}) \quad (i = 1,\ldots,n) \tag{4}$$

N é um número gerado segundo uma distribuição normal, com média zero e desvio padrão σ. Nessa definição pode-se perceber dois fatores importantes: Apenas o valor da variável objetiva do indivíduo sofre mutação. O valor do desvio padrão associado a ele, e que também faz parte do indivíduo permanece constante durante todo o processo. O segundo ponto importante é que se podem definir diferentes desvios padrão para cada variável do problema. Isso pode ser bastante útil, quando a faixa de variação possível dos valores for muito diferente para cada variável. Nesse caso, as mutações podem ser proporcionais aos tamanhos dessas faixas, ou seja o passo da mutação é variado de acordo com o valor de σ. Denomina-se esse tipo de mutação por mutação linear e é discutido com mais detalhe na seção 3.7.3.

O próximo passo é a seleção do indivíduo mais apto, que será mantido para a próxima geração. Isso é feito através da aplicação do operador **s** sobre a população P'^{t}.

$$P^{t+1} = s(P'^{t}) = \begin{cases} a_2'^{t} \cdot se\{f(x'^{t}) \le f(x^{t}) \wedge g_j(x'^{t}) \ge 0 \forall j \in \{1,\ldots,q\} \\ a_1'^{t} \cdot caso \cdot contrário \end{cases} \tag{5}$$

A iteração $\mathbf{P^t \rightarrow P^{t+1}}$ pára quando o critério de parada for atingido, ou seja:

$$t(a_1'^{t}, a_2'^{t}) = 1 \tag{6}$$

Os critérios de parada mais usuais para o processo são: número de gerações, tempo de processamento, progresso (absoluto ou relativo), atingido.

Um dado importante para o (1+1)-ES, é que sua convergência global pode ser demonstrada, com algumas hipóteses simplificadoras. A demonstração encontra-se em [5]. Para os fins desse trabalho, essa demonstração não é de fato importante, porém ilustra o grau de formalismo atingido pelas Estratégias Evolutivas nesses 40 anos de desenvolvimento.

O (1+1)-ES como definido até aqui apresenta ainda um grande problema. Os parâmetros estratégicos σ são definidos no começo do processo, e seus valores não serão alterados a partir de então. Como pode ser visto na equação (4) σ indica o passo da variação causada a x pela operação de mutação.

No começo do processo os valores de x estão bastante distantes dos valores ótimos, os quais se objetiva alcançar, pode-se, portanto, definir um passo relativamente grande para a mutação, assim, chegar-se-á mais rápido às proximidades do ótimo global. Entretanto um passo muito grande para a mutação inviabiliza o refinamento da solução, que exige passos menores. Por outro lado, se o passo for pequeno desde o início o processo será bastante lento. Por esse motivo é necessário que se determine uma forma de se fazer um ajuste dinâmico dos parâmetros estratégicos durante o processo de busca.

Para a realização desse ajuste, Rechenberg fez estudos teóricos a respeito da taxa ótima de progresso durante a otimização de alguns casos clássicos, e seus resultados podem ser expressos pelo que ficou conhecido como "Regra do 1/5 de sucesso de Rechenberg", definida por ele mesmo por:

> *A taxa de mutações que causam melhora da solução em relação ao número total de mutações deve ser de 1/5. Se essa taxa for maior que 1/5, o desvio padrão deve ser aumentado, caso contrário deve ser diminuído.*

É evidente que a maioria dos problemas de aplicação tem características diferentes daqueles analisados por Rechenberg, porém essa regra se mostra bastante eficiente como possibilidade de ajuste dinâmico desse parâmetro.

Schwefel propõe a seguinte forma para implementar-se a regra:

$$\sigma^{t+k} = \begin{cases} c_d \cdot \sigma^t, se \cdot p_s^t < 1/5 \\ c_i \cdot \sigma^t, se \cdot p_s^t > 1/5 \\ \sigma^t, se \cdot p_s^t = 1/5 \end{cases} \qquad (7)$$

Na equação (7), p^t_s é a taxa de mutações com sucesso em relação ao total de mutações, medida a cada k gerações. A alteração dos valores do parâmetro estratégico é feita, então, a cada k gerações. Schwefel sugere os seguintes valores para c_i e c_d:

$c_i = 1/0.82$

$c_d = 0.82$

Além disso, pela própria expressão de (7) verifica-se que o desvio padrão para todas as variáveis objetivas do problema é alterado da mesma forma.

3.3 A forma mais simples de EE com vários indivíduos

Um próximo passo no desenvolvimento das estratégias evolutivas foi o desenvolvimento de sua primeira implementação utilizando o conceito de população formada por um conjunto de indivíduos. Essa implementação, representada pela sigla (μ+1)-ES, introduz a idéia de se ter uma população composta de μ indivíduos, que gerarão apenas um descendente. Se μ for igual a um, teremos novamente um (1+1)-ES.

Esse tipo de estratégia evolutiva não é, de fato, muito utilizada, porém a compreensão dessa é de grande valia para o estudo dos tipos mais complexos de EE's, que serão discutidos em seguida.

Semelhantemente ao (1+1)-ES, o (μ+1)-ES pode ser definido da seguinte maneira:

$(\mu+1)\text{-ES} = (P^o, \mu, r, m, s, c_d, c_i, f, g, t)$

$\mathbf{P^0} = (\mathbf{a_1}^o, ..., \mathbf{a_\mu}^o) \in I^\mu$: População inicial. $I = R^n \times R^n$

μ: Número de indivíduos da população

r: Operador de recombinação. $I^{\mu} \rightarrow I$

m: Operador de mutação. $I \rightarrow I$

s: Operador de seleção. $I^{\mu+1} \rightarrow I^{\mu}$

c_d, $c_i \in R$: Variáveis para controle da mutação

f: Função objetiva. $R^n \rightarrow R$

$\mathbf{g_j}$: Funções de restrições. $R^n \rightarrow R$. $j \in \{1,...,q\}$

t: Critério de parada. $I^{\mu+1} \rightarrow \{0,1\}$

As únicas diferenças desse tipo de EE em relação à implementação com apenas um indivíduo são o número de indivíduos, evidentemente, e a introdução do operador de recombinação.

A operação de recombinação tem dois passos básicos: o primeiro consiste na escolha dos dois indivíduos que serão utilizados para a geração do descendente. O segundo passo é a geração do descendente de fato.

A escolha dos pais, por convenção, é feita de maneira totalmente aleatória. Com os pais escolhidos, uma possível implementação para a geração do descendente é a seguinte:

$$
\begin{aligned}
& r(P^t) = a' = (x', \sigma') \in I, x' \in R^n, \sigma' \in R^n \\
& x'_i = \begin{cases} x_{a,i}, \Psi \leq 1/2 \\ x_{b,i}, \Psi > 1/2 \end{cases}, \forall i \in \{1,...,n\} \qquad (8) \\
& \sigma'_i = \begin{cases} \sigma_{a,i}, \Phi \leq 1/2 \\ \sigma_{b,i}, \Phi > 1/2 \end{cases}, \forall i \in \{1,...,n\}
\end{aligned}
$$

O conjunto das equações (8) mostra uma implementação do operador de recombinação, que é chamado de recombinação discreta, já que as componentes do vetor são meramente copiadas para o descendente.

Os números Ψ e Φ são gerados randomicamente no intervalo [0,1] para cada componente do indivíduo, dessa forma eles são gerados n vezes para a recombinação de um único indivíduo.

Após a recombinação, ao descendente é aplicado o operador de mutação, e somando o descendente gerado ao conjunto dos μ indivíduos que compunham a população no instante t, teremos uma nova população, no mesmo instante t, com μ+1 indivíduos, e que será submetida à seleção. Então:

$$P'^{t} = (a_1'^{t},...,a_{\mu}'^{t},a_{\mu+1}'^{t}) = (a_1'^{t},...,a_{\mu}'^{t},m(r(P^{t}))) \qquad (9)$$

A seleção nesse tipo de EE é feita eliminando o indivíduo menos apto da população P'^{t}. Por isso, a população volta a ter μ indivíduos e o processo pode ser reiniciado.

$$P^{t+1} = s(P'^{t}) \qquad (10)$$

O processo de mutação é definido exatamente da mesma maneira que para o (1+1)-ES e a regra de 1/5 de sucesso de Rechenberg também pode ser implementada nessa situação.

Os (μ+1)-ES são em geral mais rápidos que os (1+1)-ES, e apresentam uma vantagem fundamental em relação a eles: como existe uma população inicial com vários indivíduos, a chance de um deles ser um bom chute inicial, quando da geração da mesma é bastante grande, melhorando bastante o tempo de processamento do algoritmo.

A única complicação introduzida é que o parâmetro μ deve ser definido pelo operador, e até aqui não existem estudos teóricos suficientemente amplos para a determinação do mesmo, sendo assim necessário um processo de tentativa e erro, buscando o melhor ajuste do parâmetro.

3.4 As estratégias evolutivas do tipo (μ+λ)-ES e (μ,λ)-ES

Após a apresentação do (1+1)-ES e do (μ+1)-ES, pode-se esclarecer a razão da notação utilizada até aqui. O primeiro número entre os parênteses representa o número de indivíduos que compõem a população. O segundo número representa o número de descendentes gerados a cada geração. Sendo para tanto usados os símbolos μ e λ, respectivamente, para denominá-los.

Nessa seção serão apresentados mais dois tipos de estratégias evolutivas: (μ+λ)-ES e (μ,λ)-ES. Sendo assim, ambos os tipos têm μ indivíduos por geração, dos quais serão gerados λ descendentes. A diferença entre os dois está na operação de seleção.

Para o (μ+λ)-ES os μ indivíduos e os λ descendentes são agrupados, formando uma população maior, à qual será aplicado o operador de seleção, ou seja, da qual serão escolhidos os μ indivíduos mais aptos, formando assim a população para a próxima geração.

Para o (μ,λ)-ES somente os λ descendentes formam a população à qual será aplicado o operador de seleção. Os indivíduos que compunham a população original são desprezados e a seleção é feita exclusivamente sobre os descendentes gerados. Por esse motivo, λ deve ser sempre maior que μ.

A diferença fundamental entre os dois tipos de EE é que o (μ+λ)-ES garante que nunca haverá diminuição da qualidade das soluções, o que não é certo o (μ,λ)-ES.

Esses dois tipos de EE se diferenciam fortemente dos anteriores, pois para eles não será utilizado o operador de mutação como definido até aqui e nem a regra de 1/5 de sucesso de Rechenberg. Para essas implementações será utilizada a idéia da auto-adaptação dos parâmetros estratégicos; introduzidas por Schwefel.

Schwefel propôs uma forma alternativa para o ajuste dinâmico do parâmetro σ^t. Baseado na idéia de que o desvio padrão associado a cada componente do indivíduo é parte do mesmo, o autor propõe que o desvio padrão também possa ser alterado pelo operador de mutação. Isso poderia ser feito conforme:

$$\sigma_{novo} = \sigma_{antigo} \cdot e^{N(0,\Delta)}$$
$$x_{novo} = x_{antigo} + N(0,\sigma_{novo}) \qquad (12)$$

Na verdade essa maneira alternativa de implementação da mutação força que o desvio padrão (passo da variação) seja também objeto da seleção. Assim, desvios padrão mais "adaptados" são selecionados e melhoram as características da solução encontrada. A vantagem dessa implementação, é que ela se adapta às características do problema em questão, ao contrário da regra do 1/5 de sucesso de Rechenberg. Em suma, a parir desse ponto os parâmetros estratégicos, assim como as variáveis objetivas, passam a ser objeto de processo evolutivo.

O único problema desse tipo de abordagem é, novamente, a introdução de um novo parâmetro para o algoritmo, Δ, que representa o passo da variação de σ. Nesse caso, também não existem estudos teóricos que forneçam valores para esse parâmetro. Deve-se determiná-lo empiricamente, voltando ao problema mencionado anteriormente.

Outras implementações para a mutação estão disponíveis na literatura, mas esbarram sempre no mesmo problema. Para uma discussão aprofundada sobre o tema ver item 3.7.3.

Neste ponto da discussão já estamos aptos a entender as vantagens e desvantagens do (μ+λ)-ES em relação ao (μ,λ)-ES. É possível demonstrar que em algumas situações, o (μ+λ)-ES apresenta vantagem seletiva para indivíduos que tiverem seus desvios padrão diminuídos, levando à estagnação do processo. Além disso, em problemas com ótimos variando no tempo esse tipo de EE tende a favorecer a permanência de ótimos antigos.

O fato de o (μ,λ)-ES não garantir a preservação dos indivíduos mais aptos de cada geração, e sim selecionar os mais aptos do conjunto dos descendentes, pode levar a algumas fases de recessão, porém evita certamente os longos períodos de estagnação.

Dois gráficos bastante representativos da evolução típica desses dois tipos de EE em relação ao tempo estão representados na figura a seguir:

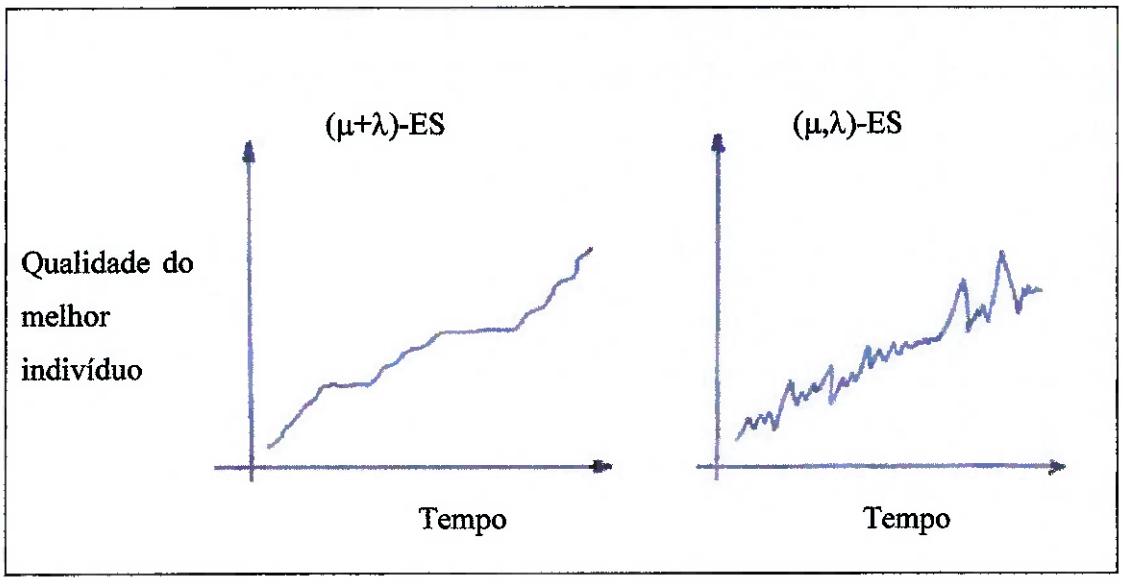


Figura 3- Gráfico comparativo da evolução das soluções nas estratégias evolutivas

A descrição formal de (μ,λ)-ES pode ser feita da seguinte maneira:

(μ,λ)-ES = (P^o, μ,λ,r,m,s,Δ,f,g,t)

$\mathbf{P^o} = (\mathbf{a}_1^o,...,\mathbf{a}_\mu^o) \in I^\mu$: População inicial. $I = R^n \times R^n$

μ: Número de indivíduos da população

λ: Número de descendentes por geração

r: Operador de recombinação. $I^\mu \rightarrow I$

m: Operador de mutação. $I \rightarrow I$

s: Operador de seleção. $I^\lambda \rightarrow I^\mu$

$\Delta \in R$: Variável para controle do passo de variação de σ^t

f: Função objetiva. $R^n \rightarrow R$

$\mathbf{g_j}$: Funções de restrições. $R^n \rightarrow R$. $j \in \{1,...,q\}$

t: Critério de parada. $I^\mu \rightarrow \{0,1\}$

O desenvolvimento do algoritmo é feito usando as mesmas formas de **s** e **r** usadas para o (μ+1)-ES, e o operador de mutação é implementado como em (12). A

diferença é que na operação de recombinação serão gerados λ descendentes, ou seja, ela será executada λ vezes.

3.5 As estratégias evolutivas do tipo (μ/ρ#λ)-ES

Esse tipo de estratégia evolutiva é uma extrapolação do modelo encontrado na natureza. Na notação, '#' pode representar tanto '+' como ','.

Nesse tipo de implementação defini-se mais um parâmetro ρ, que representa o número de indivíduos selecionados como "pais" para a geração de cada descendente. Ao invés da escolha normal de dois indivíduos, ter-se-á um descendente sendo formado a partir das variáveis e dos parâmetros de ρ indivíduos. A implementação desse tipo de EE pode ser feita alterando-se apenas a operação de recombinação, que continua sendo feita de maneira muito parecida com aquela descrita pelas equações (8).

3.6 Estratégias evolutivas para populações

É possível estender o conceito da implementação das estratégias evolutivas para a evolução de populações inteiras de solução. São geradas algumas populações de indivíduos que se desenvolvem paralelamente. De tempos em tempos ocorre uma troca de informação entre essas populações. Essa troca pode ser apenas de componentes dos indivíduos, ou mesmo de indivíduos inteiros.

Esse tipo de implementação é bastante interessante para que se tente evitar a obtenção de um mínimo local para o problema. Esse é um problema bastante presente nas estratégias evolutivas, dado que quando um indivíduo se aproxima de um mínimo local, se torna muito mais apto que os outros componentes da população. Por conseqüência os valores de suas variáveis tendem a se espalhar pela população, diminuindo sua diversidade e conseqüentemente causando sua estagnação e a impossibilidade de atingir-se um mínimo global.

Uma notação completa de todas as possibilidades de implementação das EE pode ser feita da seguinte maneira:

$[u/v\#w(\mu/\rho\#\lambda)^{/m}]$-ES

m: Número de iterações de isolamento das populações.

u: Número de populações desenvolvidas paralelamente

w: Número de populações descendentes das u populações originais

v: Número de populações que serão escolhidas para a geração das w populações descendentes

μ: Número de indivíduos de cada população

ρ: Número de indivíduos utilizados para a geração de cada descendente

λ: Número de descendentes em cada população

O processo inicia-se com a geração de u populações com μ indivíduos cada uma. Durante m gerações essas populações se desenvolvem normalmente e de maneira totalmente independente. A cada m gerações ocorre a geração de w populações descendentes, sendo cada uma delas formada a partir de v populações presentes no conjunto. Dessas w populações seleciona-se u populações que se desenvolverão independentemente por mais m gerações, e assim por diante.

Para a escolha das populações mais aptas utiliza-se, em geral, a médias dos valores obtidos pela função objetiva para os indivíduos que compõem cada uma delas. A formação de novas populações a partir de outras não segue nenhum padrão que seja dominante. Pode-se faze-lo misturando-se os indivíduos das populações genitoras usando qualquer critério que se deseje.

3.7 Aprofundamentos

3.7.1 O conceito da carga genética

Uma segunda possibilidade para evitar-se a estagnação em um mínimo local – sendo a primeira possibilidade a implementação de estratégias evolutivas para populações – é a utilização do conceito de carga genética, introduzido por Joachim Born.

A idéia fundamental desse tipo de implementação é a definição de uma carga genética básica, ou seja, um certo número de indivíduos, que deve estar sempre presente na população. Essa carga genética é utilizada para as operações de recombinação, entretanto ela não é influenciada pela operação de seleção. Esses indivíduos nunca são substituídos por descendentes.

O objetivo principal desse conceito é a introdução de um conjunto de informações sobre bons valores para as variáveis e para os parâmetros do problema, que sejam conhecidos com antecedência, e que, portanto, não precisarão ser aprendidos pelo algoritmo.

Em resumo, nesse tipo de implementação tem-se cada geração uma população constituído por elementos pré-definidos e por elementos que compõem a população normal, aqueles que sofrem mutação, recombinação e que estão submetidos à seleção.

3.7.2 Recombinação

Até aqui se utilizou para todos os tipos de EE apenas uma possibilidade de implementação para a operação de recombinação, chamada de recombinação discreta e descrita pelas equações (8). Entretanto esse é um ponto bastante importante para a eficiência do método. A operação de recombinação, junto com a mutação, é o grande mecanismo de introdução de diversidade na população, e, portanto, parte fundamental do processo de busca pela melhor solução.

Atualmente utilizam-se cinco tipos de operação de recombinação nas estratégias evolutivas, que podem ser definidas como:

$$x_i' = \begin{cases} x_{a,i}, (A) \\ x_{a,i} \cdot ou \cdot x_{b,i}, (B) \\ \frac{1}{2}(x_{a,i} + x_{b,i}), (C) \\ x_{a,i} \cdot ou \cdot x_{bi,i}, (D) \\ \frac{1}{2}(x_{a,i} + x_{bi,i}), (E) \end{cases} \qquad (13)$$

Para todos os casos, a, b e b_i são indivíduos escolhidos como pais pelo próprio operador de recombinação, e devem fazer parte da população atual. Vale lembrar, que i deve ser um número no intervalo {1,...,n}, sendo n o número de variáveis do problema de otimização.

(A): Na verdade esse caso representa a situação de não se fazer recombinação, ou seja, o descendente é copiado exatamente como o "pai".

(B): O caso B representa a recombinação discreta, como definida em (8).

(C): Esse tipo de recombinação é chamada *intermediária*. Sua implementação é motivada pela idéia de se ter uma situação com duas soluções em lados opostos do "pico ótimo". Nessa situação a média dos valores dos parâmetros pode levar a resultados excelentes. Além disso, esse tipo de recombinação tende a aumentar os valores de σ, impedindo assim a estagnação prematura da busca.

(D) e (E): Os casos D e E são muito semelhantes aos casos B e C, respectivamente. A diferença fundamental é que o segundo "pai" escolhido por **r** não é sempre o mesmo, sendo diferente para cada variável do problema. A grande vantagem dos casos D e E, é o alto grau de mistura conseguido dentro da população. Esses casos são uma possibilidade de implementação para os algoritmos propostos na seção 3.5.

Todos esses tipos de recombinações foram testados por Schwefel para vários problemas, e os melhores resultados foram obtidos com a utilização de recombinação discreta para as variáveis objetivas do problema (x), e recombinação intermediária para os parâmetros estratégicos do método (σ).

3.7.3 Mutação[6]

Como mencionado anteriormente, a recombinação e a mutação são os dois agentes introdutórios de diversidade na população, e, portanto, daremos especial atenção para as possibilidades de implementação das mutações nessa seção.

3.7.3.1 Mutação simples

A forma mais simples de implementação da mutação – não mencionada no texto até aqui – é conhecida como mutação simples. Para esse tipo de implementação, define-se um único valor para o parâmetro estratégico σ. Dessa forma, a equação (4) pode ser reescrita da seguinte forma:

$$x_i'^{t} = x_i^{t} + N(0,\sigma^{t}) \quad \text{i} = (1,...,n) \qquad (14)$$

Nesse caso, temos apenas um valor de σ para a população inteira. Esse valor pode ser alterado pela regra de 1/5 de sucesso de Rechenberg.

Para mutação simples, o conjunto de possíveis posições que um indivíduo pode ocupar, dentro do espaço de busca da melhor solução ($\mathbf{R^n}$), após sofrer mutação, é "esférico". Essa "esfera" tem centro na posição atual do indivíduo e raio proporcional ao valor de σ. Nesse caso o caminho realizado pelas soluções até o ótimo não é o ideal (gradiente) e sim fica oscilando em torno dele. Em problemas nos quais os gradientes para cada uma das coordenadas (componentes) sejam muito diferentes, não se consegue que o parâmetro estratégico (σ) se adapte adequadamente.

A figura a seguir mostra uma interpretação gráfica do espaço das possíveis posições de um indivíduo depois da mutação (a esquerda) , e o caminho das soluções até o ótimo para mutação simples. È considerado um problema de duas dimensões.

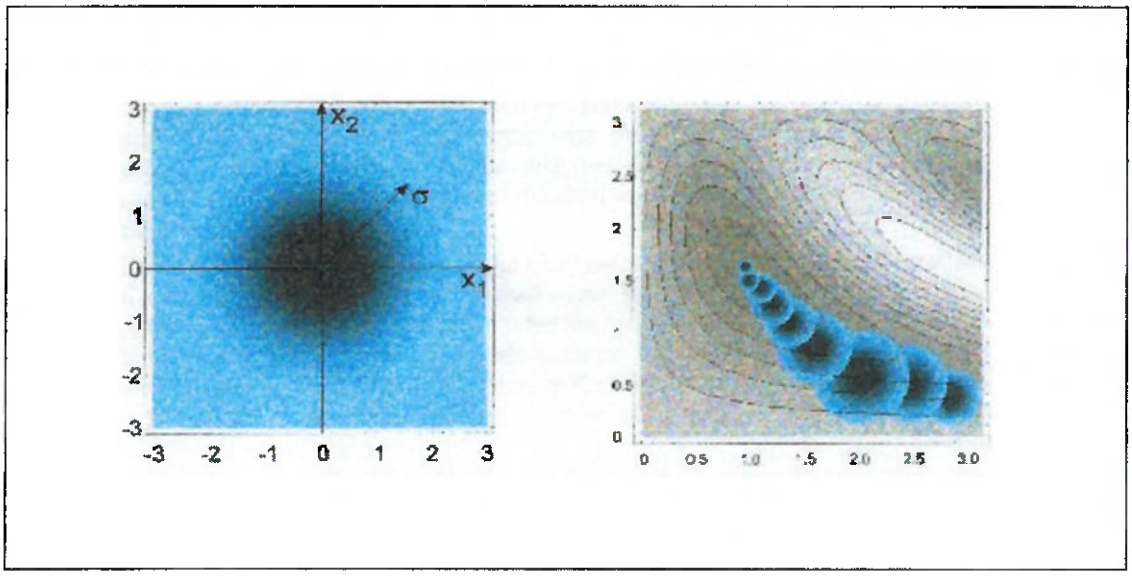


Figura 4- Visualização gráfica da mutação simples

3.7.3.2 Mutação linear

A mutação linear é a forma de mutação utilizada até aqui para todas as formas de EE. Nesse caso associa-se a cada variável objetiva de cada indivíduo um parâmetro estratégico, que indica o passo da variação dessa variável durante a mutação. Esse tipo de mutação é normalmente implementado por (12).

A grande diferença desse tipo de mutação é a inclusão do parâmetro estratégico como objeto da evolução, ou seja, tanto da própria mutação, como da recombinação e da seleção.

Nesse caso o espaço de possíveis posições para um indivíduo que sofre mutação em $\mathbf{R}^n$ pode ser imaginado como um "elipsóide", sendo as dimensões de seus semi-eixos – que são paralelos aos eixos coordenados – proporcionais aos valores de σ para cada uma das variáveis objetivas.

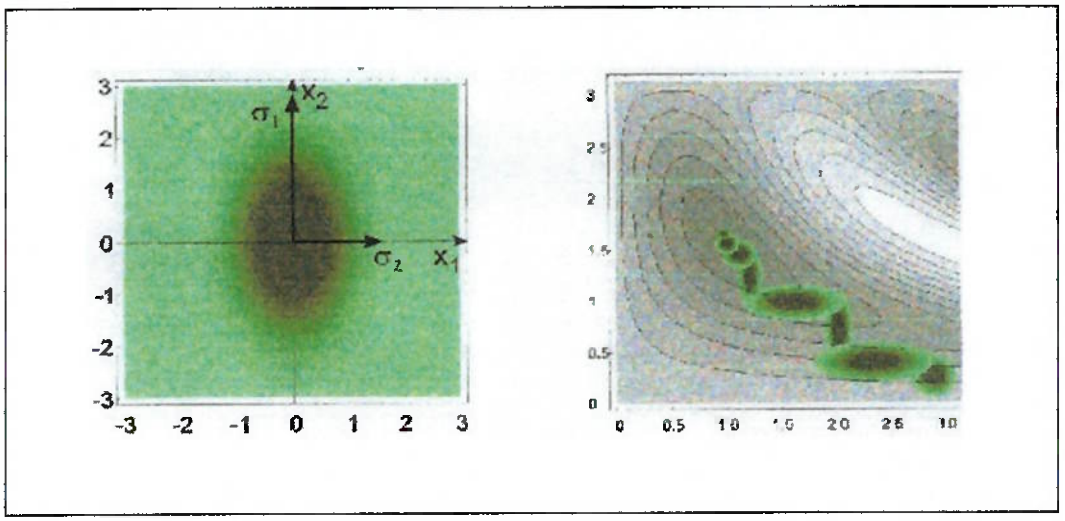


Figura 5- Visualização gráfica da mutação linear

Nesse tipo de implementação adiciona-se ao sistema um parâmetro, Δ, que deve ser ajustado empiricamente. A figura 5 mostra, de maneira semelhante à figura 4, uma interpretação gráfica para a mutação linear.

3.7.3.3 Mutação correlacionada

A última, e mais complexa, possibilidade de implementação da mutação é chamada mutação correlacionada. Nesse tipo de mutação define-se um parâmetro estratégico σ para cada indivíduo, assim como para mutação linear. Entretanto, define-se também um outro parâmetro estratégico θ, que é o ângulo entre as direções dos passos de variação para cada variável e as direções dos eixos coordenados.

Uma interpretação para esse tipo de mutação, é que o espaço de possíveis posições que o indivíduo mutado pode ocupar é um "elipsóide", como no caso anterior, porém aqui se define um ângulo de inclinação do mesmo, em relação aos eixos coordenados estabelecidos para o problema. A grande vantagem desse tipo de mutação, é que a busca pela melhor solução é feita, ou tende a ser, exatamente na direção do gradiente da função, ou seja, com o menor número de iterações possíveis.

Para essa situação, cada indivíduo pode ser representado por $\mathbf{I} = (\mathbf{x},\boldsymbol{\sigma},\boldsymbol{\theta})$, sendo que $\mathbf{x}$ tem n componentes, assim como σ. Entretanto θ tem $n(n-1)/2$ componentes, já que o mesmo é definido para cada par de variáveis objetivas, ou melhor, para cada par de direções definidas pelo problema de n dimensões.

Vale ressaltar que o parâmetro θ é também objeto do processo evolutivo, e é variado de maneira análoga à variação de σ, como proposto em (12). A figura 6 mostra uma interpretação gráfica para a mutação correlacionada para um problema de duas dimensões.

Para a forma de implementação de mutação correlacionada ver [7], e para uma forma alternativa de mutação ver [8].

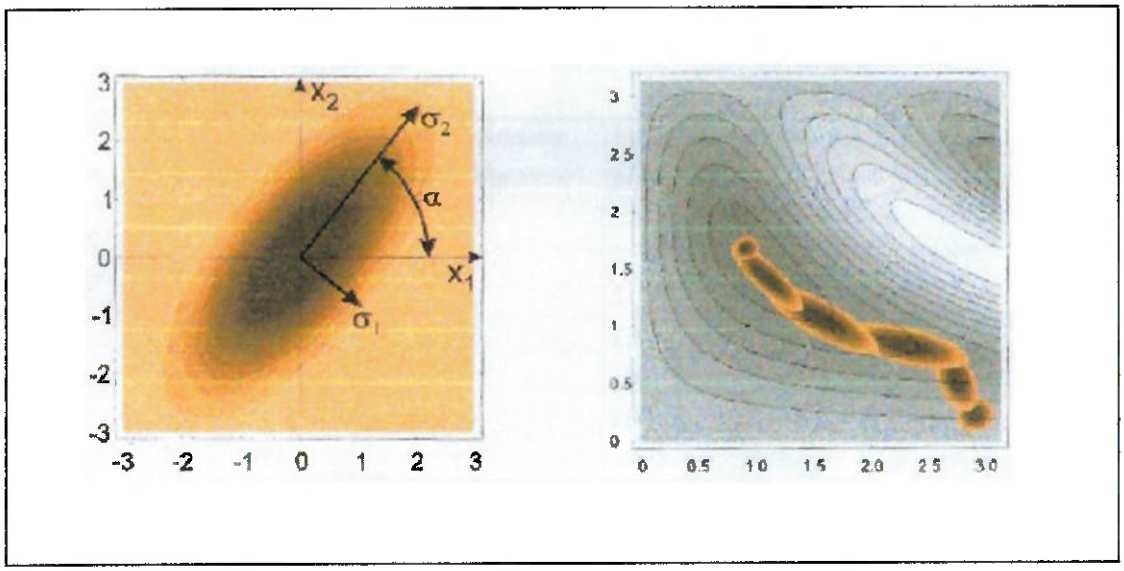


Figura 6: visualização gráfica da mutação correlacionada

3.7.3.4 Comparação entre as formas de mutação

Para a mutação simples temos apenas um parâmetro estratégico definido para o indivíduo inteiro. Para a mutação linear temos um número de parâmetros estratégicos igual ao número de dimensões do problema, ou seja, o número de variáveis objetivas n. Finalmente para a mutação correlacionada temos n parâmetros σ e mais $n(n+1)/2$

parâmetros θ. Dessa forma o esforço computacional exigido pela mutação correlacionada cresce de forma quadrática com a dimensão do problema,e , portanto, é inviável para problemas de muitas dimensões.

Por outro lado a mutação correlacionada é a que tem melhores possibilidades de adaptação às condições de um problema, sendo assim a preferida para a maior parte das aplicações.

3.8 Conclusões sobre as estratégias evolutivas

As estratégias evolutivas são concebidas em sua forma mais complexa, (μ,λ)-ES – desconsiderando a utilização de estratégias evolutivas para populações -, de tal forma que ocorre um processo de aprendizado em dois níveis.

O primeiro nível de aprendizado é a evolução das variáveis objetivas para valores que representem um mínimo da função – de preferência um mínimo global. O aprendizado no segundo nível se dá justamente quanto aos parâmetros estratégicos, que evoluem para valores que possibilitem a evolução das variáveis objetivas de maneira ótima. Isso é feito ao se incluir os parâmetros estratégicos como objetos do processo evolutivo, ou seja, eles mesmos passam a ser selecionados de maneira que valores mais adequados tendem a permanecer nas populações, e sem que seja necessário nenhum controle externo sobre os mesmos, como, por exemplo, a regra de 1/5 de sucesso.

Esse aprendizado em dois níveis é a diferença fundamental entre as estratégias evolutivas e os algoritmos genéticos, e deve ser levado em consideração quando da escolha do tipo de algoritmo evolutivo a ser utilizado em determinada aplicação.

4 ALGORITMOS GENÉTICOS

Dentre a variedade de técnicas fornecidas pela computação evolutiva, um dos casos que merece maior estudo, dada a capacidade de resolver o problema de vibração do braço robótico, são os algoritmos genéticos. A seguir, um estudo mais aprofundado sobre o tema.

4.1 Introdução e breve histórico

O fim da década de 50 e começo da década de 60 foi bastante fértil no desenvolvimento da computação evolutiva, principalmente por causa do surgimento e do desenvolvimento de máquinas de alto poder computacional, os computadores. Dentro desse cenário, foram criadas algumas das técnicas existentes até hoje de computação evolutiva. Uma delas, surgida na década de 60, foram os algoritmos genéticos.

Os algoritmos genéticos foram inventados por John Holland nos anos 60, sendo desenvolvido por ele e seus alunos, na Universidade de Michigan, durante as décadas de 60 e 70. O procedimento até então na computação evolutiva era o desenvolvimento dos métodos focados na solução de um problema específico, portanto a área era carente de alguma base teórica. Mas o que John Holland e seus alunos buscavam consistia em (1) estudar e explicar rigorosamente os processos adaptativos naturais e (2) projetar sistemas artificiais por software que contivessem os mecanismos importantes de sistemas naturais, como crossover, mutação e reprodução [9].

Em 1975 Holland lançou seu livro *Adaptions in Natural and Artifficial Systems*, no qual apresentou os algoritmos genéticos como uma abstração da evolução biológica e deu o embasamento teórico para sua implementação, e mesmo para os outros tipos de AE [2].

4.2 Definição

Algoritmos genéticos são algoritmos de busca de soluções baseados nos mecanismos de seleção natural e da genética. Eles combinam a teoria de seleção natural (sobrevivência do indivíduo mais apto) com as técnicas de troca de informação entre indivíduos, reproduzindo as etapas de reprodução dos seres vivos.

Assim, tipicamente,uma implementação por algoritmos genéticos consiste em se desenvolver uma população, ou um conjunto de indivíduos, que interagirá entre si para escolher (e desenvolver) os indivíduos mais aptos, que trocarão informação entre si de forma a formar uma nova geração cujos indivíduos contenham informação dos indivíduos mais aptos da geração passada.

Um indivíduo, no caso, representa uma possível solução para o problema proposto. Consiste, geralmente, em um vetor (string) de valores, geralmente binários, que codificam uma solução. Portanto, dado o problema, primeiramente deve-se desenvolver uma codificação para as informações que as possíveis soluções devem conter, de forma que ela possa ser representada por uma seqüência de valores binários. Por analogia à biologia, onde toda a informação necessária de cada indivíduo está codificada numa seqüência de informações (as seqüências de bases nitrogenadas que constituem o RNA ou o DNA), os algoritmos aqui em estudo são denominados algoritmos genéticos, dada a semelhança de suas soluções com as seqüências genéticas de informação.

Assim, gera-se, estocasticamente, uma primeira população de indivíduos onde cada um representa uma possível solução. Deve-se implementar, a seguir, uma função geralmente denominada *fitness*. Essa função atribui um valor a cada solução que classifica a capacidade de solução do problema para cada indivíduo. De uma forma geral, quanto melhor a solução codificada por um indivíduo para o problema, maior o valor que a função *fitness* atribui a esse individuo. Deve-se, então, passar à seleção dos melhores indivíduos para que estes gerem novos indivíduos que passarão à próxima geração. Isso é feito da seguinte maneira: cada indivíduo tem uma probabilidade proporcional ao seu valor de *fitness* para ser selecionado. Então, seleciona-se

aleatoriamente indivíduos (onde os melhores serão mais provavelmente selecionados) que, associados dois a dois, gerarão novos indivíduos que passarão à geração seguinte.

Quando dois indivíduos são emparelhados para que gerem novos indivíduos, mais uma vez as técnicas emprestadas da natureza se fazem necessárias. Para que novos indivíduos (prole), possivelmente melhores que os "pais" que os geraram, sejam criados, eles precisam que (1) carreguem material genético de seus pais e (2) apresentem inovações que possam desenvolver, posteriormente, melhores características nas próximas gerações. Assim, usa-se de duas técnicas naturais, que são o crossover e a mutação. Ou seja, a partir do crossover os indivíduos "pais" trocam informação e a prole sofre mutação para que novas mudanças, possivelmente para melhor, sejam criadas.

A partir de uma população é criada então uma nova população que reunirá informação genética dos melhores indivíduos da geração passada e ainda algumas mudanças aleatórias, originadas por mutação, que poderá trazer melhorias ás gerações futuras. Esse processo é repetido quantas vezes se fizerem necessário até que os parâmetros exigidos sejam alcançados por uma população.

Dessa forma Holland introduziu na computação evolucionaria uma técnica com uma formulação teórica consistente, o que serviu a partir de então como base para a maioria dos desenvolvimentos na área a partir de então.

4.3 Paralelo com funções biológicas: reprodução, crossover e mutação

Quando geramos uma população estocasticamente, conseguimos criar tanto indivíduos que serão soluções ruins quanto indivíduos que representarão boas soluções. Porém, dificilmente conseguiremos um individuo que represente uma solução ótima logo na primeira população, dado que os indivíduos são representados por longas seqüências de dados. Assim, o algoritmo genético deve ter um método de otimização das soluções, ou seja, deve utilizar as soluções passadas para criar novas soluções de forma a buscar, gradualmente, uma solução ótima para o problema.

Os métodos propostos por John Holland são os métodos emprestados da natureza, a reprodução, o crossover e a mutação.

A reprodução é implementada na escolha dos indivíduos que formarão a nova geração. Existem algumas diferentes técnicas na seleção dos indivíduos dentro dos algoritmos genéticos, porem a mais comumente usada é o operador proporcional. Funciona da seguinte maneira: cada indivíduo passa pela função de avaliação, a função *fitness* , e a cada individuo é atribuído um valor, proporcional à sua eficiência como solução. Os indivíduos que serão selecionados para participar da reprodução são selecionados aleatoriamente, porém, cada indivíduo tem uma probabilidade de ser selecionado proporcional ao valor que a função *fitness* lhe atribuiu. Funciona como uma "roda da fortuna", onde cada indivíduo tem uma parcela da roda proporcional ao valor atribuído pela função *fitness*. O operador então "gira" a roda n vezes, onde n é o número de indivíduos, e os n indivíduos selecionados participarão então da reprodução, que formará a geração seguinte.

Dentro do processo de reprodução ocorrem os outros dois operadores principais utilizados nos algoritmos genéticos: a mutação e o crossover.

Para a reprodução, ao imitar-se a natureza, poder-se-ia usar um dos dois tipos de reprodução existentes na natureza, a reprodução *assexuada* e a reprodução *sexuada*.

Biologicamente, a reprodução sexuada representa um avanço com relação à reprodução assexuada. Isso porque, ao misturar informação genética de dois indivíduos, existe a possibilidade de se criar um individuo "filho" com características genéticas melhores do que a dois pais. Grosso modo, pode-se conseguir um filho com as boas características de um dos pais somada com as boas características do outro pai. Obviamente também há a possibilidade de se gerar um "filho" pior do que os pais, porem este seria eliminado nas gerações seguintes. No caso da reprodução assexuada,o processo restringir-se-ia a simplesmente passar às gerações seguintes os indivíduos selecionados na geração anterior. O processo de otimização então ficaria lento, pois dependeria unicamente das mutações aleatórias para haver progresso.

Assim, nesse processo de reprodução, é importante que os indivíduos "filhos" tragam consigo a informação genética dos pais, pois seus pais foram selecionados, como explicado anteriormente, como os mais aptos da geração passada. E é necessário que haja troca de informação genética entre os indivíduos "pais". Esse processo de troca de

informações é conhecido como crossover, e a ocorrência dele também é aleatória, sendo a probabilidade de ocorrência um parâmetro a ser definido, mas que gira em torno de 70% .

Dentro dos algoritmos genéticos existem alguns tipos diferentes de crossover, porém existem dois que são os mais comumente utilizados.

O primeiro tipo de crossover, chamado *crossover a ponto,* funciona da seguinte maneira: emparelham-se dois indivíduos selecionados de uma geração.e escolhe-se aleatoriamente um gene. A partir desse gene, trocam-se as informações genéticas dos dois indivíduos, gerando dois novos indivíduos, de acordo com o esquema da figura seguinte.

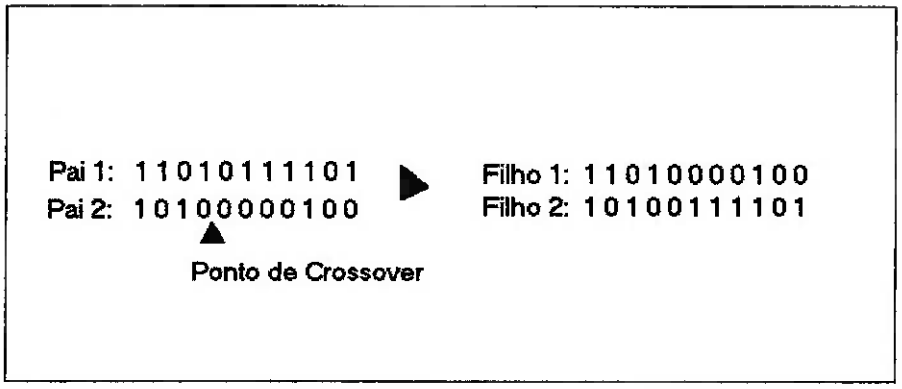


Figura 7- Primeiro tipo de Crossover

O segundo tipo de crossover é chamado *crossover por partes*. Diferente do crossover a ponto, o crossover por partes não troca toda a cadeia de genes a partir de um ponto, mas apenas um pedaço de material genético a partir do ponto de crossover. ou seja, apenas um determinado número de genes (número este aleatório) a partir do gene selecionado, de acordo com a figura seguinte.

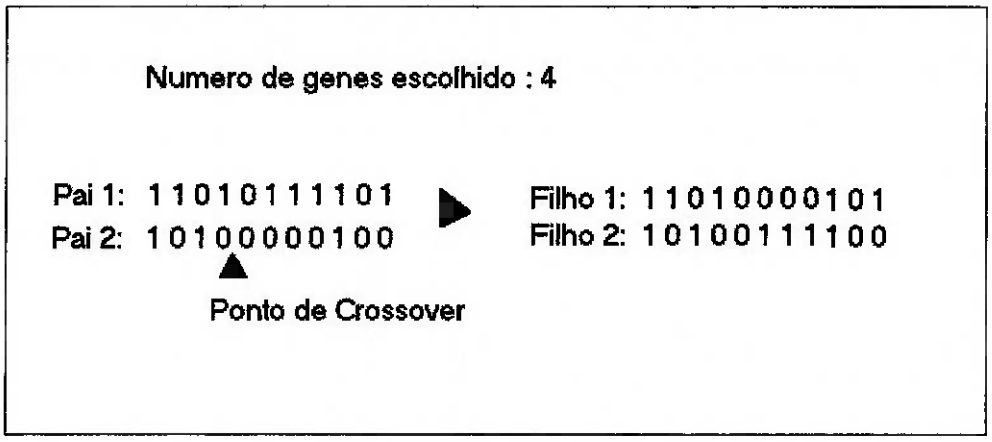


Figura 8- Segundo tipo de Crossover

Apesar da eficiência desse método de reprodução, ainda se faz necessário outro método para podermos obter uma solução ótima para nosso problema. Caso usássemos unicamente a reprodução como método de busca, encontraríamos uma limitação nos algoritmos. A solução ótima alcançada ficaria restrita às soluções dos indivíduos da população inicial, o que poderia representar um ótimo local da função a ser otimizada. Isso quer dizer que, caso os indivíduos da população inicial não contivessem intrinsecamente a solução ótima global para o problema, essa solução nunca seria alcançada.

Para contornar esse problema a *mutação* foi introduzida no método de John Holland. Na natureza, a mutação consiste na transformação de uma ou mais bases da cadeia genética, geralmente causada por radiação externa. No caso dos algoritmos genéticos, a mutação consiste somente em se transformar um bit "0" em "1" e vice versa, na cadeia de informação dos indivíduos "filhos". A mutação tem como função básica manter a diversidade das populações, ou seja, não permite que as populações fiquem restritas a algumas soluções sem nunca cobrir as outras soluções possíveis.

Assim, após a "reprodução" e o crossover, os indivíduos filhos passam por um processo de mutação. Também a probabilidade de ocorrência de mutação, para cada bit, é um parâmetro a ser definido de acordo com o problema. Esse parâmetro gira em torno dos 5%. O esquema da figura seguinte ilustra uma ocorrência de mutação:

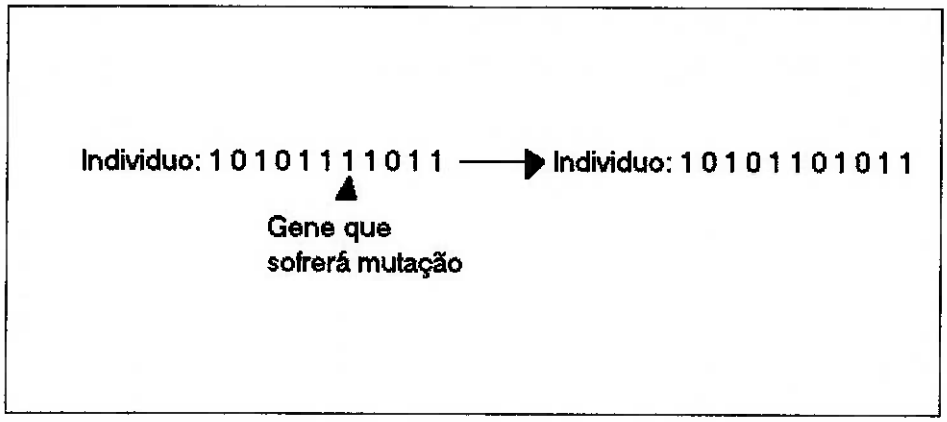


Figura 9- Processo de mutação

A mutação é uma função a que se deve tomar bastante cuidado, principalmente quanto a definição de seu parâmetro de probabilidade, dada a capacidade destrutiva dessa função.

4.4 Influência do tamanho da população nos algoritmos

Quando se inicia a implementação de solução por algoritmos evolutivos, o primeiro parâmetro a ser considerado é o tamanho da população. Goldberg (1985) propôs que o tamanho de uma população deva depender unicamente do tamanho dos strings dos indivíduos, com a forma

$N = 1{,}65 \cdot 2^{0{,}21\,L}$

Onde L é o tamanho do string, em bits.

Schaffer et al (1989) sugeriu que uma população de 20 a 30 indivíduos com probabilidade de mutação entre 0.005 e 0.01 e probabilidade de crossover entre 0.75 e 0.95 funcionaria bem para qualquer aplicação com algoritmos evolutivos.

Goldberg, Déb e Clark (1992) sugeriram que o tamanho da população dependeria também da não – linearidade do problema,

$$N \approx O\left(2^K \frac{\sigma^2}{d^2}\right)$$

Na qual k é a ordem de não–linearidade do problema, σ^2 é a variância do problema e d é a diferença dos valores de *fitness* entre os ótimos locais e globais. Conclui-se da equação que, quanto maior a não – linearidade do problema, maior a população.

De qualquer forma, o que se pode observar dos métodos de calculo de tamanho de população é que o tamanho da população deve ter a mesma ordem do que o comprimento dos indivíduos (N ~ O(L)). Assim, se tivermos indivíduos com 30 bits, deveremos ter populações com 30 a 50 indivíduos, assim como se tivermos indivíduos com 200 bits deveremos ter populações com 100 ou 200 indivíduos.

Pode-se imaginar que, quanto maior a população, melhor será a performance do algoritmo. Para resolver esta questão, é proposto o seguinte exemplo:

Deve-se minimizar a função

$f_1(x_1,x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 + 0.1 * [(x_1 - 3)^2 + (x_2 - 2)^2]$; $-6 \leq x_1, x_2 \leq 6$

Cujo gráfico "contour" é o seguinte,

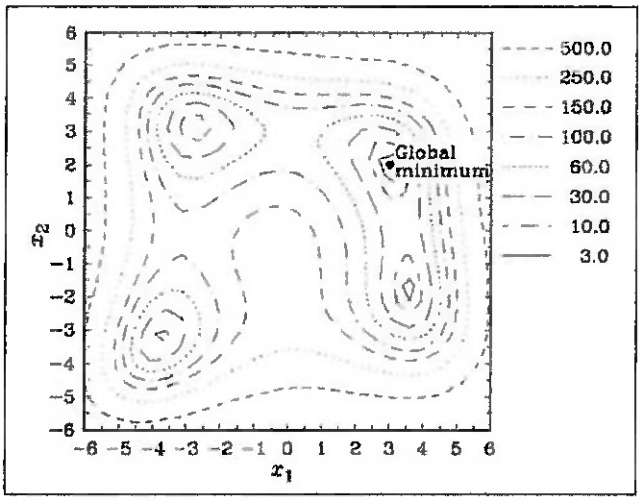


Figura 10- Gráfico "Contour" da função f1

A implementação da solução dar-se-á com indivíduos de tamanho 12 bits (que codificarão valores de x_1 e x_2 entre –6 e 6). O algoritmo será testado para vários tamanhos de população. Para cada população, haverão 50 rodadas. Cada rodada terminará quando atingir um dos dois casos abaixo:

1- Chegar a uma solução próxima (+- 0,02) do mínimo, ou

2- Atingir o número máximo de gerações, igual a 9000.

Usando-se probabilidade de mutação igual a 0,05 e de crossover igual a 0,75, foi implementado o método. Os resultados foram os seguintes:

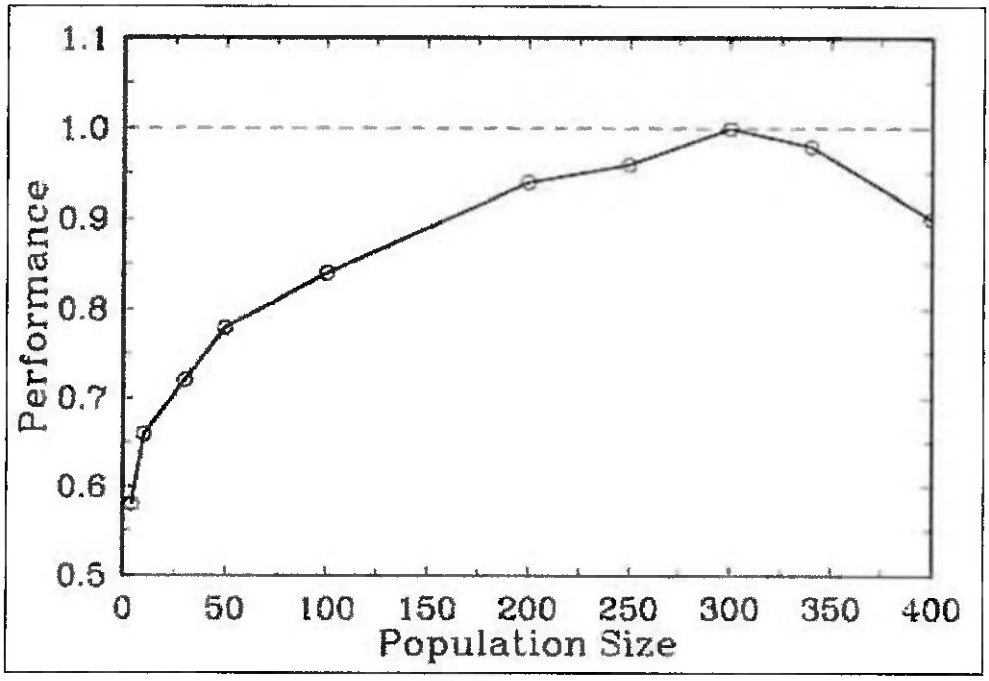


Figura 11- Performance versus tamanho de população

O índice "performance" é a razão entre o número de rodadas que atingiram o primeiro critério (rodadas com sucesso) dividido pelo número de rodadas (50) para cada população.

Percebe-se, no gráfico, três regiões distintas. Para populações muito pequenas, a performance é muito baixa. Conforme o tamanho da população aumenta, o sucesso vai aumentando até atingir um ponto crítico, onde quase todas as rodadas obtêm sucesso. A partir desse ponto, a performance cai. Isso se deve ao pequeno número de populações

possível, visto que se aumentarmos o número de gerações para populações grandes a sua performance aumentaria.

Portanto, podemos observar que, a partir do ponto crítico, um aumento do número de indivíduos de uma população não trará uma performance melhor ao algoritmo. Somente trará maior tempo de cálculo, pois para manter uma boa performance precisaríamos aumentar o número de gerações e, portanto, o tempo de cálculo [10].

4.5 Influência dos parâmetros de mutação e crossover nos algoritmos

Tem como uso comum os valores de 5% de probabilidade de ocorrência de mutação e de 70% de probabilidade de ocorrência de crossover. Esses números foram obtidos empiricamente, sem que as teorias sobre o assunto fossem realmente conclusivas a respeito desses valores.

Porém, por analise simples podemos ao menos ter uma idéia sobre a grandeza desses números.

Quanto à mutação, podemos imaginar que muito provavelmente a sua ocorrência é destrutiva, dado que é mais possível que uma mutação vá prejudicar o individuo do que aprimora-lo. Assim, e sua ocorrência for alta a mutação seria nociva, acabando por prejudicar as populações e fazendo com que o método divirja.

Já a ocorrência de crossover deve ser alta, visto que seu poder destrutivo é pequeno. Porém, não deve atingir a totalidade, pois deve permitir que alguns indivíduos possam passar a geração seguinte.

Um exemplo para a ilustração dos efeitos dos parâmetros de mutação e de crossover é dado a seguir. No caso de minimizar a seguinte função:

$$f_1(x_1,x_2) = (x_1^2 + x_2 - 11) + (x_1 + x_2^2 - 7)^2 \; ; \; 0 \leq x_1, x_2 \leq 6$$

Usando-se o mesmo tipo de simulação usado no exemplo anterior, com 50 rodadas para cada combinação de valores de p_c (probabilidade de ocorrer crossover) e p_m (probabilidade de ocorrer mutação), obteve-se os seguintes resultados:

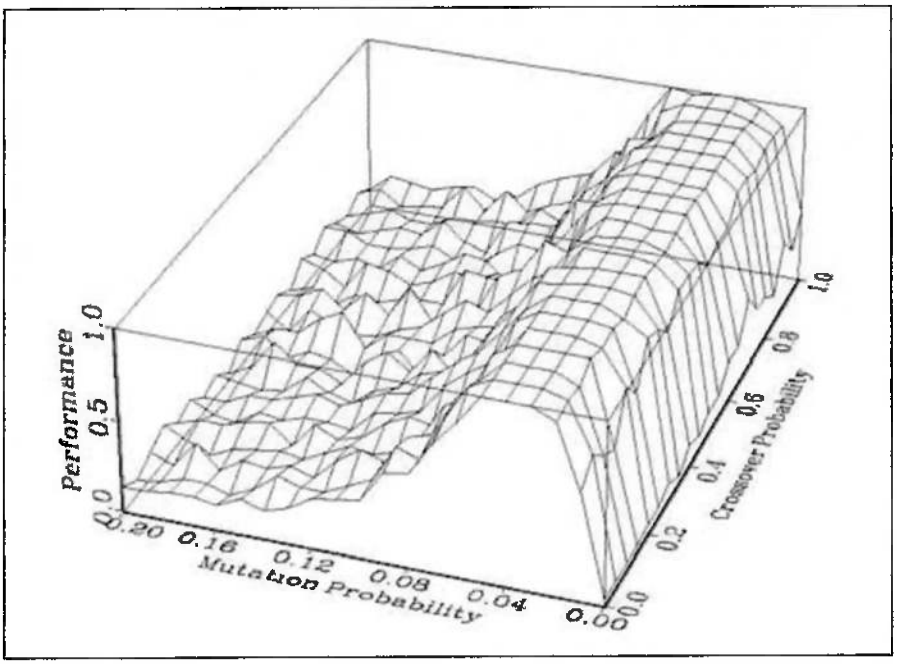


Figura 12- Performance dos algoritmos em função dos parâmetros de mutação e crossover

Verifica-se, acima, que os algoritmos não são muito sensíveis quanto à variação do parâmetro de crossover, mas pode-se observar que altas taxas de mutação são prejudiciais ao algoritmo[10].

Atualmente alguns autores defendem que, em algoritmos de maior complexidade e maior necessidade de gerações, os parâmetros de crossover e de mutação deve ser intrínseco aos indivíduos, ou seja, cada indivíduo tem, na sua genética, codificada a sua probabilidade de sofrer crossover e mutação [11],[12]. Porém não cabe ao escopo desse trabalho a implementação desse método, dada a maior complexidade dos indivíduos face à pequena melhora que talvez obtivéssemos no método.

4.6 Convergência dos Algoritmos Evolutivos

A eficácia dos algoritmos genéticos pode ser verificada numa grande variedade de maneiras. Um critério óbvio seria a convergência. Holland sugeriu que a convergência não seria uma boa medida de performance ao observar que existem procedimentos de busca para funções de larga escala que convergem mas que despendem um tempo muito grande de processamento, e que tais processos não seriam úteis na solução de problemas complexos. Holland sugeriu também que um algoritmo de otimização que evita a convergência para ótimos locais leva vantagem sobre outros algoritmos que, mesmo possuindo boa convergência, fique preso em ótimos locais. Assim, a convergência não seria um bom parâmetro para comparar algoritmos diferentes. Porém, Holland propôs que dentro dos algoritmos poderiam ser desenvolvidas técnicas que evitassem os ótimos locais e que, nesse caso, a convergência seria uma boa medida da eficácia do método, podendo comparar os algoritmos genéticos entre si e com outras técnicas não evolutivas.

5 IMPLEMENTAÇÕES DE ALGORITMOS EVOLUTIVOS

5.1 Estratégias Evolutivas

A seguir, é mostrada uma aplicação das Estratégias Evolutivas para a otimização dos valores dos ganhos de um controlador PI utilizado para o controle de um motor de corrente contínua. O modelo do motor foi estabelecido a partir das experiências realizadas na disciplina PMC-526. Assim a função de transferência para o motor é:

$$\frac{Y(s)}{X(s)} = \frac{Kw}{T \cdot s + 1} \qquad (15)$$

com Kw = 0.977 e T = 0.52s.

Para esse problema foi implementada uma estratégia do tipo (2,10)-ES, com mutação linear.

5.1.1 Programa Principal

```
global K Ti tout y aux Td adaant
% Programa para otimização dos ganhos de um controlador PI para controle de
% um motor CC - Referência à experiência realizada na disciplina PMC-526
% Nesse problema as variáveis a serem otimizadas são K e Ti
% Os critérios de otimização são máximo sobressinal e tempo de acomodação
% Será determinado um algoritmo do tipo (mi,lambda)-ES, com mutação linear

% Parâmetros do modelo do motor CC
Kw = 0.977;
T = 0.52;

% Parâmetros para o algoritmo
mi = 2;
lambda = 10;    % lambda > mi
n = 2;
delta = 2;
moptions = simset('DstWorkspace','base');
adaant = inf;

% Processo de otimização
Pt = Po(mi,n);
Ptlinha = m(r(Pt,lambda),delta);
tempo = 1;
fim = t(Ptlinha,tempo);
while  fim == 0
   clear Pt;
   Pt = s(Ptlinha,mi);
   Ptlinha = m(r(Pt,lambda),delta);
   tempo = tempo + 1;
   fim = t(Ptlinha,tempo);
   end
end
```

O script mostrado representa o ciclo padrão do algoritmo evolutivo.

5.1.2 Operação para criar população

```
% Função para geração da população inicial
function fu = Po(mi,n)
for i = 1: mi
   for j = 1:n
      Pop(i,j,1) = rand(1);   % Valor da variável objetiva
      Pop(i,j,2) = rand(1);   % Valor do parâmetro estratégico sigma
   end
end
fu = Pop;
```

Essa implementação de geração de população inicial é randômica, pode-se determinar uma população inicial baseada em valores já conchecidos.

5.1.3 Operação de mutação

```
% Operador de mutação
function fu = m(populacao,delta)
[ linha coluna prof ] = size(populacao);
for i = 1:linha
   for j = 1:coluna
      Pop(i,j,2) = populacao(i,j,2)*exp(delta*randn(1));
      Pop(i,j,1) = populacao(i,j,1) + Pop(i,j,2)*randn(1);
   end
end
fu = Pop;
```

Implementação de acordo com as equações (12).

5.1.3 Operação de recombinação

```
% Operador de recombinação, usando a sugestão de Schwefel
function fu = r(populacao,lambda)
[ linha coluna prof ] = size(populacao);
for i = 1: lambda
   for j = 1: coluna
      pai1 = pai(linha);
      pai2 = pai(linha);
      a = rand(1);
      if a <= 0.5
         escolhido = pai1;
      else
         escolhido = pai2;
      end
      Pop(i,j,1) = populacao(escolhido,j,1);
      Pop(i,j,2) = 0.5*(populacao(pai1,j,2) + populacao(pai2,j,2));
   end
end
fu = Pop;
```

Recombinação discreta para variáveis objetivas e intermediária para os parâmetros estratégicos.

5.1.4 Critério de parada

```
% Critério de parada
function fu = t(populacao,tempo)
global K Ti tout y moptions aux Td adaant
para = 0;
[ linha coluna prof ] = size(populacao);
for i = 1: linha
   adaptacao(i) = f(populacao,i);
end
menor = acha_menor(adaptacao);
if adaptacao(menor) < 0.1
   para = 1;
end
if tempo == 300
   para = 1;
end
if adaptacao(menor) < 1000
   if abs(adaptacao(menor)-adaant) < 0.0000000001
      para = 1;
   end
end
adaant = adaptacao(menor);
fu = para;
```

Nessa implementação são usados critérios para parada de acordo com número de iterações, pelo progresso absoluto e pelo progresso relativo.

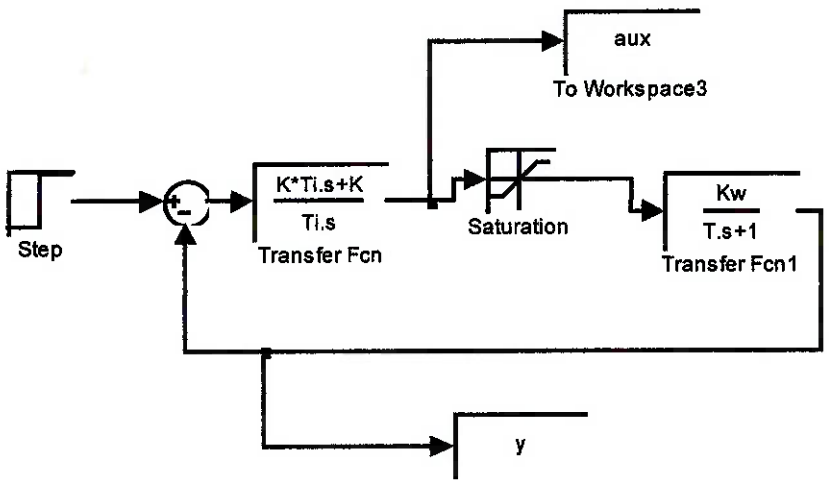
5.1.5 Função Objetiva

```
% Função objetiva
function fu = fpi(populacao,individuo)
global K Ti tout y moptions aux Td
[ linha coluna prof ] = size(populacao);
restricao = 0;
for j = 1: coluna
   if populacao(individuo,j,1) <= 0
      restricao = 1;
   elseif populacao(individuo,j,1) > 100
      restricao = 1;
   end
end
if restricao == 1
   fu = inf;
else
   K = populacao(individuo,1,1);
   Ti = populacao(individuo,2,1);
   sim('controlador_pi', 10, moptions); % Devolve as variáveis tout e y
   tam = size(aux);
   tam1 = tam(1,1);
   for i = 1: tam1
      if abs(aux(i)) >= 5
         restricao = 1;
      end
```

```
    end
    Mp = sobressinal(tout,y);
    if Mp < 0
        Mp = 10;
    end
    ts = tempo_s(tout,y);
    dm = abs(Mp-0.02);
    dt = abs(ts - 0.3);
    if restricao == 1
        fu = inf;
    else
        fu = dm + dt;
    end
end
fu;
```

Para a avaliação das soluções são utilizadas simulações feitas no SIMULINK. Do resultado da simulação do sistema para entrada em degrau são calculados os valores de sobressinal e tempo de acomodação. O valor da função objetiva para o individuo é estabelecido como a soma das diferenças entre os valores obtidos e os valores desejados para os mesmos.

5.1.6 Modelo do sistema utilizado nas simulações



5.1.7 Resultados

Foram realizadas simulações para os seguintes valores de sobressinal e tempo de acomodação:

- Mp = 0.02 e ts = 0.3s
- Mp = 0.1 e ts = 0.5s
- Mp = 0.2 e ts = 3s

Os resultados obtidos nesses três casos podem ser vistos nas figuras 13, 14 e 15 respectivamente. Os valores obtidos para o sobressinal e tempo de acomodação foram:

Mp = 0,0206 e ts = 0,325s. Ganhos: K = 3,8 e Ti = 0,4.

Mp = 0,1054 e ts = 0,575s. Ganhos: K = 4,92 e Ti = 0,19.

Mp = 0,227 e ts = 2,775s. Ganhos: K = 0,14 e Ti = 0,04.

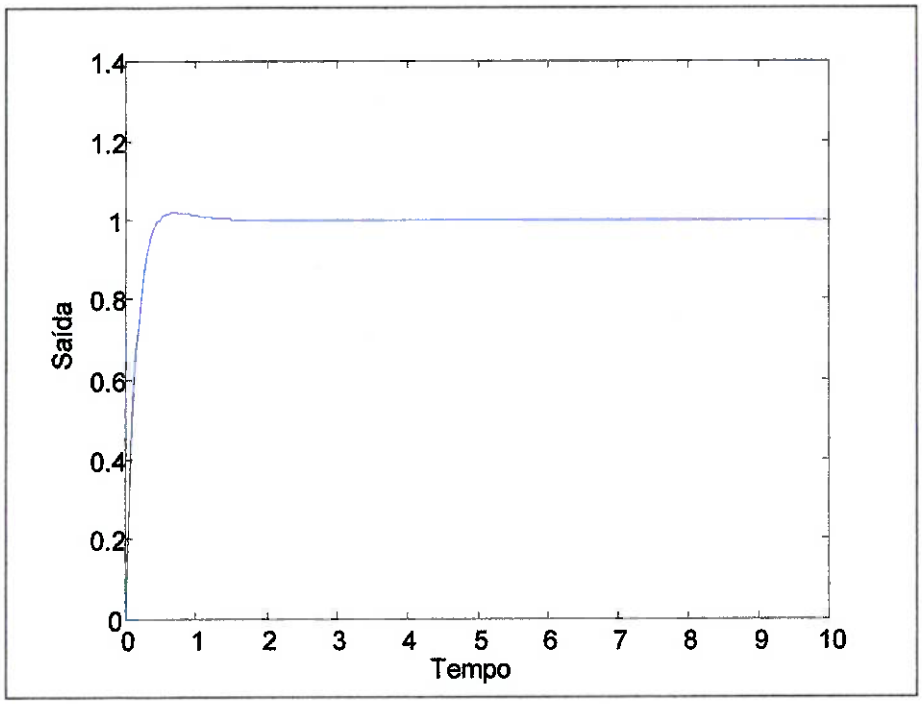


Figura 13- Resposta do sistema para entrada a degrau

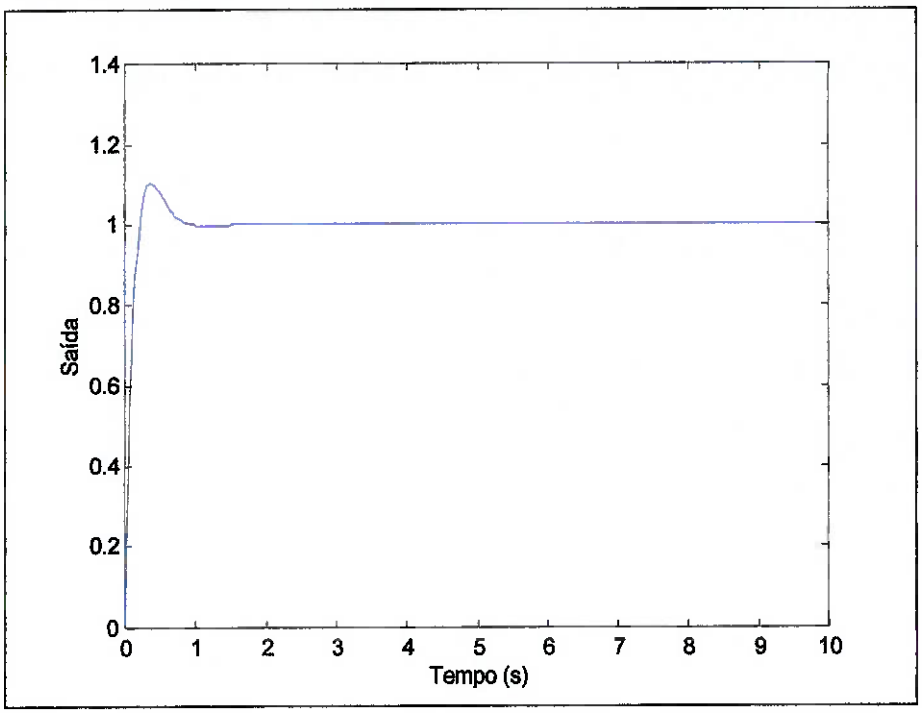


Figura 14- Resposta do sistema para entrada a degrau

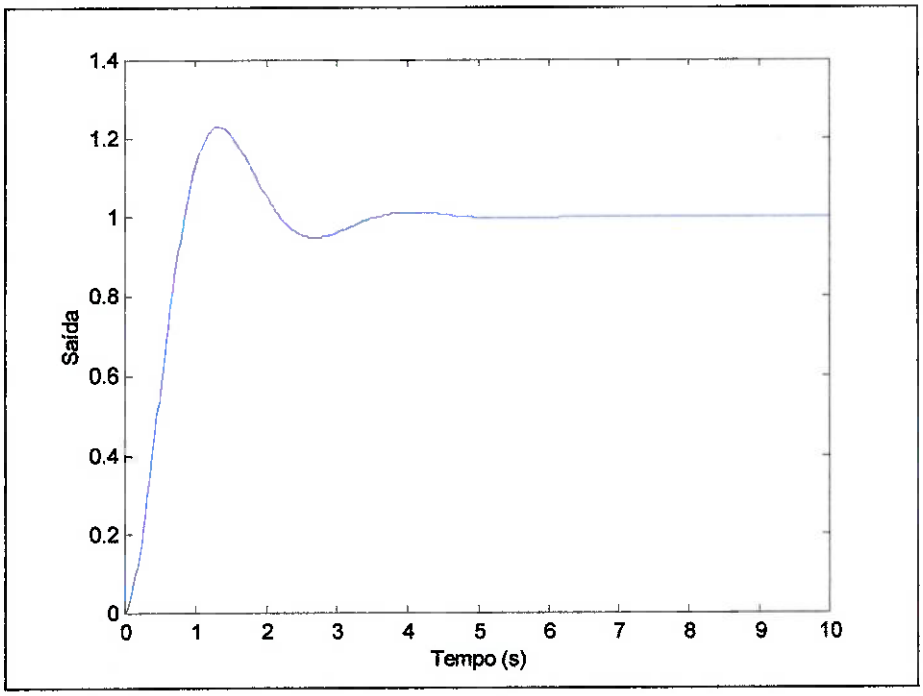


Figura 15- Resposta do sistema para entrada a degrau

Como pode ser observado nas figuras 13, 14 e 15, os resultados obtidos são extremamente satisfatórios, tendo-se alcançado valores bastante próximos dos desejados, com esforço computacional bastante baixo.

5.2 Algoritmos Genéticos

A seguir, uma implementação simples dos algoritmos genéticos desenvolvida no programa MatLab.

5.2.1 Programa Principal

```
individuos=100;
genes=12;
A=criar_populacao(individuos, genes);
pc=0.7; % Crossover rate
pm=0.001; % Mutation rate
rodadas = 0;
ind=0;
fit = fitness(A);
while ind==0            % o algoritmo rodara ate a f parada mandar (linha 22)
   sel = roda_fortuna(A,fit);
   for a=1:individuos/2                  % Ocorrerao individuos/2 cruzamentos
        r(1)=sel(2*a-1);                 % Selecao do 1o. individuo
        r(2)=sel(2*a);                   % Selecao do 2o. individuo
        c=crossover(A(r(1),:),A(r(2),:),pc);  % Crossover entre os individuos
        c(1,:)=mutacao(c(1,:),pm);            % Mutacao no primeiro individuo
        c(2,:)=mutacao(c(2,:),pm);            % Mutacao no segundo individuo
        B(a,:)=c(1,:);                        % Armazenamento das proles na matriz B
        B(a+individuos/2,:)=c(2,:);
    end
    A=B;
    ind=parada(A);
    fit = fitness(A);
    max = maximo(fit);
    valor=fit(max);
    end
    rodadas=rodadas+1;
    imp(rodadas)=valor;
end
for i=1:rodadas
    x(i)=i;
end
```

O problema consiste em se encontrar o Maximo da função

$$f(x) = \sum_{i=0}^{genes} x_i \cdot 2^i$$

ou seja, encontrar o indivíduo que tenha todos os seus genes igual a 1. As funções de criação de população, fitness, reprodução, crossover, mutação e parada são implementadas a seguir.

5.2.2 Função criar população

```
function criar_populacao=criar_populacao(individuos, genes)
for a=1:individuos
    for b=1:genes
        y=rand;
        if y <= 0.5
            y=0;
        else
            y=1;
        end
        pop(a,b)=y;
    end
end
criar_populacao=pop;
```

A população é gerada estocasticamente, gerando uma população com boa variedade de indivíduos.

5.2.3 Função Fitness

```
function fitness=fitness(matriz)
[linhas colunas] = size(matriz);
total(linhas)=0;
for i = 1: linhas
   for a=1:colunas
      total(i)=total(i)+(2^(a-1))*(matriz(i,a));            % Para a fitness ser o valor
decimal do binario
      %total=total+1*vetor(a);                    % Para a fitness ser a soma dos 1's
   end
end
fitness=total;
```

A função fitness implementa a função de otimização e retorna o valor encontrado para cada individuo.

5.2.4 Função de reprodução ("Roda da Fortuna")

```
function roda_fortuna=roda_fortuna(A,fit)
[ind gen]=size(A);
total=0;
for i=1:ind
    total=total+fit(i); % Soma-se todos os fitnesses para que cada individuo tenha uma
end                     % probabilidade igual a divisao entre seu fitness e o total dos
                        % individuos
for i=1:ind
    if i>1                          % Se nao for o 1o. individuo, entao o intervalo
        prob(i)=prob(i-1)+ fit(i)/total;    % correspondente ao individuo e a soma das
probabi-
    else                           % lidades dos individuos anteriores mais a do proprio
        prob(i)=fit(i)/total;      % Se for o 1o, individuo, seu intervalo e de 0 a sua
    end                                                 % probabilidade
end
clear Sel;
Sel = 0;
for i = 1:ind
   aux = 1;
   a=rand(1);
   while a > prob(aux)
      aux = aux + 1;
   end
   Sel(i) = aux;
end
roda_fortuna=Sel;
```

Percebe-se, no código-fonte, como é utilizada a seleção proporcional dos genes, utilizando os valores atribuídos pela função fitness, que é alocado pela função principal no vetor fit.

5.2.5 Função crossover

```
function crossover=crossover(ind1,ind2,prob)
[n m]=size(ind1);
a=rand;
if a<=prob
    loc=ceil(rand*m);
    sent=rand
    if sent > 0.5        % a probabilidade do ultimo termo
        for i=loc:m
            g=ind1(i);
            ind1(i)=ind2(i);
            ind2(i)=g;
        end
    else
        for i=loc:-1:1
            g=ind1(i);
            ind1(i)=ind2(i);
            ind2(i)=g;
        end
    end
end
crossover=[ind1;ind2];
```

Essa função crossover implementa um terceiro método, variante do crossover a ponto. Aqui, escolhe-se o gene onde se iniciara o crossover e depois se seleciona o lado onde se dará o crossover: se do gene pra frente ou para trás.

5.2.6 Função Mutação

```
function mutacao=mutacao(M,pm)
[a b]=size(M);
r=M;
for i=1:b
    z=rand;
    if z<=pm
        if r(1,i)==0
            r(1,i)=1;
        else
            r(1,i)=0;
        end
    end
end
mutacao=r;
```

A função mutação é simples, procurando fazer bit a bit a alteração de mutação.

5.2.7 Função de parada

```
function parada=parada(A)
[ind genes]=size(A);
resp=0;
for i=1:ind
    total=0;
    for j=1:genes
        total=total+A(i,j);   % O algoritmo para se todos os genes forem iguais 1
    end
    if total==genes
        resp=i;
    end
end
parada=resp;
```

Dado que o algoritmo é relativamente simples, o algoritmo de parada sinalizará a interrupção do processo unicamente quando for atingido o objetivo, sem limitação quanto ao numero de gerações criadas.

5.2.8 Resultados

Os resultados obtidos foram:

```
Rodadas =   261
ans =     1     1     1     1     1     1     1     1     1     1     1     1
```

E o gráfico obtido foi:

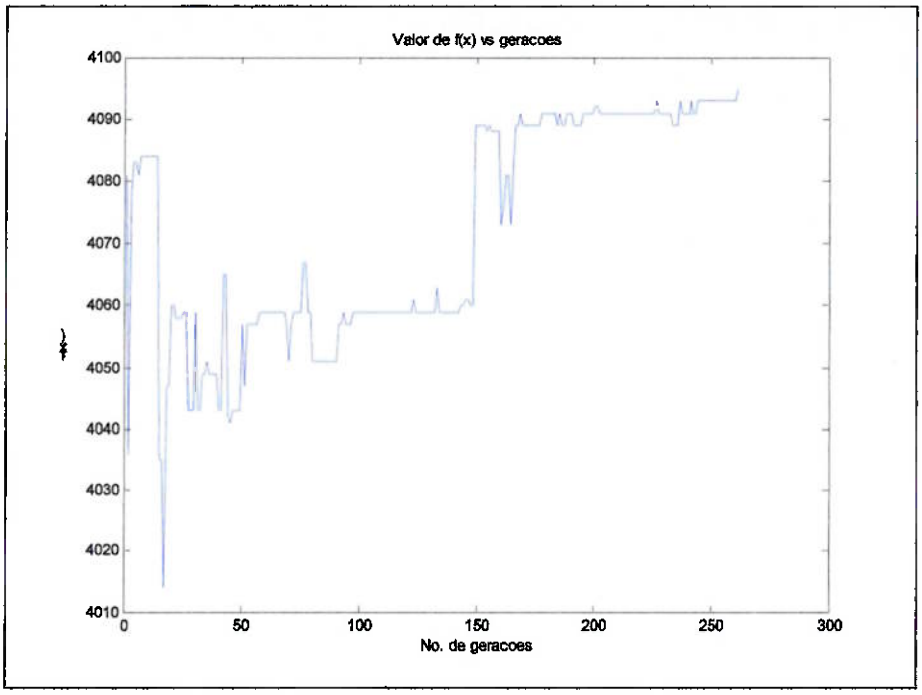


Figura 16- f(x) versus gerações, Algoritmos Genéticos

6 SELEÇÃO DO MÉTODO A SER UTILIZADO

Para a seleção do método a ser utilizado na solução do problema de vibração do braço robótico, montamos a seguinte matriz de decisão:

Matriz de Decisão					
		Algoritmos Genéticos		Estratégias Evolutivas	
Parâmetro	Peso	Nota	Valor	Nota	Valor
Esforço Computacional	3	6	18	7	21
Resolução das soluções	3	5	15	9	27
Ajuste de parâmetros	2	7	14	8	16
Interface de dados	2	5	10	10	20
Referências	1	10	10	8	8
Total			67		92

Tabela 1- Matriz de Decisão

Os parâmetros de decisão adotados são:

Esforço computacional: métodos que utilizam algoritmos evolutivos são métodos que exigem grande capacidade computacional, dado que são métodos em que são feitos muitos cálculos. Assim, consideramos que o esforço computacional é um parâmetro decisivo na escolha do método, dado que o tempo de calculo dos computadores pode ser grande,

Resolução das soluções: para a resolução do problema os resultados obtidos devem ter uma precisão razoável. Porém, para obtermos boa resolução em algoritmos genéticos devemos utilizar um número de bits muito grande, o que prejudicaria o esforço computacional, e assim atribuímos valor baixo aos algoritmos genéticos nesses critérios.

Ajuste de parâmetros: os parâmetros utilizados nos métodos são importantes na velocidade de solução. No caso dos algoritmos genéticos, os parâmetros são pré-selecionados. Se por um lado computacionalmente isso é bom, dado que não haverá

esforço computacional na determinação desses parâmetros, por outro lado não é garantido que esses parâmetros são ótimos para a solução. Já as estratégias evolutivas as soluções já contêm os parâmetros, o que significa que estes serão selecionados e que, apesar de prejudicar o esforço computacional, serão ótimos para a solução do problema,

Interface de dados: A inserção de dados e o resultado obtido são codificados em algoritmos genéticos (e, portanto, a interface não é boa, dado que para entendermos os dados tem que haver a decodificação dos mesmos), porém nas estratégias evolutivas os parâmetros são reais e, portanto, não há a necessidade de codificação/decodificação dos dados,

Referências: A bibliografia nos assuntos é boa, assim como textos onde se têm implementações do método. Porém a bibliografia nos algoritmos genéticos é ainda mais extensa.

Assim, pela matriz de decisão a nossa escolha para o método utilizado no problema de vibração em braço robótico é as estratégias evolutivas.

Outro critério que encontramos para a escolha das estratégias evolutivas foi o fato de que encontramos textos de otimização de problemas a parâmetros reais, do modo como utilizaremos, como em [5], [13], [14], [15] e [16].

7 MODELO DO BRAÇO ROBÓTICO FLEXÍVEL

O braço robótico utilizado no presente trabalho está esquematizado na figura 17:

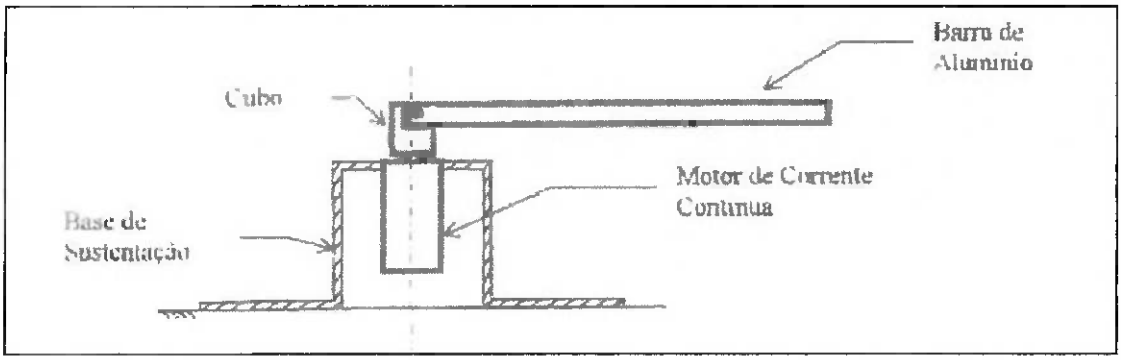


Figura 17- Esquema do braço robótico

Para a modelagem matemática do braço flexível, convém apresentar alguns dos métodos utilizados para o equacionamento da dinâmica do braço.

Utilizou-se para analise da vibração do braço o Princípio do Referencial Flutuante. Dado um corpo que sofre pequenas deformações elásticas, muitas vezes é difícil especificar eixos para a medida dessas deformações. Isso porque se é escolhido um sistema inercial, uma rotação eventualmente causada no corpo exigiria uma análise matematicamente complicada. Porém, se fixarmos o eixo ao próprio corpo poderemos medir as deflexões relativamente ao próprio eixo, o que simplifica a análise. Utiliza-se, portanto, para esse trabalho um tipo de referencial flutuante chamado *Tisserand Frame* [17].

O *Tisserand Frame* está representado na figura 18:

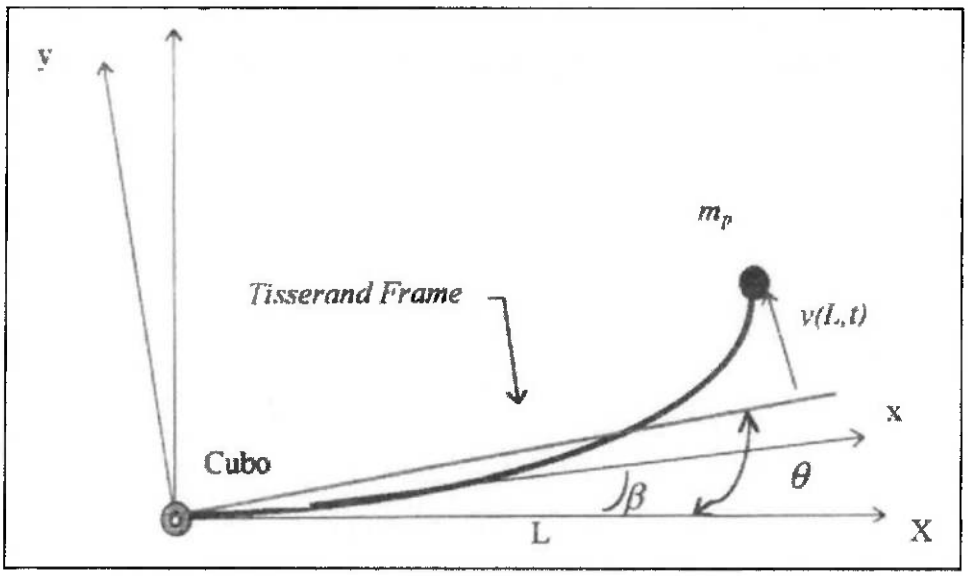


Figura 18- Referencial flutuante

Definido o referencial em relação ao qual serão medidas as vibrações na ponta do braço, através do Princípio dos Trabalhos Virtuais e do Princípio Estendido de Hamilton derivam-se as equações que regem a dinâmica do braço [17].

O modelo matemático do sistema fica:

$$\begin{Bmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{\eta}_1 \\ \ddot{\eta}_1 \\ \dot{\eta}_2 \\ \ddot{\eta}_2 \end{Bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -\omega_1^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -\omega_2^2 & 0 \end{bmatrix} \begin{Bmatrix} \theta \\ \dot{\theta} \\ \eta_1 \\ \dot{\eta}_1 \\ \eta_2 \\ \dot{\eta}_2 \end{Bmatrix} + \begin{bmatrix} 0 \\ 1/I_{total} \\ 0 \\ \phi_1'(0) \\ 0 \\ \phi_2'(0) \end{bmatrix} .T(t) \qquad (16)$$

Na qual,

θ - posição angular do braço;

η_1 – coordenada modal do primeiro modo de vibração do braço.

η_2 – coordenada modal do segundo modo de vibração do braço.

ϕ_1 – Função de forma do primeiro modo de vibração.

ϕ_2 – Função de forma do segundo modo de vibração.

I_{total} – Momento de inércia total do conjunto cubo+eixo.

ω_1 – freqüência natural do primeiro modo de vibração.

ω_2 – freqüência natural do segundo modo de vibração.

T(t) – Torque aplicado pelo motor.

A saída do sistema fica da seguinte maneira:

$$\begin{Bmatrix} \theta \\ y_{rel}(L) \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \phi_1(L) & 0 & \phi_2(L) & 0 \end{bmatrix} \begin{Bmatrix} \theta \\ \dot{\theta} \\ \eta_1 \\ \dot{\eta}_1 \\ \eta_2 \\ \dot{\eta}_2 \end{Bmatrix} \qquad (17)$$

Onde $y_{rel}(L)$ é o desvio em y no ponto L, ou seja, na ponta da barra flexível. É esta saída que será objeto de minimização do nosso trabalho.

8 APLICAÇÕES DOS ALGORITMOS EVOLUTIVOS PARA O CONTROLE DO BRAÇO FLEXÍVEL

Como objetivo do trabalho, far-se-á a otimização de um controlador por alocação de pólos para o sistema. O controlador será obtido para um deslocamento de 15° do braço, objetivando-se a menor vibração possível na ponta do mesmo, e também o menor tempo de acomodação possível.

Em [17] Gildin projetou um controlador por alocação de pólos, e inicialmente buscou-se apenas a otimização dos ganhos já obtidos em [17]. Em seguida, utilizou-se as estratégias evolutivas para encontrar os ganhos do controlador por alocação de pólos a partir de valores iniciais gerados estocasticamente. Assim, pode-se comparar os resultados obtidos para analisar a eficiência do algoritmo na busca pelo resultado ótimo.

Por fim, buscaremos projetar um controlador novo usando PID's. Os parâmetros dos controladores serão o objeto de otimização, e nesse caso pode-se verificar a eficiência do método para a sintonia de um sistema de controle para o qual não existe um método tradicional.

9 CONTROLADOR POR ALOCAÇÃO DE PÓLOS PARA O BRAÇO FLEXÍVEL

Os resultados obtidos em [17] para o controlador por alocação de pólos são novamente reproduzidos aqui para efeito de comparação com os resultados obtidos com a utilização de estratégias evolutivas.

Seguindo o procedimento normal para o projeto de um controlador por alocação de pólos foram definidos os pólos desejados do sistema, com a determinação dos pólos dominantes de tal forma que a resposta apresentasse sobressinal de 10% e tempo de acomodação de 1s. A matriz de ganhos obtida para o sistema foi:

$$K = [7.5534 \quad 2.6800 \quad -65.2853 \quad -0.1523 \quad -82.6199 \quad -0.2741] \qquad (18)$$

Como o objetivo é fazer com que o braço realize uma manobra de rotação de 0 a 15° e pare, tendo a vibração em sua ponta livre minimizada, utilizou-se então como entradas para o sistema:

- Entrada em degrau de 15° para o ângulo θ de posição do braço.
- Entrada de valor zero para o deslocamento do braço em relação ao referencial flutuante

As respostas obtidas para a posição angular e o deslocamento da ponta do braço podem ser vistas nas figuras 19 e 20 respectivamente. A figura 21 mostra a curva de torque que deve ser exercido sobre o braço.

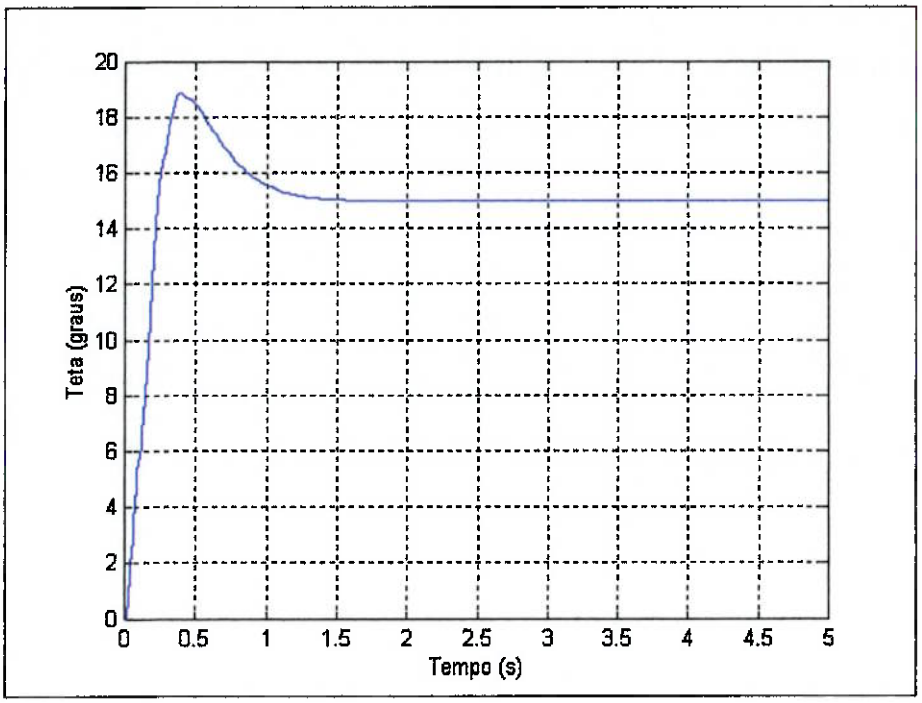


Figura 19- Angulo θ em função do tempo do sistema inicial

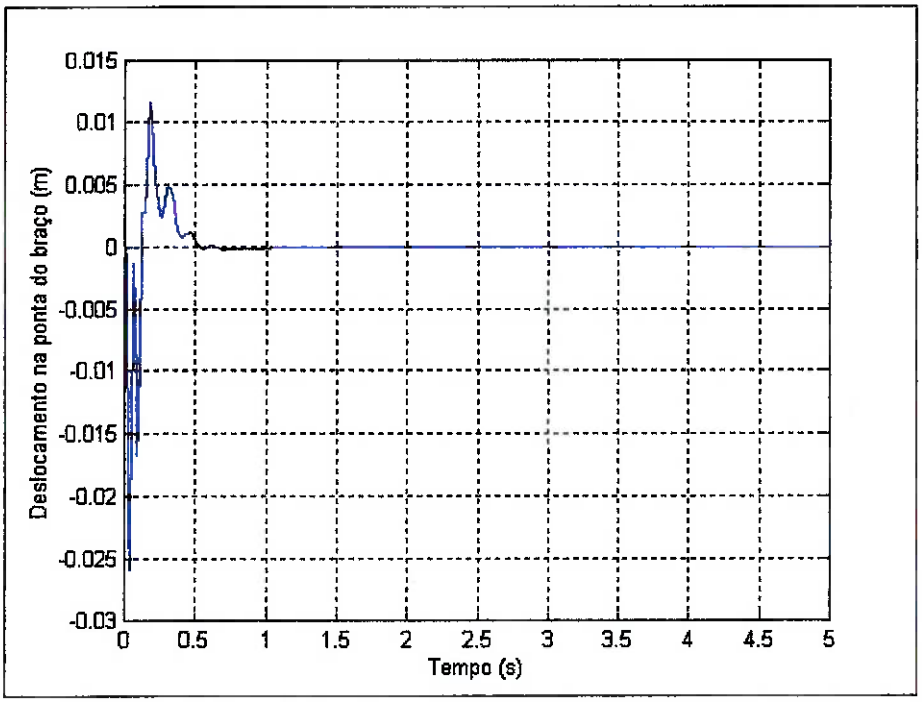


Figura 20- Deslocamento na ponta do braço em função do tempo para sistema inicial

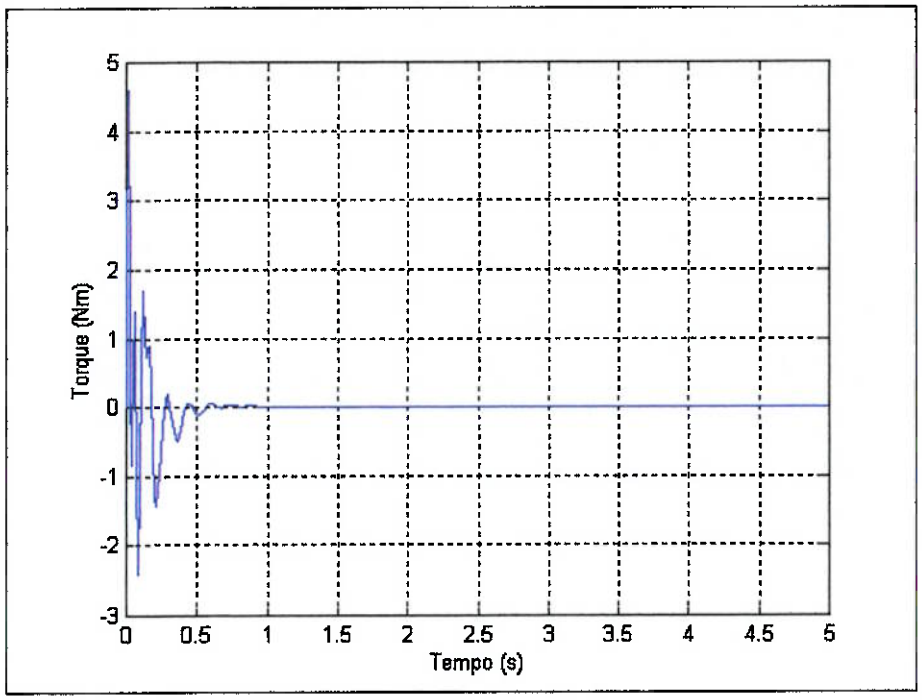


Figura 21- Torque em função do tempo do sistema inicial

Com um algoritmo baseado nas estratégias evolutivas fez-se a otimização da matriz de ganhos. Foram feitas duas experiências para esse objetivo; a primeira, e mais simples, foi a otimização dos ganhos a partir de um valor inicial igual aos obtidos em (18). A segunda experiência foi a tentativa de obtenção de valores para os ganhos a partir de chutes iniciais estocásticos. Os resultados são relatados a seguir.

9.1 Otimização dos ganhos a partir de um valor inicial dado

Para a implementação das estratégias evolutivas na solução desse problema foram utilizados os seguintes critérios:

- Estratégia Evolutiva do tipo (μ,λ)-ES.
- População inicial fixa, com valores iguais aos de (18).
- Mutação linear, como em (12).

- Recombinação discreta para as variáveis objetivas e recombinação intermediária para os parâmetros estratégicos.
- Critérios de parada baseados em número de iterações e progresso relativo das soluções.

Os principais parâmetros a serem ajustados para o bom funcionamento do algoritmo são os valores de μ e λ, e principalmente, a determinação da função objetiva. Para que fosse possível a avaliação dos valores obtidos foram utilizados, inicialmente, valores de sobressinal de 10% e tempo de acomodação de 1s. Além disso, incluiu-se um outro fator de avaliação das soluções baseado no deslocamento da ponta do braço, que deveria também ser minimizado.

A função objetiva utilizada foi:

$$f = 2 \cdot dm + dt + 10 \cdot pico + 100 \cdot df + 100 \cdot des \tag{19}$$

Na qual,

dm – diferença entre o sobressinal obtido e o desejado

dt – diferença entre o tempo de acomodação obtido e o desejado

pico – valor máximo de deslocamento da ponta do braço relativo ao referencial flutuante

df – diferença entre o ângulo θ obtido e o desejado (15°)

des – deslocamento médio da ponta do braço relativo ao referencial flutuante

Para essa implementação foram utilizados $\mu = 2$ e $\lambda = 30$.

A seguir, podemos observar como ocorre a evolução dos parâmetros em função das iterações ocorridas:

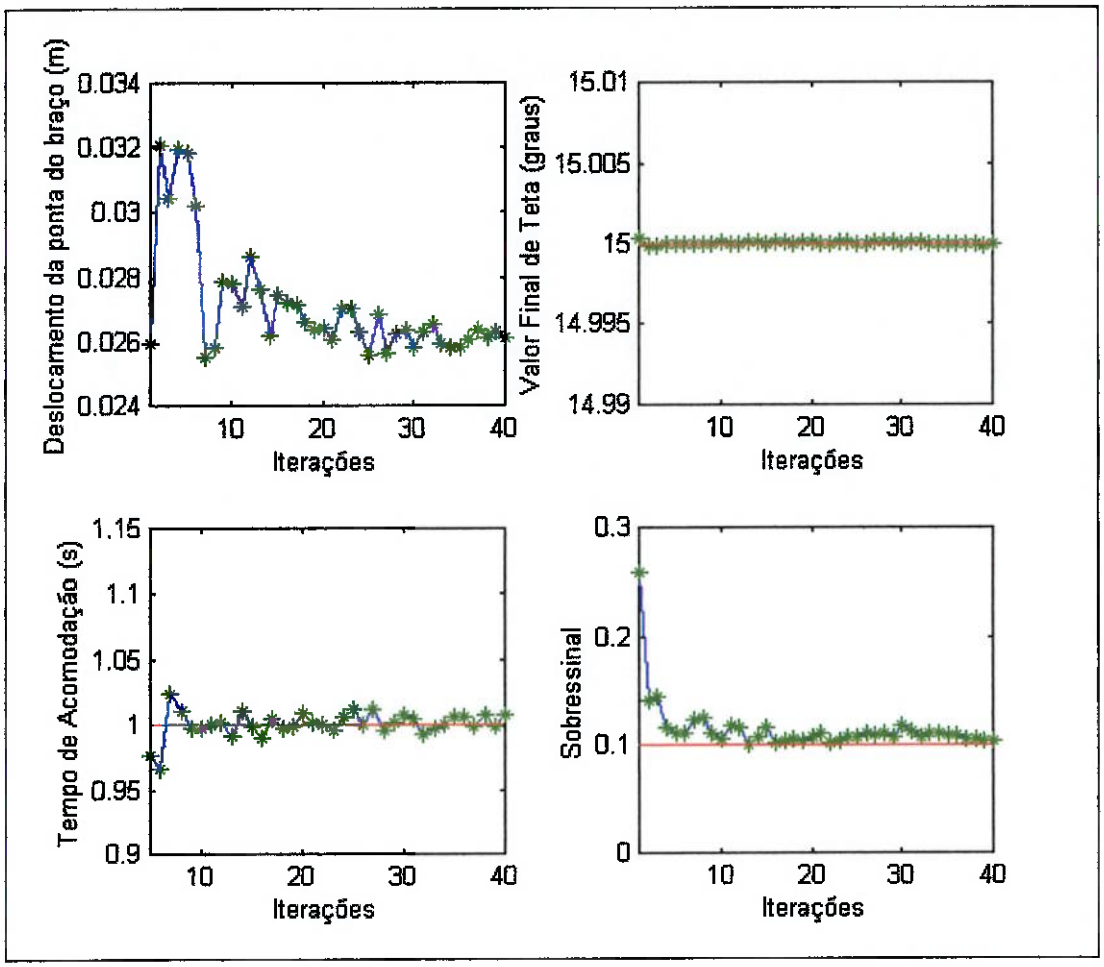


Figura 22- Características do sistema: deslocamento na ponta, θ, tempo de acomodação e sobressinal para sistema após iterações com valores iniciais fixos.

Verifica-se na figura 22 que a principal melhora obtida para o sistema foi o valor do sobressinal que inicialmente estava bastante distante do valor desejado e que depois da otimização ficou em aproximadamente 10% como desejado.

Dado que temos por valores iniciais ganhos fixos, podemos a partir deles buscar uma otimização local. Por esse motivo, o nosso algoritmo não precisará procurar em todo o domínio possível valores para as variáveis objetivas e, dessa forma, escolheu-se um valor baixo para μ, o que diminui bastante o esforço computacional.

A seguir, podemos observar o comportamento do braço para os ganhos calculados.

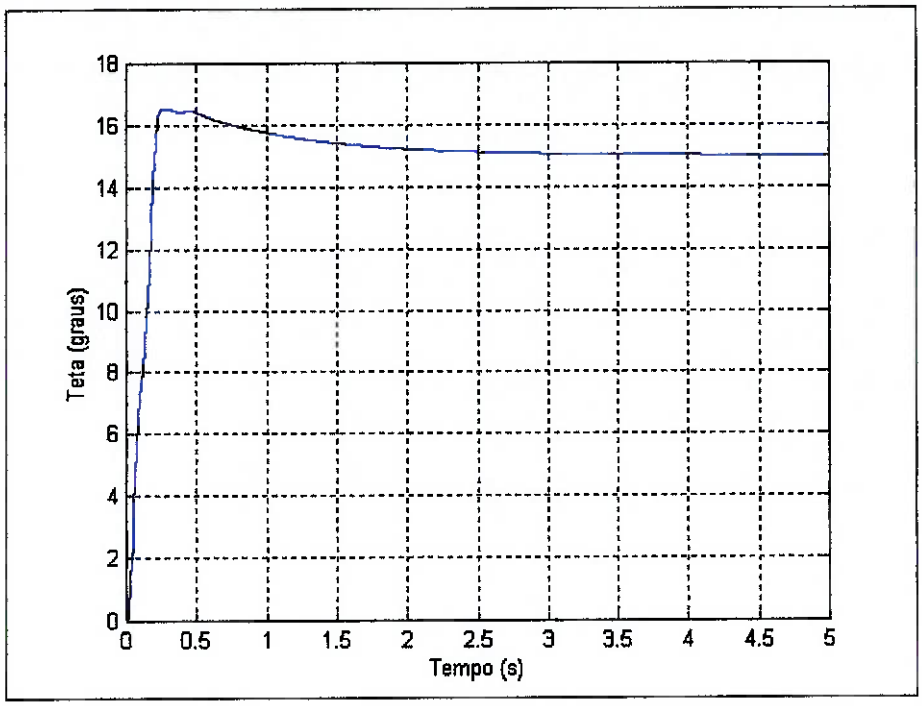


Figura 23- ângulo θ em função do tempo após iterações com valores iniciais dados.

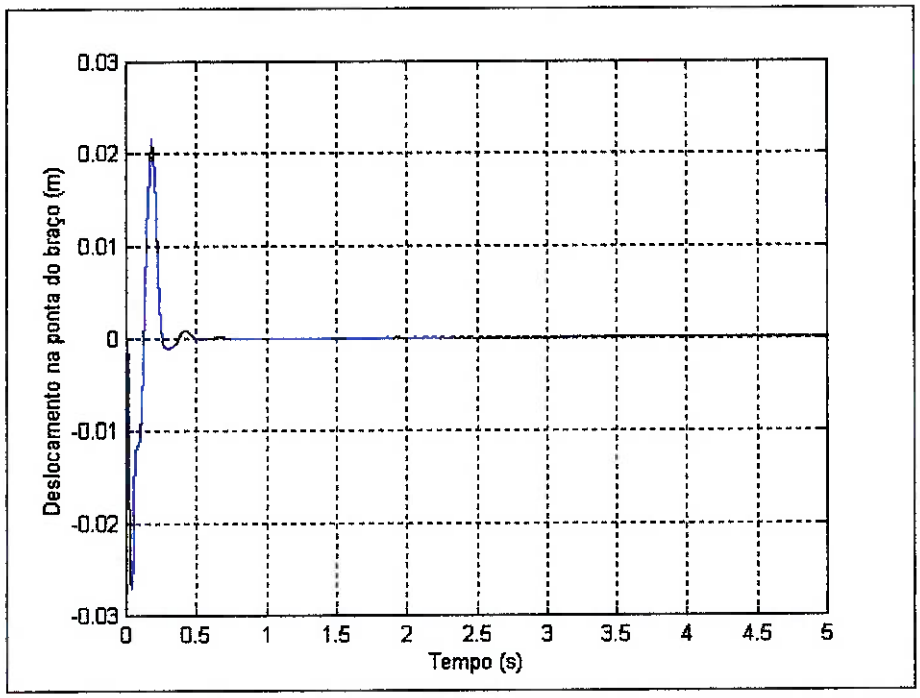


Figura 24- Deslocamento na ponta do braço após iterações com valores iniciais dados

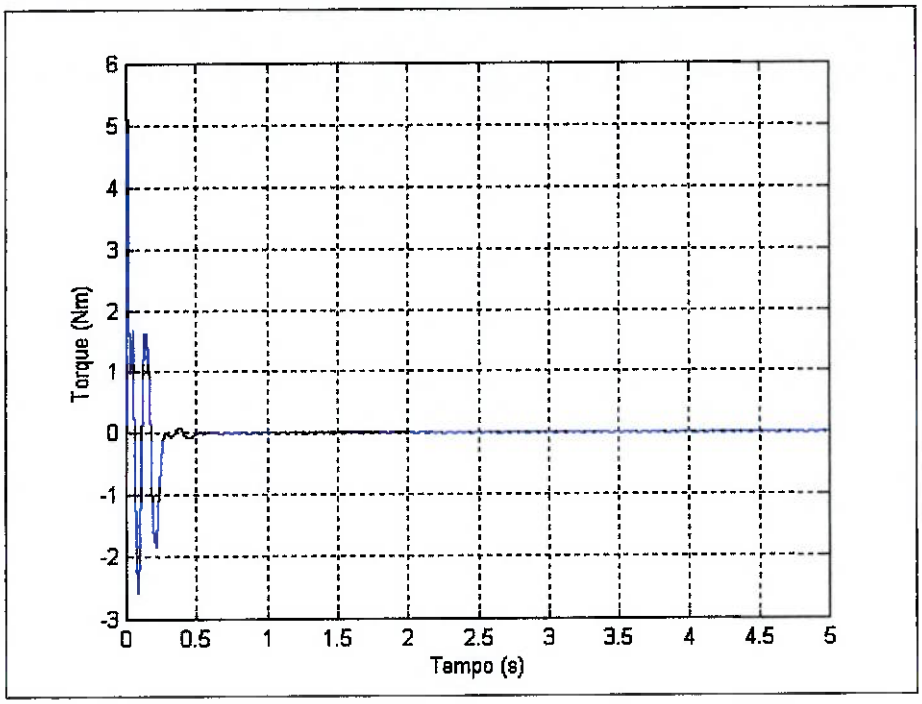


Figura 25- Torque no braço após iterações com valores iniciais dados

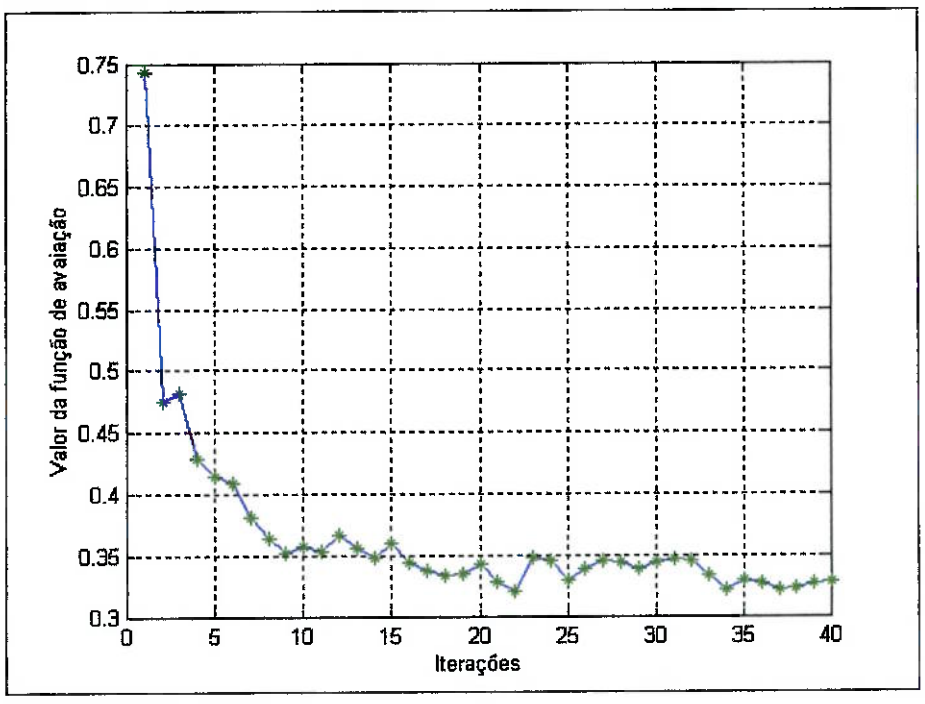


Figura 26- Evolução do valor da função de avaliação com as iterações.

Finalmente, podemos observar a evolução dos valores calculados pela função objetiva após cada iteração no gráfico da figura 26.

A partir dessa figuras, podemos comparar o movimento do braço com otimização dos parâmetros em relação ao comportamento inicial.

O deslocamento da ponta do braço em relação ao referencial flutuante, que é medida da vibração resultante do movimento do braço apresentou comportamento bastante interessante. Apesar do valor de pico obtido ser praticamente igual ao obtido para os valores iniciais de ganhos, pode-se observar pelas figuras que o número de oscilações é significativamente menor no segundo caso, no qual observa-se praticamente apenas uma oscilação completa.

Para o ângulo final obtido não houve variação, o que era desejado dado que o valor obtido para a solução inicial já era igual ao especificado. Por outro lado, para o sobressinal obtido verifica-se grande melhora, já que o mesmo passou de um valor inicial de 0,2583 para um valor de 0,1018 com os valores de ganho obtidos pelo método das estratégias evolutivas.

O tempo de acomodação passou de 0,97s para 1s, o que pode parecer uma piora, porém se lembrarmos que o valor desejado é de 1s vemos que o processo de otimização levou o valor do tempo de acomodação para o valor desejado e não para o menor possível, o que é bastante coerente.

Assim, é possível concluir que as estratégias evolutivas cumpriram seu papel ao otimizar os parâmetros do controlador já projetado em [17].

9.2 Otimização dos ganhos a partir de um valor inicial obtido estocasticamente

Para a implementação das estratégias evolutivas na solução desse problema foram utilizados os seguintes critérios:

- Estratégia Evolutiva do tipo $(\mu+\lambda)$-ES.
- População inicial gerada de maneira aleatória.
- Mutação linear, como em (12).

- Recombinação discreta para as variáveis objetivas e recombinação intermediária para os parâmetros estratégicos.
- Critério de parada baseado em número de iterações.

Assim como no caso anterior, os principais parâmetros a serem ajustados para o bom funcionamento do algoritmo são os valores de μ e λ, e a determinação da função objetiva. Da mesma forma, para que fosse possível a avaliação dos valores obtidos foram utilizados, inicialmente, valores de sobressinal de 10% e tempo de acomodação de 1s e o deslocamento da ponta do braço, que deveria também ser minimizado.

A função objetiva utilizada foi a mesma que a utilizada no caso anterior.

$$f = 2 \cdot dm + dt + 10 \cdot pico + 200 \cdot df + 100 \cdot des \tag{19}$$

Para essa implementação foram utilizados $\mu = 2$ e $\lambda = 6$.

Os resultados obtidos nesse caso podem ser vistos nas figuras a seguir:

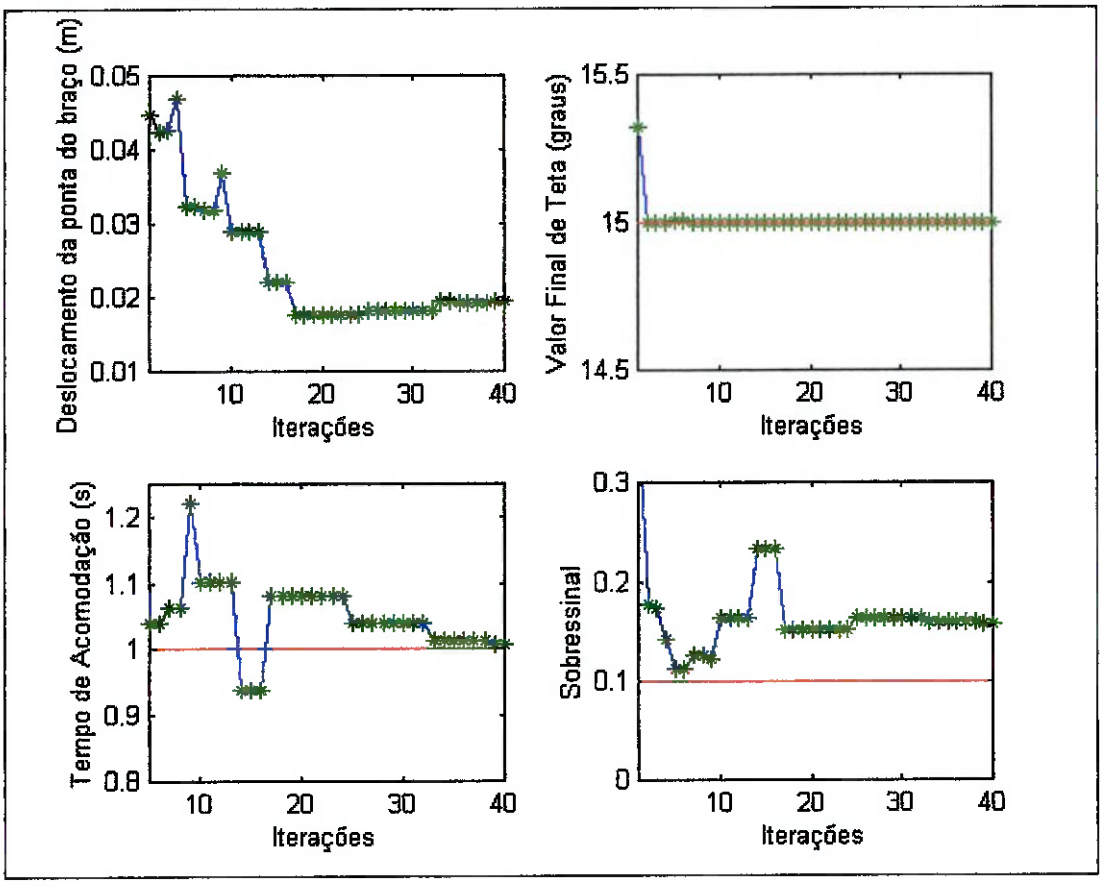


Figura 27- Características do Sistema - deslocamento, teta, tempo de acomodação, sobressinal com valores iniciais gerados estocasticamente

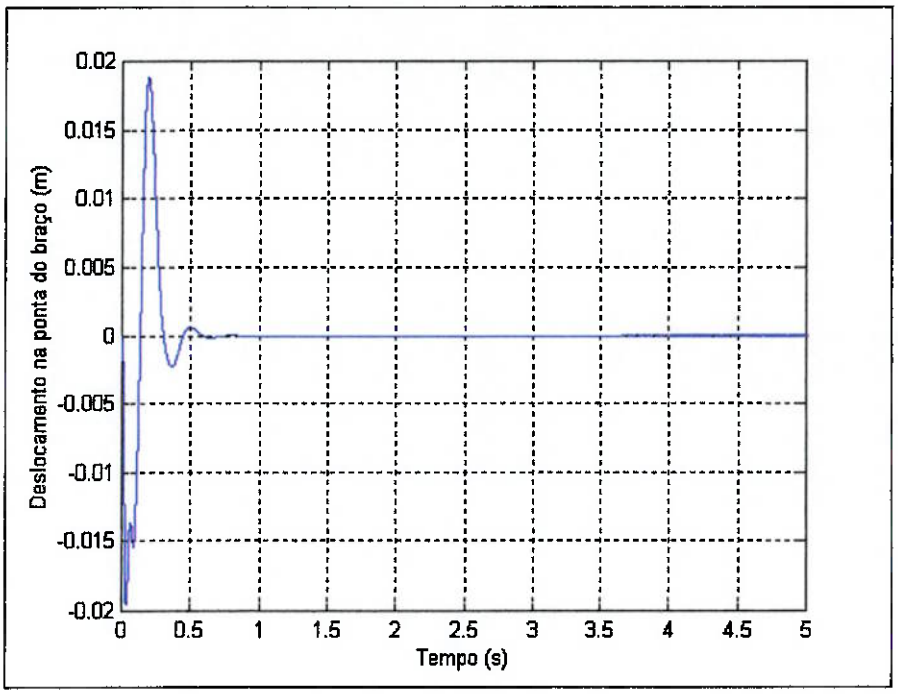


Figura 28- Deslocamento na ponta do braço, sistema com valores iniciais gerados estocasticamente

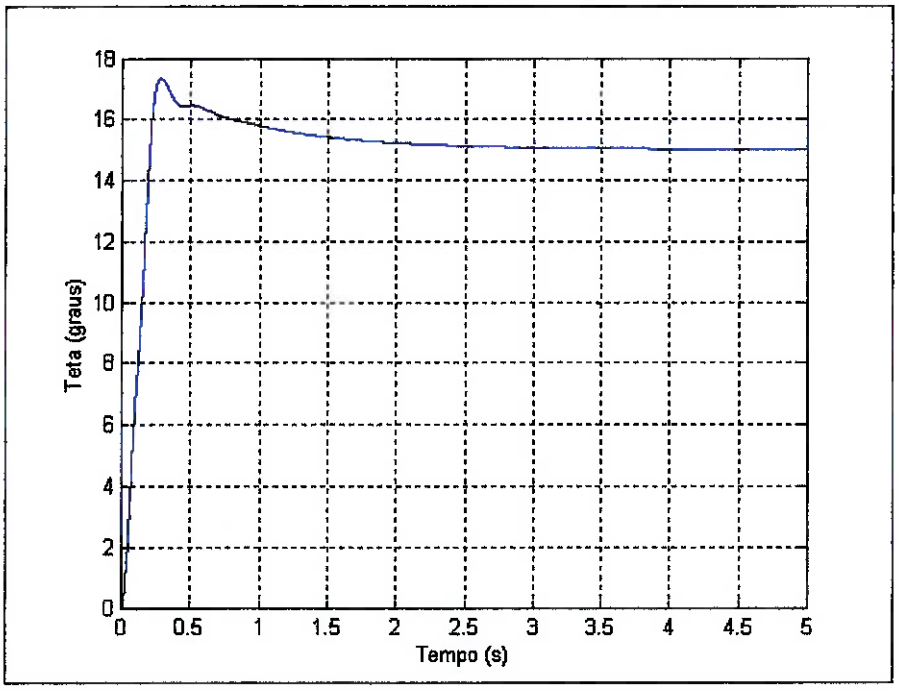


Figura 29- Angulo final θ, sistema com valores iniciais gerados estocasticamente

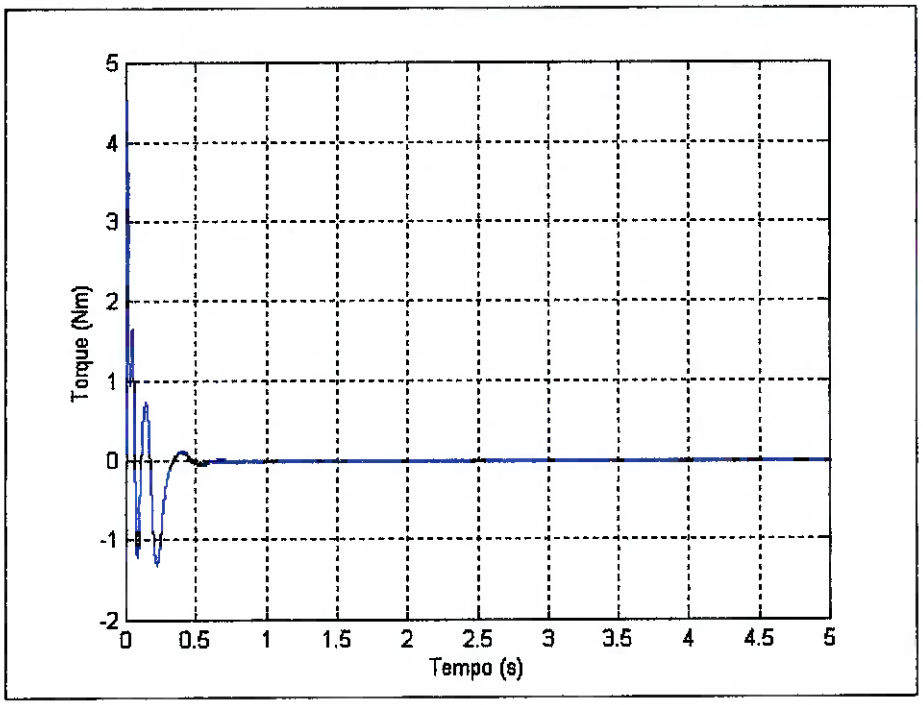


Figura 30: Torque em função do tempo em sistema com valores iniciais gerados estocasticamente

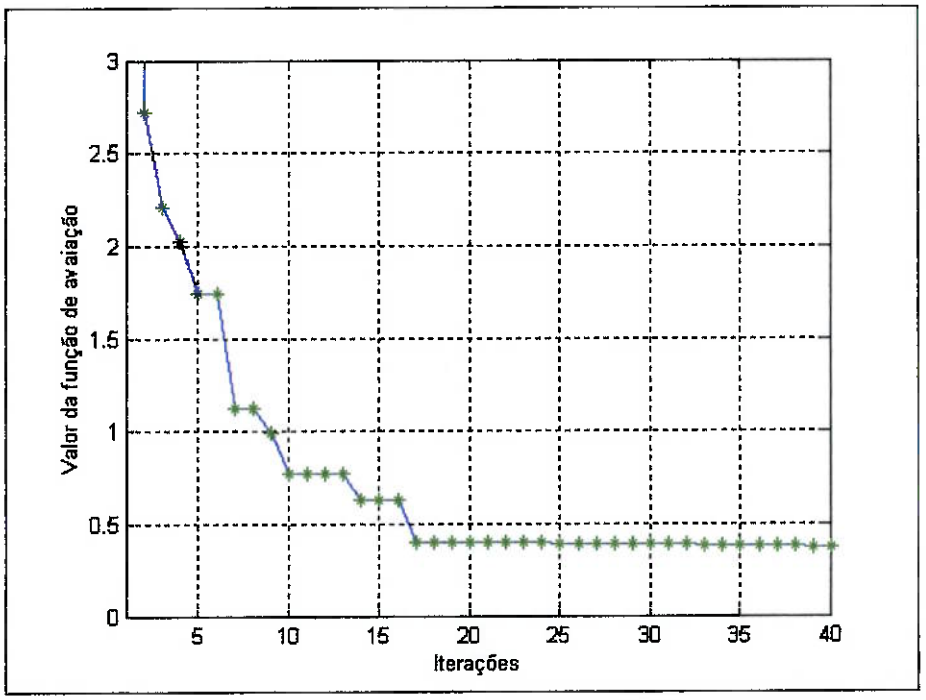


Figura 31- Valor da função de avaliação em sistema com valores iniciais gerados estocasticamente

Para essa implementação algumas considerações importantes devem ser feitas.

O primeiro ponto importante se refere ao fato de se ter usado estratégias evolutivas do tipo $(\mu+\lambda)$-ES ao invés de estratégias (μ,λ)-ES. Nas estratégias do tipo "$\mu+\lambda$" a formação da população no instante $\mathbf{t+1}$ é feita a partir dos μ pais e dos λ descendentes obtidos no instante $\mathbf{t}$ garante-se a permanência dos indivíduos mais aptos na população seguinte, o que pode não acontecer nas estratégias do tipo "μ,λ", nas quais pode-se perder boas soluções. Dessa forma, foram utilizadas $(\mu+\lambda)$-ES, pois a busca no espaço inteiro das possíveis soluções se mostrou uma tarefa bastante difícil, e com esse tipo de estratégias garante-se uma convergência mais fácil para as melhores soluções, que podem, entretanto, não serem as melhores possíveis. O resultado da utilização desse tipo de estratégias pode ser claramente visto contrastando-se as figuras (26) e (31).

Um segundo ponto importante são os valores de μ e λ utilizados. Inicialmente pensou-se em utilizar um valor bastante grande para μ, o que, possivelmente, levaria a melhores valores para chutes iniciais. Entretanto, a experiência mostrou que é mais fácil tentar o processo de otimização algumas vezes com valor pequeno de μ a uma única vez com valor grande de μ.

A partir dos resultados obtidos, podemos observar, também, que a implementação não produziu resultados tão bons quantos aqueles obtidos com um valor inicial determinado, mesmo depois de diversas tentativas.

Uma suposição seria que o máximo global seria muito 'estreito', de tal forma que conseguir um indivíduo em sua região seria extremamente difícil. Assim, deduzimos que o problema tem um ou mais máximos locais, "largos", onde a grande maioria dos indivíduos se coloca, e, por isso, a otimização se dá somente nos máximos locais, o que mostra que uma avaliação prévia do sistema pode significar a obtenção de resultados bastante melhores, principalmente para problemas, como este, que possuem mais de um máximo.

10 CONTROLADOR PID PARA O BRAÇO FLEXÍVEL

Partindo da idéia de um controlador por alocação de pólos não apresentar um caráter integrativo pensou-se na utilização de controladores PID para o sistema, melhorando assim a condição de regime do sistema. Entretanto, trata-se de um sistema de duas entradas e duas saídas e no qual o controle de ambas as variáveis – ângulo do braço e deslocamento na ponta – é igualmente importante.

Para o problema pensou-se em uma implementação como a mostrada na figura 32

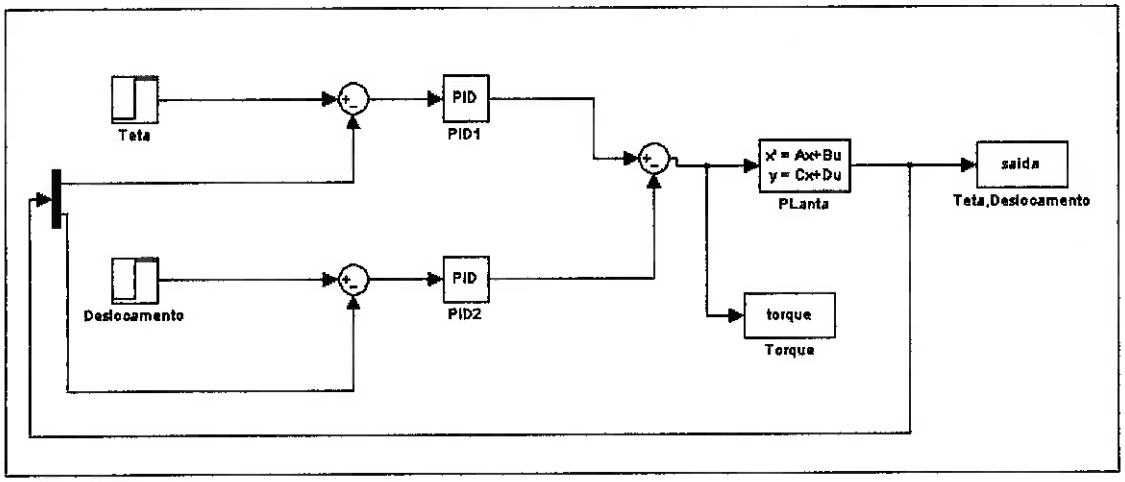


Figura 32- Modelo com 2 controladores PID

O sistema funciona da seguinte maneira: as variáveis a serem controladas, que são θ e o deslocamento, são as saídas da planta. Cada uma delas é subtraída dos valores de referência (15° para posição angular e 0 para o deslocamento), e esses valores são as entradas de cada um dos PID's. As saídas dos PID's são subtraídas uma da outra e o sinal resultante é entrada de torque para a planta.

10.1 Otimização dos ganhos com estratégias evolutivas

Para a implementação das estratégias evolutivas na solução desse problema foram utilizados os seguintes critérios:

- Estratégia Evolutiva do tipo (μ,λ)-ES.
- População inicial gerada de maneira aleatória.
- Mutação linear, como em (12).
- Recombinação discreta para as variáveis objetivas e recombinação intermediária para os parâmetros estratégicos.
- Critério de parada baseado em número de iterações.

Para que os resultados fossem comparados com os anteriores utilizou-se a mesma função objetiva que a utilizada para as duas outras implementações, ou seja:

$$f = 2 \cdot dm + dt + 10 \cdot pico + 200 \cdot df + 100 \cdot des \qquad (19)$$

Para essa implementação foram utilizados $\mu = 2$ e $\lambda = 10$.

Os resultados obtidos para esse caso podem ser vistos nas figuras 33, 34, 35, 36 e 37.

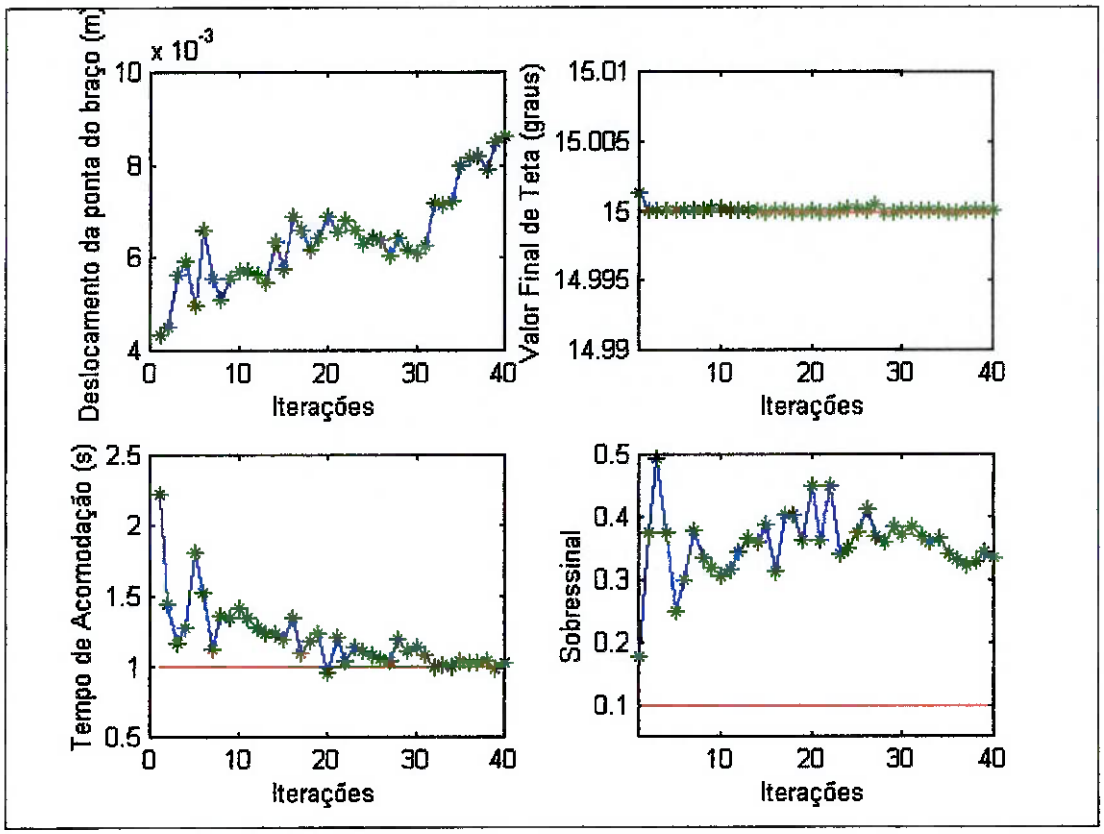


Figura 33- Características do sistema - Características do sistema: pico de deslocamento na ponta, teta, tempo de acomodação e sobressinal para sistema com PID´s.

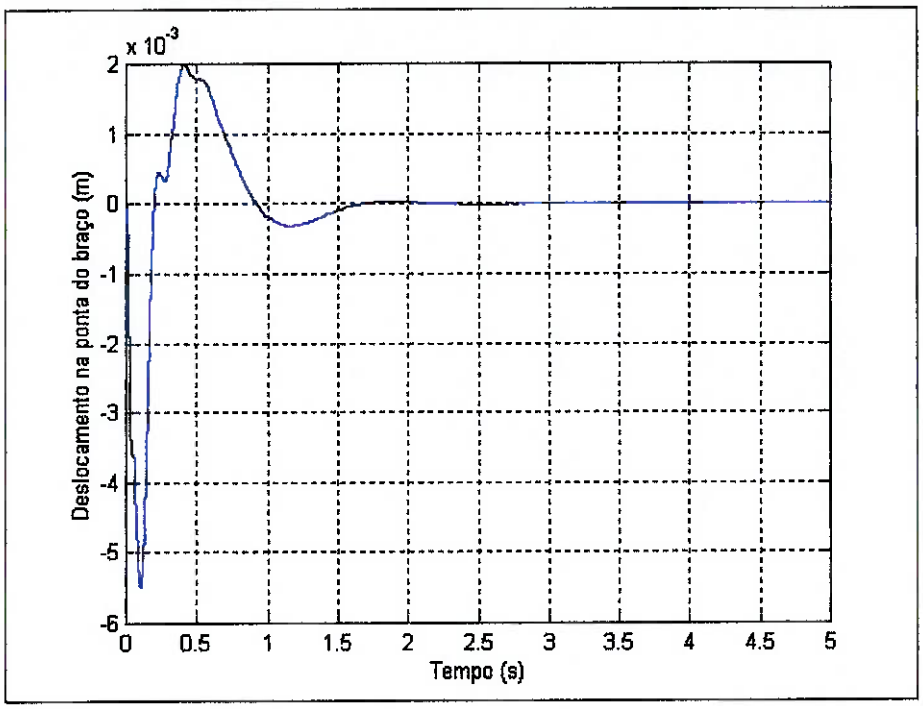


Figura 34- Deslocamento na ponta do braço em função do tempo com PID

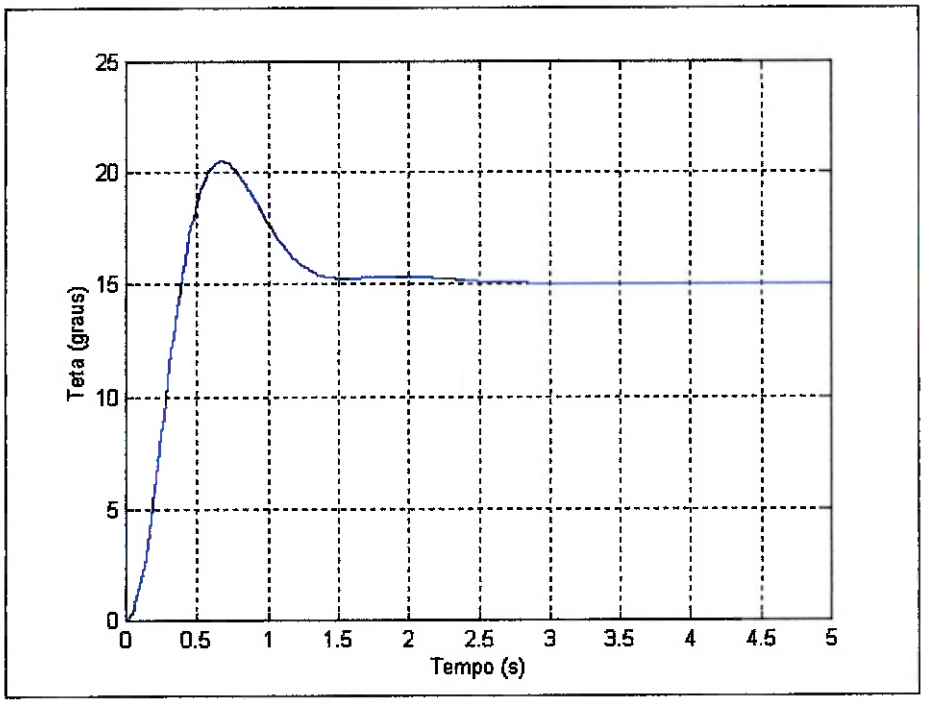


Figura 35- Angulo final θ em função do tempo com PID

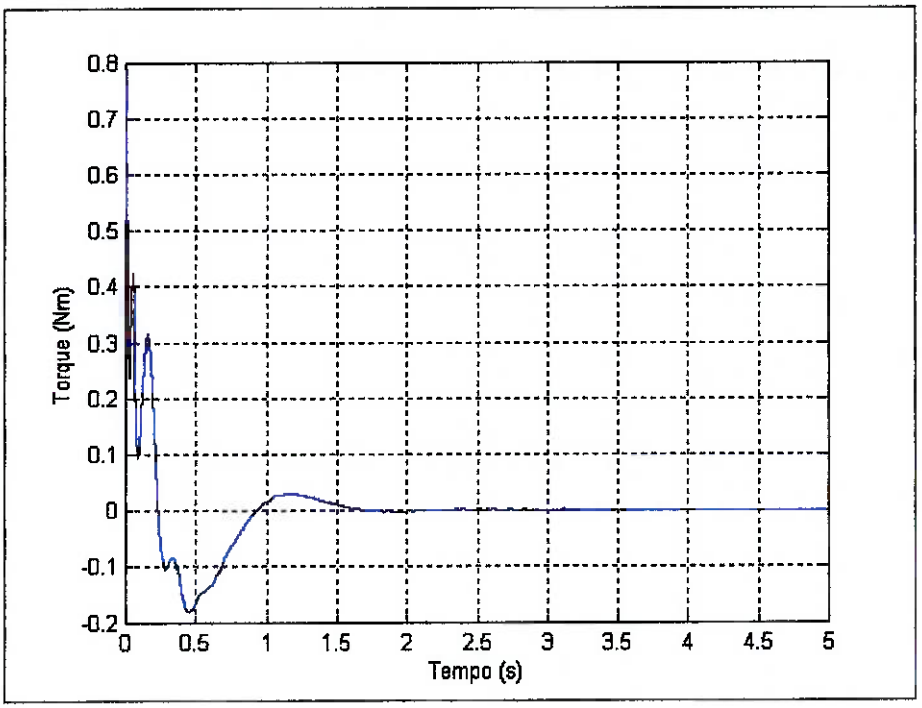


Figura 36- Torque em função do tempo com PID´s.

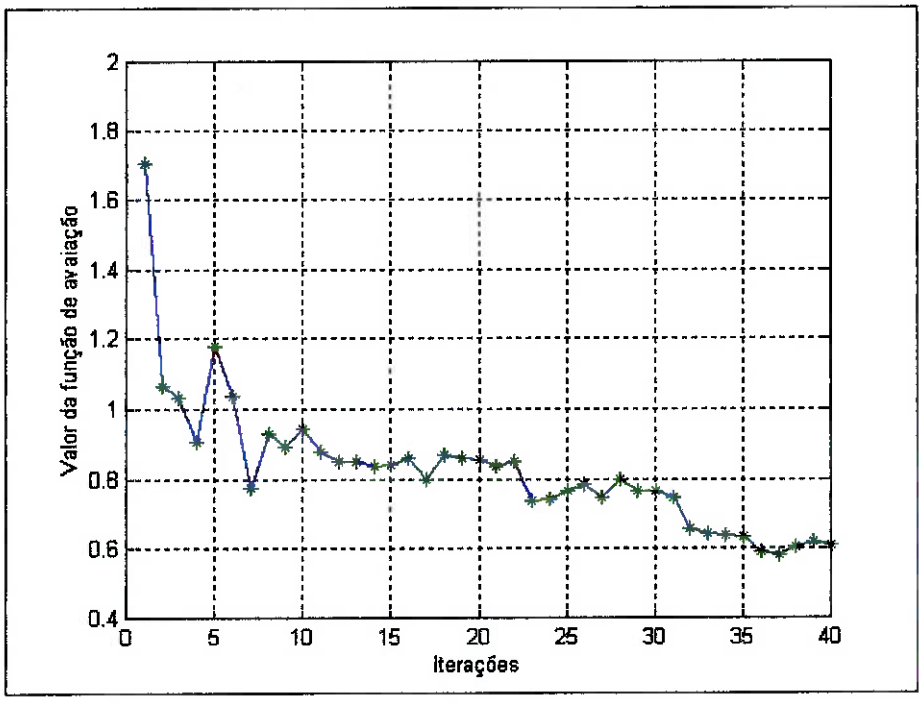


Figura 37- Valor da função de avaliação em função da iteração com PID´s

O primeiro ponto importante a ser mencionado é o fato do valor da função objetiva obtida nessa implementação ser maior que aquela obtida nos dois outros casos. No primeiro caso - com valor inicial fixo e controle por alocação de pólos - o valor obtido foi de aproximadamente 0,33; no segundo caso - com valores inicias aleatórios e controle por alocação de pólos – o valor foi de aproximadamente 0,46; já nessa implementação com controladores PID o valor obtido foi de 0,61.

O fato de função objetiva ter atingido um valor maior que nos outros casos não elimina dessa implementação algum mérito. Pode-se ver na figura 34 que o deslocamento na ponta do braço apresenta uma ordem de grandeza menor que a obtida nas outras implementações. Por outro lado, o valor do sobressinal obtido foi bastante distante do valor desejado. O torque necessário para a manobra também se mostra bastante menor que o requerido nos outros casos, como pode ser visto na figura 36.

Outro ponto interessante é o fato dos valores em regime serem excelentes. Na figura 22 vemos uma pequena oscilação no regime para o controlador por alocação de pólos, o que não é observado na figura 35, na qual não é possível se verificar qualquer oscilação no regime.

10.2 Outro resultado com os controladores PID

Ainda com a planta da figura 32 e com o algoritmo descrito no item 10.1 foram feitas experiências em relação à função objetiva utilizada. Em uma dessas experiências utilizou-se os seguintes parâmetros para a função objetiva:

Sobressinal: 1%

Tempo de acomodação: 1s

$$f = 20 \cdot dm + 5 \cdot dt + 10 \cdot pico + 1500 \cdot df + 1000 \cdot des \qquad (20)$$

Os resultados obtidos com essa função objetiva podem ser vistos nas figuras 38, 39 e 40.

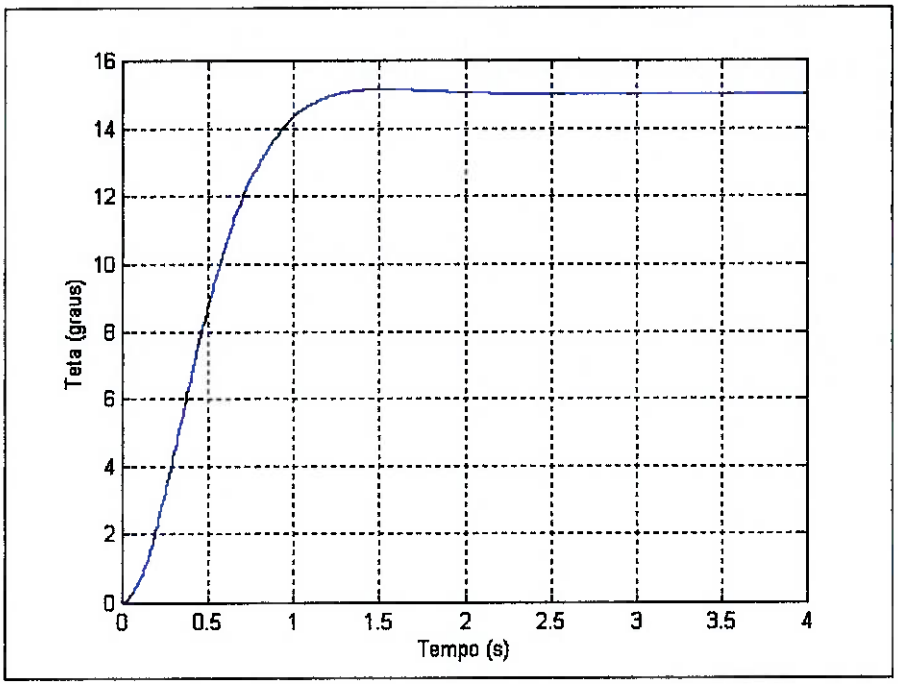


Figura 38: Ângulo Teta em função do tempo para nova função objetiva

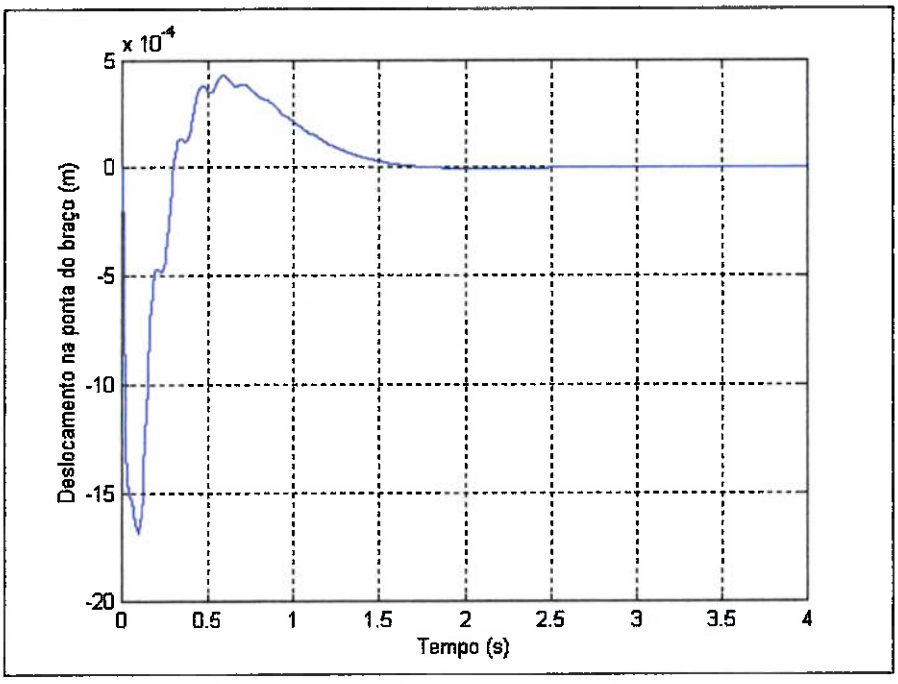


Figura 39: Deslocamento na ponta do braço para nova função objetiva

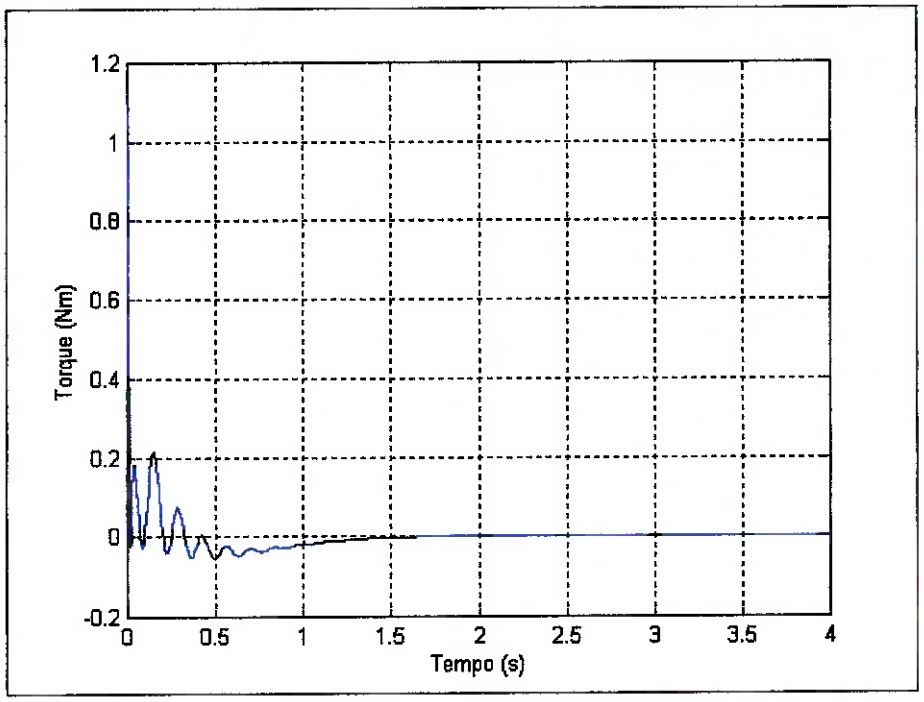


Figura 40: Torque de entrada na planta para nova função objetiva

É preciso que não se tenha a falsa impressão de que este resultado é muito melhor que os obtidos até aqui, o que seria muito natural ao se observar o valor do deslocamento na ponta do braço que chegou a valores inferiores a 1mm. Entretanto, o sobressinal foi aproximadamente nulo - o que pode parecer bom para o posicionamento de um braço robótico – estando em desacordo com os objetivos estabelecidos inicialmente.

Por outro lado essa solução foi avaliada com a função objetiva estabelecida em (19) e os resultado obtido foi 0,35. O melhor resultado havia sido 0,33 para o controlador por alocação de pólos feito a partir de valores iniciais fixos, e o resultado obtido para essa estratégia com controladores PID havia sido 0,61.

Concluímos desse fato que existem outras funções objetivas que podem levar a soluções melhores para o problema estabelecido pela função (19). De maneira simplificada isso significa que a função objetiva influi significativamente no "caminho" tomado pela evolução no espaço das possíveis soluções do problema até um valor

considerado ótimo. E esses caminhos podem ser de tal forma que se torne impossível a obtenção de máximos/mínimos globais para o problema.

A partir desse ponto será feita, então, uma distinção entre a função objetiva do problema e o que chamaremos de funções de busca. A idéia é que se possa obter funções de busca que quando otimizadas levem a indivíduos que estejam próximos dos máximos globais da função objetiva do problema. E após ter-se atingido essas regiões uma implementação tradicional das estratégias evolutivas, usando esses indivíduos como população inicial leva à melhor solução do problema. No próximo capítulo propõe-se uma nova metodologia para a obtenção das recém batizadas funções de busca.

11 UMA CONTRIBUIÇÃO PARA A BUSCA DE MÁXIMOS GLOBAIS

Os resultados obtidos nas seções 9.1 – controlador por alocação de pólos otimizado a partir de valores iniciais dados – e 9.2 – controlador por alocação de pólos a partir de valores iniciais aleatórios – mostram a importância da população inicial para a obtenção de boas soluções para um problema resolvido com estratégias evolutivas.

Para a compreensão desse fato pode-se pensar em um problema de otimização de uma única variável objetiva para o qual a função objetiva é como a representada na figura 42. Percebemos que a obtenção do máximo global será bastante difícil através de estratégias evolutivas, pois dificilmente ter-se-á um indivíduo da população inicial próximo ao máximo global – bastante estreito. Dessa forma, os indivíduos tenderão sempre a evoluir em direção ao máximo local, mais "largo".

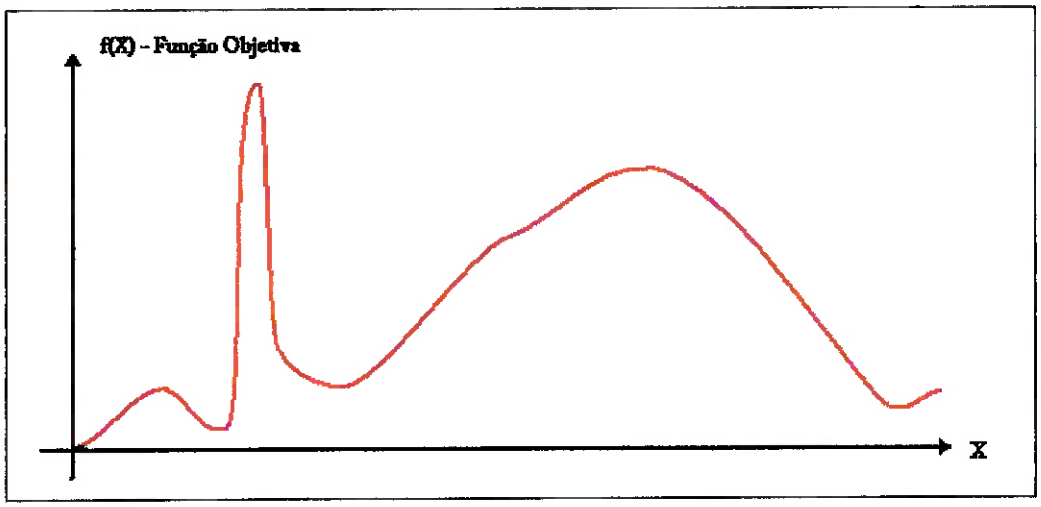


Figura 41: Exemplo de problema de otimização com máximo global estreito

Inicialmente pensamos que basta que seja gerada uma população inicial suficientemente grande para que pelo menos um de seus componentes esteja nas proximidades do máximo global. Porém, pelo menos no problema estudado nesse trabalho, essa idéia não pode ser confirmada. O aumento do número de indivíduos da população levou a um esforço computacional extremamente grande, o que inviabilizou a

busca pela melhor solução. Por esse motivo é que propomos uma nova metodologia para a obtenção dos máximos globais.

Pelos resultados descritos no capítulo 10, concluímos que não necessariamente a busca por soluções que maximizem uma certa função objetiva precisa ser feita utilizando a própria função objetiva e por esse motivo definimos a idéia de função de busca. Um outro exercício mental para que se possa entender essa idéia está representado na figura 42.

Se encontrássemos uma função cujo máximo mais fácil de ser obtido, quer global ou não, estivesse próximo do máximo global da função original, poder-se-ia obter o máximo dessa função auxiliar e utilizar então esse valor de **x** como chute inicial para o processo de otimização na função original. Essa função auxiliar teria, então, um papel de "guiar" as soluções por "caminhos" que com a utilização da função objetiva original seriam muito pouco prováveis de serem percorridos, e por esse motivo é que essas funções serão chamadas funções de busca.

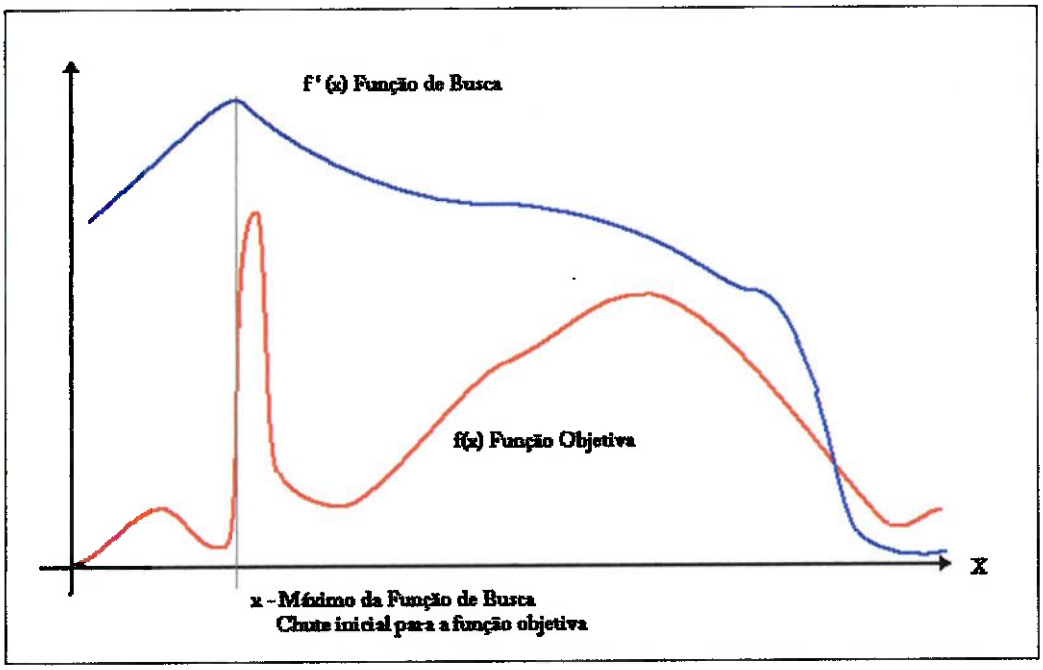


Figura 42: Exemplo de função de busca

A idéia é encontrar algumas funções de busca que possam fornecer valores iniciais para a otimização na função objetiva em si. Para isso, inicialmente utilizamos um processo de tentativa e erro. Escolheu-se aleatoriamente uma função de busca, utilizou-se uma estratégia evolutiva para encontrar o máximo dessa função e por fim avaliou-se o resultado obtido com a função objetiva original. Esse processo foi repetido algumas vezes sem sucesso até que percebemos que se tratava de um novo problema de otimização, o de se encontrar os parâmetros da função de busca.

O método utilizado para a otimização desses parâmetros foi novamente uma implementação das estratégias evolutivas para a qual os indivíduos seriam conjuntos de valores de parâmetros para as funções de busca e cuja adaptação seria o valor da função objetiva original calculada no ponto obtido pela otimização através da função de busca em questão.

11.1 Implementação para o caso do movimento do braço flexível com controladores PID

Foi implementado um algoritmo como o proposto para a resolução do problema do controle do movimento do braço flexível de acordo com o sistema proposto na figura 32.

Para as funções de busca desse problema deve-se determinar sete parâmetros:

- Sobressinal desejado (**A**)
- Tempo de acomodação desejado (**B**)
- Peso da distância entre o sobressinal desejado e o obtido (**C**)
- Peso da distância entre o tempo de acomodação desejado e o obtido (**D**)
- Peso do deslocamento de pico da ponta do braço em relação ao referencial flutuante (**E**)
- Peso do deslocamento médio da ponta do braço em relação ao referencial flutuante (**F**)

- Peso da distância do ângulo θ final obtido e o desejado (**G**)

O primeiro passo para a implementação foi a determinação de uma função de avaliação para um conjunto de valores (A,B,C,D,E,F,G) dado. Para isso fez-se uma estratégia evolutiva do tipo (2+10)-ES que fornece os melhores valores de ganhos para os controladores PID's para essa função determinada pelos parâmetros A, B, C, D, E, F e G. O sistema é novamente simulado para esses valores de ganhos e o resultado é analisado pela função (19).

Dado que se pode avaliar, então, a qualidade de uma função de busca fica evidente a implementação de uma outra estratégia evolutiva para obtenção dos melhores valores de A, B, C, D, E, F e G para o problema. Para isso implementou-se então uma estratégia evolutiva do tipo (4+6)-ES.

A cada geração desse segundo processo de otimização armazenou-se o valor de ganhos para os controladores que resultaram nos menores valores para a função (19). Foram obtidas apenas 5 gerações para esse processo, e esses cinco conjuntos de valores de ganhos foram utilizados, finalmente, como população inicial para uma última estratégia do tipo (5+25)-ES que tinha, essa sim, a função (19) como função de avaliação.

Os resultados obtidos para essa implementação podem ser vistos nas figuras 43, 44 e 45, que mostram o ângulo θ do braço, o deslocamento da ponta do braço em relação ao referencial flutuante e o torque fornecido à planta, respectivamente.

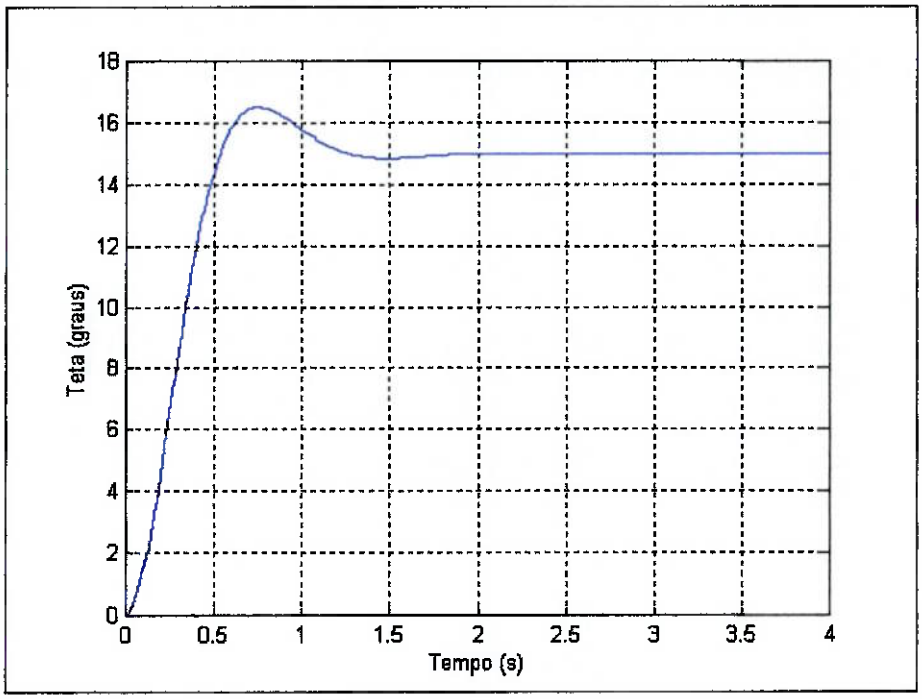


Figura 43: Ângulo Teta para o resultado final com controladores PID

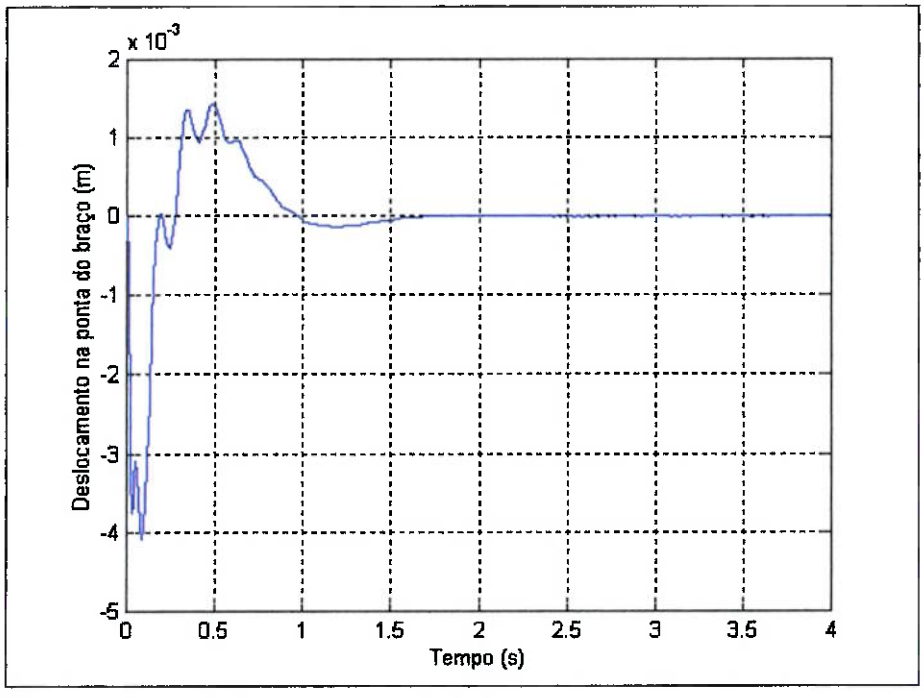


Figura 44: Deslocamento da ponta do braço para resultado final com controladores PID

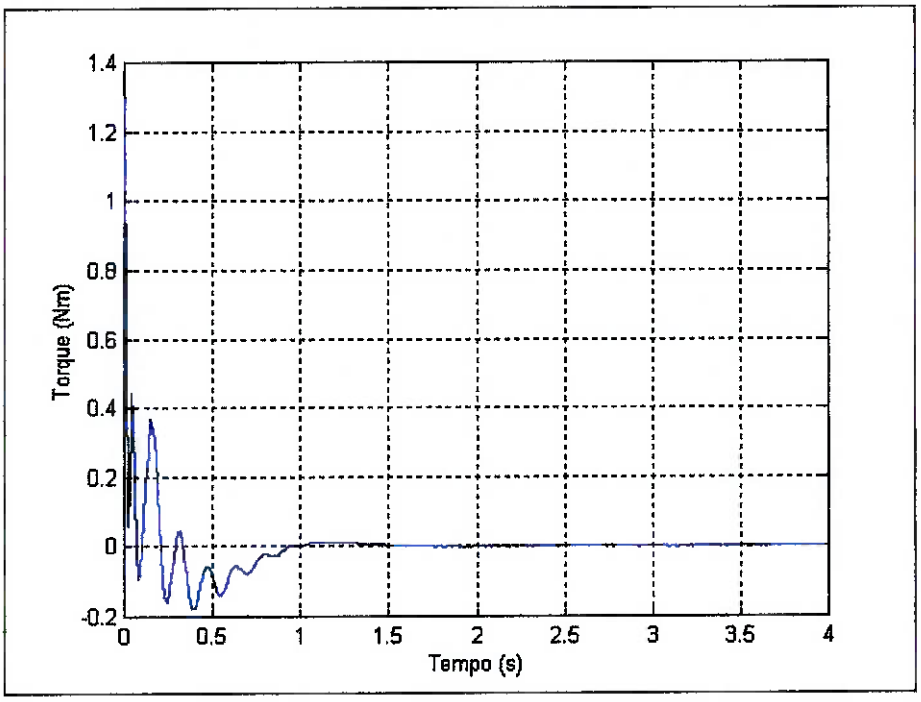


Figura 45: Torque fornecido à planta para resultado final com controladores PID

O valor final para a função objetiva original (19) para a solução obtida nesse procedimento foi aproximadamente 0,068, o que é um resultado muito superior aos obtidos até aqui. Mostrando que, pelo menos nesse caso, o procedimento proposto foi bastante eficiente.

12 RESULTADOS E CONCLUSÕES

A tabela a baixo mostra todos os resultados obtidos nesse trabalho. Nos três primeiros casos – Seções 9, 9.1 e 9.2 -, nos quais utilizou-se um controlador por alocação de pólos são mostradas as matrizes de ganhos equivalentes a esses controladores, e nos três outros – Seções 10.1, 10.2 e 11 -, quando foram utilizados os controladores PID são mostrados os ganhos dos mesmos na forma Kp + Ki/s + Kd.s, para cada um dos controladores. Na tabela pode-se ver também os valores obtidos para cada um dos casos para a função objetiva estabelecida em (19) (**fo**) e os valores de sobressinal (**Mp**), tempo de acomodação (**ts**), pico de deslocamento na ponta do braço em relação ao referencial flutuante (**dp**) e finalmente o ângulo final do braço(θf).

Seção	Controlador	fo	Mp	Ts (s)	dp (m)	θf (°)
9	[7,5 2,68 -65,28 -0,15 -82,62 -0,27]	0,75	0,26	0,97	0,026	15
9.1	[7,48 6,44 -65,61 -0,72 -123,22 -2,85]	0,33	0,10	1,0	0,027	15
9.2	[25,64 20,63 -189,83 -8,01 -1225,8 -17,21]	0,46	0,16	1,01	0,02	15
10.1	[3,28 3,51 0,53] e [18,56 34,7 1,59]	0,61	0,33	1,0	0,0085	15
10.2	[4,18 0,00005 1,9525] e [440,18 0,74 7,94]	0,35	0,01	0,97	0,0017	15
11	[4,98 0,00004 0,98] e [200,56 267,12 2,38]	0,068	0,10	1,0	0,004	15

Tabela 2: Resultados para todas as simulações

A primeira conclusão importante do trabalho é a comprovação da eficiência dos algoritmos evolutivos, principalmente das estratégias evolutivas para a solução de problemas de controle. Essa conclusão não é tirada exclusivamente dos resultados obtidos nesse trabalho, mas principalmente a partir da ampla gama de resultados descritos na literatura.

O trabalho mostrou resultados bastante satisfatórios para o controle do braço flexível, dada a comparação com os resultados obtidos por métodos tradicionais [17]. Além disso, a implementação das estratégias evolutivas se mostrou bastante simples, e não requisitou esforço computacional exagerado, ou seja, o método pode ser executado em um computador pessoal em um tempo da ordem de minutos, podendo assim fornecer resultados para problemas de otimização em tempo bastante reduzido, mesmo que se trate de problemas de grande complexidade e difícil modelagem.

Como ponto principal para o bom desempenho do método observa-se uma configuração acertada dos números de indivíduos nas populações e, principalmente, os valores utilizados na população inicial.

Quanto à população inicial gerada de maneira estocástica, observa-se uma necessidade de gerar indivíduos que possam varrer a maior parte do espaço das possíveis soluções, e que esses indivíduos estejam uniformemente distribuídos pelo espaço. Além disso, é importante que o passo de mutação inicial seja estabelecido de acordo com os valores iniciais de cada uma das variáveis objetivas, e que inicialmente esses valores sejam grandes o suficiente para permitir que as próximas gerações tenham indivíduos em regiões que ainda não haviam sido "exploradas".

Uma possibilidade de geração de população inicial que se mostrou bastante interessante é a utilização de valores obtidos por métodos tradicionais para a resolução dos problemas. Esses valores sofrerão, então, uma otimização local que leva rapidamente a resultados bastante satisfatórios e, pelo menos, melhores que os obtidos pelos métodos tradicionais.

Caso não haja um procedimento tradicional para a resolução do problema, como no caso da sintonia dos controladores PID no sistema proposto na figura 32, propõe-se nesse trabalho um procedimento alternativo para a geração da população inicial. Esse procedimento está descrito no capítulo 11 e, evidentemente, necessita de futuras comprovações para sua eficiência. No caso estudado nesse trabalho o procedimento levou a resultados excelentes como se pode ver na Tabela 2.

Os algoritmos evolutivos se revelam hoje em dia uma alternativa muito interessante para a resolução de problemas de otimização, podendo inclusive ser

utilizados em problemas que apresentem não linearidades, descontinuidades ou outras características que são as grandes causas de problemas para os métodos tradicionais de otimização. Além disso, os algoritmos podem ser usados em problemas práticos de difícil modelagem, podendo ser aplicados na própria planta, sendo cada solução avaliada pela resposta medida da planta.

O trabalho cumpriu dois papéis importantes. O primeiro de resolver o problema de controle para o movimento do braço flexível, e o segundo de servir como apoio para quaisquer tentativas posteriores de implementação de estratégias evolutivas para a solução de problemas de controle.

13 BIBLIOGRAFIA

[1] K. Price and R. Storn, "Differential Evolution", Dr. Dobb's Journal, pp. 18, 1997.

[2] Mitchell and Melanie, "An Introduction to Genetic Algorithms", 1997.

[3] Nissen, Volker, "Einführung in Evolutionäre Algorithmen: Optimierung nach dem Vorbild der Evolution", Braunschweig, Wiesbaden, Vieweg 1997.

[4] Seebach, Gündisch, "Evolutionsstrategien", 1999

[5] Bäck, Hoffmeister, Schwefel, "A Survey of Evolution Strategies",

[6] Hildebrand, Lars, "Ein Ansatz zur Erweiterung von Evolutionsstrategien um gerichtete Mutation"

[7] Günter R.,"On Correlated Mutations in Evolution Strategies".

[8] Hansen, N and Ostermeier, A., "Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: The Covariance Matrix Adaptation". In *Proceeding of the 1996 IEEE Intern. Conf. On Evolutionary Computation, 312-317.*

[9] Goldberg, David E., "Genetic Algorithms in Search, Optimization, and Machine Learning", 1989

[10] Kalyanmoy, Déb "Genetic Algorithm in Search and Optimization: The Technique and Applications"

[11] Fogel, David B. "Evolutionary Computation: Toward a New Philosophy of Machine Intelligence", 1995

[12] D.W. Person, N.C Steele and R. F. Albrecht "Artificial Neural Nets and Genetic Algorithms", 1995

[13] Binh, T. and Korn, U.; "An evolution strategy for the multiobjective optimization".

[14] Koumoutsakos, Freund and Parekh; "Evolution strategies for parameter optimization in jet flow control"

[15] Richter, R. and Hofmann, W.;"Optimized control on a two axis – robot by means of Evolution Strategies.

[16] Coelho, Almeida and Coelho; "Design and Tunning of Intelligent and Self-Tuning PID Controllers."

[17] Gildin, Eduardo, "Desenvolvimento de um Controlador Adaptativo para Manipuladores Flexíveis com Incertezas de Carga", 1998

ANEXO A

Códigos dos programas em MatLab

I) Códigos para otimização da alocação de pólos com chute inicial dado

1) MI_LAMBDA_BRACO.M

```
% Programa Principal - Otimizacao de Alocacao de Polos

clear
global A B C D G K Ident moptions aux tout saida saida2

acha_matrizes  % Gera o modelo do simulink
open_system('braco') %Linha utilizada caso o modelo ja exista

% Parâmetros para o algoritimo
mi = 2;         % mi indivíduos por geração
lambda = 6;    % lambda > mi, lambda filhos gerados
n = 6;          % seis variáveis a serem otimizadas
delta = 2;      % Parâmetro para a mutação - passo para a variação de sigma
%moptions = simset('DstWorkspace','base','FixedStep',0.002,'Solver','ode5');

moptions = simset('DstWorkspace','base');
adaant = inf;

for iteracao = 1: 1
   % Processo de optimização
   Pt = Po(mi,n) % Geração da população inicial - Chute inicial igual ao do Gildin
   %Ptlinha = Pt;
   %for i = 2: lambda/mi
       %Ptlinha = [ Pt ; Ptlinha ];
   %end
   Ptlinha = m(r(Pt,lambda),delta);    % Geração dos lambda descendentes - Utilizado com
início estocástico
   tempo = 1;
   Ptlinha = agrupa(Pt,Ptlinha)
   for i = 1: lambda
      adaptacao(i) = f_aloc(Ptlinha,i);
   end
   melhor = acha_menor(adaptacao);
   Ktodos(iteracao,tempo,:) = Ptlinha(melhor,:,1);
   Valortodos(iteracao,tempo) = adaptacao;
   for i = 1: 6
      K(i) = Ptlinha(melhor,i,1);
   end
   sim('braco', 8, moptions) % Devolve as variáveis tout e y
   Mptodos(iteracao,tempo) = abs(sobressinal(tout,saida(:,1)));
   tstodos(iteracao,tempo) = tempo_s(tout,saida(:,1));
   finaltodos(iteracao,tempo) = valor_final(saida(:,1));
   mediatodos(iteracao,tempo) = soma_des(saida(:,2),tout); %deslocamento médio no tempo
   picotodos(iteracao,tempo) = des_pico(saida(:,2));

   fim = t_aloc(Ptlinha,tempo);
   melhordt = inf;
   while  fim == 0
      clear Pt;
      Pt = s_aloc(Ptlinha,mi);
      Ptlinha = m(r(Pt,lambda),delta);
      Ptlinha = agrupa(Pt,Ptlinha);
      tempo = tempo + 1;
      fim = t_aloc(Ptlinha,tempo);
```

```
      for i = 1: lambda
         adaptacao(i) = f_aloc(Ptlinha,i);
      end
      melhor = acha_menor(adaptacao);
      K = Ptlinha(melhor,:,1);
      Ktodos(iteracao,tempo,:) = Ptlinha(melhor,:,1);
      Valortodos(iteracao,tempo) = adaptacao(melhor);
      sim('braco', 8, moptions) % Devolve as variáveis tout e y
      Mptodos(iteracao,tempo) = abs(sobressinal(tout,saida(:,1)));
      tstodos(iteracao,tempo) = tempo_s(tout,saida(:,1));
      finaltodos(iteracao,tempo) = valor_final(saida(:,1));
      mediatodos(iteracao,tempo) = soma_des(saida(:,2),tout); %deslocamento médio no
tempo
      picotodos(iteracao,tempo) = des_pico(saida(:,2));
      Valortodos(1,tempo)
   end
end

close_system('braco')
```

2) ACHA_MATRIZES.M

```
% Procedimento para determinação do modelo do braço
% 11.11.02
global mp L b ha vol ro massa Ih I I1 E
% --------------- D A D O S -------------------
mp = 0.1;   % Massa na ponta da barra. Único parâmetro alterável
L = 0.7;
b = 0.0254;
ha = 0.001;
vol = L*b*ha;
massa = 2710*vol;
ro = massa/L;
Ih = 1.29e-4;
I = (b*ha^3)/12;  % Momento de inércia da viga
I1 = (ha*b^3)/12;  % outro eixo
E = 6.3e10; % Módulo de elasticidade

% --- D E T E R M I N A Ç Ã O  D O S  P A R Â M E T R O S ---------
% Procedimento para determinação das matrizes do modelo no espaço de estados

chutes = [ 5.3 8.7 11.8 16.0 20.3];
for i = 1: 2
   is = num2str(i);
   betar(i) = fzero('massap',chutes(i));
   alfar(i) = betar(i)^4*Ih/ro;
   gamar(i) = betar(i)^4*mp/ro;
   w(i) = sqrt(betar(i)^4*E*I/ro);
   aux = betar(i)*L;
   cb = cos(aux);
   sb = sin(aux);
   shb = sinh(aux);
   chb = cosh(aux);
   b3 = betar(i)^3;
   A = [ 0 1 0 1 ; alfar(i) -betar(i) alfar(i) betar(i) ; -sb -cb shb chb ; -
b3*cb+gamar(i)*sb b3*sb+gamar(i)*cb b3*chb+gamar(i)*shb b3*shb+gamar(i)*chb ];
   x(4) = 1;
   B = [ -1 ; -betar(i); -chb; -b3*shb-gamar(i)*chb];
   x(1:3) = A(1:3,1:3)\B(1:3);
   dL = L/100;
   xp = 0:dL:L;
   % Modos de vibração
   for j = 1 : length(xp)
      y = betar(i)*xp(j);
      Fi(i,j) = x(1)*sin(y) + x(2)*cos(y) + x(3)*sinh(y) + x(4)*cosh(y);
   end
```

```
   Fil(i) = betar(i)*(x(1)+x(3));
   % Normalização
   aux1 = 0;
   for j = 1: length(xp)-1
      aux1 = aux1 + ro*dL*((Fi(i,j)+Fi(i,j+1))/2)^2;

   end
   delta = aux1 + Ih*(Fil(i)^2) + mp*(Fi(i,1)^2);
   x = x/sqrt(delta);
   for j = 1 : length(xp)
      y = betar(i)*xp(j);
      Fi(i,j) = x(1)*sin(y) + x(2)*cos(y) + x(3)*sinh(y) + x(4)*cosh(y);
   end
   %figure
   %plot(xp,Fi(i,:))
   Fil(i) = betar(i)*(x(1)+x(3));
end

Itotal = Ih + ro*(L^3/3) + mp*L^2;

% Cálculo das matrizes

A = [ 0 1 0 0 0 0 ; 0 0 0 0 0 0 ; 0 0 0 1 0 0 ; 0 0 -w(1)^2 0 0 0 ; 0 0 0 0 0 1 ; 0 0 0 0
-w(2)^2 0 ];
B = [ 0 ; 1/Itotal ; 0 ; Fil(1) ; 0 ; Fil(2) ];

[ u v ] = size(Fi);
C = [ 1 0 0 0 0 0 ; 0 0 -Fi(1,u) 0 -Fi(2,u) 0 ];
D(2,1) = 0;

% pólos de malha fechada
%pc = [ -5 -7 -8+42i -8-42i -20+110i -20-110i];
po = [ -20.3 -21.3 -25+42i -25-42i -30+110i -30-110i];

% Cálculo das matrizes de ganhos
%K = place(A,B,pc);   % controlador
G = place(A',C',po)'; % observador

% Geração do modelo no simulink e sequente simulação para malha fechada com alocação de
pólos
% Saida: 1a. coluna: Teta, angulo do referencial flutuante
% 2a. coluna: Y relativo, ou seja, em relação ao referencial
Ident = [ -1 0 ; 0 -1 ];
new_system('braco')
open_system('braco')
open_system('mganhoI')
open_system('mganhoC')
open_system('mganhoG')
open_system('mganhoA')
open_system('mganhoB')
open_system('mganhoK')
add_block('built-in/Step', 'braco/entrada1') % Teta
add_block('built-in/Step', 'braco/entrada2') % Deslocamento
add_block('built-in/Sum',  'braco/Soma1')
add_block('built-in/Sum',  'braco/Soma2')
add_block('built-in/Sum',  'braco/Soma3')
add_block('built-in/ToWorkspace', 'braco/saida')
add_block('built-in/ToWorkspace', 'braco/saida2')
add_block('built-in/StateSpace', 'braco/planta')
add_block('mganhoI/ganho', 'braco/ganhoI')
add_block('mganhoC/ganho', 'braco/ganhoC')
add_block('mganhoG/ganho', 'braco/ganhoG')
add_block('mganhoA/ganho', 'braco/ganhoA')
add_block('mganhoB/ganho', 'braco/ganhoB')
add_block('mganhoK/ganho', 'braco/ganhoK')
add_block('built-in/Integrator', 'braco/integrador')
add_block('built-in/mux','braco/mux')
add_block('built-in/Saturation','braco/saturacao')
```

```
close_system('mganhoI')
close_system('mganhoC')
close_system('mganhoG')
close_system('mganhoA')
close_system('mganhoB')
close_system('mganhoK')

entrada = 15*pi/180;
des = 0;
ts = num2str(0);
bs = num2str(0);
as = num2str(entrada);
sas = num2str(0);
t2s = num2str(0);
b2s = num2str(0);
a2s = num2str(des);
Buffers = 'inf';
cs = num2str(2);
As = mat2str(A);
Bs = mat2str(B);
Cs = mat2str(C);
Ds = mat2str(D);
lss = num2str(-10);  % limite inferior da saturacao
uss = num2str(10);   % limite superior

set_param( 'braco/saturacao', 'UpperLimit', uss , 'LowerLimit', lss )
set_param( 'braco/entrada1', 'Time', ts )
set_param( 'braco/entrada1', 'Before', bs )
set_param( 'braco/entrada1', 'After', as )
set_param( 'braco/entrada1', 'SampleTime', sas )
set_param( 'braco/entrada2', 'Time', t2s )
set_param( 'braco/entrada2', 'Before', b2s )
set_param( 'braco/entrada2', 'After', a2s )
set_param( 'braco/entrada2', 'SampleTime', sas )
set_param( 'braco/saida', 'VariableName', 'saida' )
set_param( 'braco/saida2', 'VariableName', 'saida2' )
set_param( 'braco/planta', 'A', As, 'B', Bs, 'C', Cs, 'D', Ds )
set_param( 'braco/saida', 'Buffer', Buffers )
set_param( 'braco/Soma1', 'Inputs', ['+' '-'] )
set_param( 'braco/Soma2', 'Inputs', ['+' '-'] )
set_param( 'braco/Soma3', 'Inputs', ['+' '+' '+'] )
set_param( 'braco/mux', 'Inputs', cs)
set_param( 'braco/saida', 'Buffer', Buffers )
set_param( 'braco/saida2', 'Buffer', Buffers )

add_line( 'braco' ,'entrada1/1', 'mux/1')
add_line( 'braco' ,'entrada2/1', 'mux/2')
add_line( 'braco' ,'mux/1', 'Soma1/1')
add_line( 'braco' ,'Soma1/1', 'ganhoI/1')
add_line( 'braco' ,'ganhoI/1', 'Soma2/1')
add_line( 'braco' ,'Soma2/1', 'ganhoG/1')
add_line( 'braco' ,'ganhoG/1', 'Soma3/1')
add_line( 'braco' ,'Soma3/1', 'integrador/1')
add_line( 'braco' ,'integrador/1', 'ganhoK/1')
add_line( 'braco' ,'ganhoK/1', 'saturacao/1')
add_line( 'braco' ,'saturacao/1', 'planta/1')
add_line( 'braco' ,'ganhoK/1', 'ganhoB/1')
add_line( 'braco' ,'ganhoB/1', 'Soma3/2')
add_line( 'braco' ,'integrador/1', 'ganhoA/1')
add_line( 'braco' ,'ganhoA/1', 'Soma3/3')
add_line( 'braco' ,'integrador/1', 'ganhoC/1')
add_line( 'braco' ,'ganhoC/1', 'Soma2/2')
add_line( 'braco' ,'planta/1', 'Soma1/2')
add_line( 'braco' ,'planta/1', 'saida/1')
add_line( 'braco' ,'ganhoK/1', 'saida2/1')

save_system('braco')
```

3) MASSAP.M

```
% Acha os termos da planta
function massap = massap(betar)
global mp L b ha vol ro massa Ih I Il E
clear termo;
alfar = betar^4*Ih/ro;
gamar = betar^4*mp/ro;
t = betar*L;
cb = cos(t);
sb = sin(t);
shb = sinh(t);
chb = cosh(t);
b3 = betar^3;
b4 = betar^4;

termo(1) = 2*alfar*gamar*sb*chb;
termo(2) = -2*b4*sb*chb;
termo(3) = -4*betar*gamar*sb*shb;
termo(4) = -2*alfar*b3;
termo(5) = -2*alfar*b3*cb*chb;
termo(6) = -2*alfar*gamar*cb*shb;
termo(7) = 2*b4*cb*shb;

aux2 = 0;
for aux = 1: 7
   aux2 = aux2 + termo(aux);
end

massap = aux2;
```

4) PO.M

```
% Função para geração da população inicial
function fu = Po(mi,n)
for i = 1: mi
   for j = 1:n
      Pop(i,j,2) = (rand(1)-rand(1));%^2;  % Valor do parâmetro estratégico sigma
      Pop(i,:,1) = [7.5534    2.6800  -65.2853   -0.1523  -82.6199   -0.2741]; % Valores
obtidos pelo Gildin
end
fu = Pop;
```

5)R.M

```
% Operador de recombinação, usando a sugestão de Schwefel - ver pág 19
%¨essa função gera lambda individuos que ficam armazenados na matriz Pop
function fu = r(populacao,lambda)
[ linha coluna prof ] = size(populacao);
for i = 1: lambda
   for j = 1: coluna
      pai1 = pai(linha);
      pai2 = pai(linha);
      a = rand(1);
      if a <= 0.5
         escolhido = pai1;
      else
         escolhido = pai2;
      end
      Pop(i,j,1) = populacao(escolhido,j,1);
      Pop(i,j,2) = 0.5*(populacao(pai1,j,2) + populacao(pai2,j,2)); % pag 27
   end
```

```
end
fu = Pop;
```

6) PAI.M

```
% Função para escolha de pais na operação de recombinação
function fu = pai(numero)
a = rand(1);
aux = 1;
for cont1 = 1: numero
   if a < aux/numero
      p = aux;
   else
      aux = aux + 1;
   end
end
fu = p;
```

7) M.M

```
% Operador de mutação - pág21
function fu = m(populacao,delta)
[ linha coluna prof ] = size(populacao);
for i = 1:linha
   for j = 1:coluna
      Pop(i,j,2) = populacao(i,j,2)*exp(delta*randn(1));
      Pop(i,j,1) = populacao(i,j,1) + Pop(i,j,2)*randn(1);
   end
end
fu = Pop;
```

8)AGRUPA.M

```
% Agrupa Pt e Ptlinha como a nova população para estratégia mi+lambda
function fu = agrupa(Pt,Ptlinha)
[ a b c ] = size(Pt);
[ d e f ] = size(Ptlinha);
Aux = Pt;
Aux(a+1:a+d,:,:) = Ptlinha(1:d,:,:);
fu = Aux;
```

9) F_ALOC.M

```
% Função objetiva
function fu = f_aloc(populacao,individuo)
global A B C D G K Ident moptions aux adaant tout saida saida2
restricao = restricoes(populacao,individuo);
if restricao == 1
   fu = inf;
else
   for o = 1: 6
      K(1,o) = populacao(individuo,o,1);
   end
   sim('braco', 8, moptions) % Devolve as variáveis tout, saida e saida2
   Mp = abs(sobressinal(tout,saida(:,1)));
   ts = tempo_s(tout,saida(:,1));
   dm = abs(Mp-0.1);
   dt = abs(ts - 1);
   final = valor_final(saida(:,1));
   df = abs(final - 15);
   des = soma_des(saida(:,2),tout); %deslocamento médio no tempo
   pico = des_pico(saida(:,2));
```

```
    fu = 2*dm + dt + 10*pico + 200*df + 100*des;
end
```

10)RESTRICOES.M

```
% Função para o cálculo das restrições
function fu = restricoes(populacao,individuo)
[ linha coluna prof ] = size(populacao);
restricao = 0;
for j = 1: coluna
   if populacao(individuo,j,1) > 100
      restricao = 1;
   end
end
if populacao(individuo,1,1) < 0
   restricao = 1;
end
if populacao(individuo,2,1) < 0
   restricao = 1;
end
if populacao(individuo,3,1) > 0
   restricao = 1;
end
if populacao(individuo,4,1) > 0
   restricao = 1;
end
if populacao(individuo,5,1) > 0
   restricao = 1;
end
if populacao(individuo,6,1) > 0
   restricao = 1;
end
fu = restricao;
```

11)SOBRESSINAL.M

```
% Função que calcula o sobressinal para a saida obtida
function fu = sobressinal(tout,y)
tamanho = size(tout);
aux = tamanho(1,1);
cinf = y(aux);
aux2 = acha_maior(y);
cpico = y(aux2);
fu = (cpico - cinf)/cinf;
```

12)TEMPO_S.M

```
% Calcula o tempo de acomodacao para o sistema
function tempo_s = tempo_s(tout,y)
tamanho = size(tout);
aux = tamanho(1,1);
cinf = y(aux);  % valor da saida em regime
i = aux;
while (abs(y(i)-cinf)/cinf <= 0.05) & (i > 1)
   i = i - 1;
end
tempo_s = tout(i);
```

13)VALOR_FINAL.M

```
% Calcula desvio do valor final para os 15 graus
function fu = valor_final(vetor)
[a b ] = size(vetor);
```

```
fu = 180*vetor(a)/pi;
```

14)SOMA_DES.M

```
% Calculo a soma dos valores do deslocamento absoluto em relação ao referencial móvel
function fu = soma_des(vetor,tout)
[ a b ] = size(vetor);
total = 0;
for i = 1: a - 1
   dt = tout(i+1) - tout(i);
   if vetor(i)*vetor(i+1) >= 0
      total = total + abs(vetor(i)*dt + (vetor(i+1)-vetor(i))*dt/2);
   else
      h1 = abs(vetor(i));
      h2 = abs(vetor(i+1));
      total = total + (h1/2 + h2^2/(2*h1))*dt/(1+h2/h1);
   end

end
fu = total/(tout(a) - tout(1));
```

15)DES_PICO.M

```
% Calculo o valor de pico do deslocamento absoluto em relação ao referencial móvel
function fu = des_pico(vetor)
[ a b ] = size(vetor);
maior = 0;
for i = 1: a
   if abs(vetor(i)) > maior
      maior = abs(vetor(i));
   end
end
fu = maior;
```

16)ACHA_MENOR.M

```
% Função que acha o ,menor valor dentro de um vetor e retorna sua posição
function acha_menor = acha_menor(vetor)
aux = size(vetor);
tamanho = aux(1,2);
menor = 1;
for cont = 1: tamanho
   if vetor(cont) < vetor(menor)
      menor = cont;
   end
end
acha_menor = menor;
```

17)T_ALOC.M

```
% Critério de parada
function fu = t_aloc(populacao,tempo)
global aux adaant tout saida saida2
para = 0;
[ linha coluna prof ] = size(populacao);
for i = 1: linha
   adaptacao(i) = f_aloc(populacao,i);
end
menor = acha_menor(adaptacao);
if adaptacao(menor) < 0.001
   para = 1;
end
if tempo == 40
```

```
   para = 1;
end
%if adaptacao(menor) < 1000
%   if abs(adaptacao(menor)-adaant) < 0.0000001
%      para = 1;
%   end
%end
adaant = adaptacao(menor);
fu = para;
```

18)S_ALOC.M

```
% Operador de seleção, problema de minimização
function fu = s_aloc(populacao,mi)
global aux
[ linha coluna prof ] = size(populacao);
for i = 1: linha
   adaptacao(i) = f_aloc(populacao,i);
end
auxiliar = populacao;
for i = 1: mi
   posmenor = acha_menor(adaptacao);
   Pop(i,:,:) = auxiliar(posmenor,:,:);
   adaptacao(posmenor) = inf;
end
fu = Pop;
```

II) Códigos para alocação de pólos com chutes iniciais estocásticos.

Para esta implementação, utilizaram-se os mesmos arquivos que foram utilizados com chute inicial dado. O único arquivo que sofreu modificação foi o seguinte:

1) PO.M

```
% Função para geração da população inicial
function fu = Po(mi,n)
for i = 1: mi
   for j = 1:n
      Pop(i,j,2) = (rand(1)-rand(1));%^2;  % Valor do parâmetro estratégico sigma
   end
   Pop(i,1,1) = abs(100*(rand(1)-rand(1)));  % Valor da variável objetiva
   Pop(i,2,1) = abs(100*(rand(1)-rand(1)));  % Valor da variável objetiva
   Pop(i,3,1) = -abs(100*(rand(1)-rand(1)));  % Valor da variável objetiva
   Pop(i,4,1) = -abs(100*(rand(1)-rand(1)));  % Valor da variável objetiva
   Pop(i,5,1) = -abs(100*(rand(1)-rand(1)));  % Valor da variável objetiva
   Pop(i,6,1) = -abs(100*(rand(1)-rand(1)));  % Valor da variável objetiva
   end
fu = Pop;
```

III) Códigos para otimização simples dos ganhos dos PIDs com função objetiva original.

1)PID_BRACO.M

```
% Programa principal - PID
```

```
clear
global A B C D tout saida moptions tempo Valortodos Ktodos picotodos Mptodos tstodos
finaltodos mediatodos iteracao modelo
acha_matrizes_pid
%open_system('braco_pid') %Linha utilizada caso o modelo ja exista. Na verdade nao
precisa abrir para simular

% Parâmetros para o algorítimo
mi = 2;         % mi indivíduos por geração
lambda = 10;    % lambda > mi, lambda filhos gerados
n = 6;          % seis variáveis a serem otimizadas
delta = 1.8;      % Parâmetro para a mutação - passo para a variação de sigma
moptions = simset('DstWorkspace','base');
tempomax = 40;
fim = 0;
modelo = 'sub_braco';
interrompe = 1;

fim = 0;
clear Pt
clear Ptlinha
%clear K
% Processo de optimização
Pt = Po(mi,n) % Geração da população inicial - Chute inicial aleatório
Ptlinha = m(r(Pt,lambda),delta);  % Geração dos lambda descendentes
tempo = 1
Ptlinha = agrupa(Pt,Ptlinha);
while  fim == 0
      Pt = s_pid2(Ptlinha,mi)
      Ptlinha = m(r(Pt,lambda),delta);
      tempo = tempo + 1
      Ptlinha = agrupa(Pt,Ptlinha);
      fim = t_pid2(tempo,tempomax);
end
for i = 1: lambda + mi
   adaptacao(i) = f_pid_final(Ptlinha,i);
end
melhor = acha_menor(adaptacao);
if adaptacao(melhor) > 1000  % para a simulacao caso a melhor delas de mais que 200
   break
end
K = Ptlinha(melhor,:,1);
Ktodos(iteracao,tempo,:) = Ptlinha(melhor,:,1);
Valortodos(iteracao,tempo) = adaptacao(melhor);
moptions = simset('SrcWorkspace','current');
sim(modelo, 8, moptions) % Devolve as variáveis tout e y
Mptodos(iteracao,tempo) = abs(sobressinal(tout,saida(:,1)));
tstodos(iteracao,tempo) = tempo_s(tout,saida(:,1));
finaltodos(iteracao,tempo) = valor_final(saida(:,1));
mediatodos(iteracao,tempo) = soma_des(saida(:,2),tout); %deslocamento médio no tempo
picotodos(iteracao,tempo) = des_pico(saida(:,2));
Valortodos(iteracao,tempo)

grafico_pid
```

2)ACHA_MATRIZES_PID.M

```
% Procedimento para determinação do modelo do braço
% 11.11.02
global mp L b ha vol ro massa Ih I I1 E
% --------------- D A D O S -------------------
mp = 0.1;    % Massa na ponta da barra. Único parâmetro alterável
L = 0.7;
b = 0.0254;
ha = 0.001;
vol = L*b*ha;
massa = 2710*vol;
```

```
ro = massa/L;
Ih = 1.29e-4;
I = (b*ha^3)/12;  % Momento de inércia da viga
I1 = (ha*b^3)/12;  % outro eixo
E = 6.3e10; % Módulo de elasticidade

% --- D E T E R M I N A Ç Ã O  D O S  P A R Â M E T R O S ---------
% Procedimento para determinação das matrizes do modelo no espaço de estados

chutes = [ 5.3 8.7 11.8 16.0 20.3];
for i = 1: 2
   is = num2str(i);
   betar(i) = fzero('massap',chutes(i));
   alfar(i) = betar(i)^4*Ih/ro;
   gamar(i) = betar(i)^4*mp/ro;
   w(i) = sqrt(betar(i)^4*E*I/ro);
   aux = betar(i)*L;
   cb = cos(aux);
   sb = sin(aux);
   shb = sinh(aux);
   chb = cosh(aux);
   b3 = betar(i)^3;
   A = [ 0 1 0 1 ; alfar(i) -betar(i) alfar(i) betar(i) ; -sb -cb shb chb ; -
b3*cb+gamar(i)*sb b3*sb+gamar(i)*cb b3*chb+gamar(i)*shb b3*shb+gamar(i)*chb ];
   x(4) = 1;
   B = [ -1 ; -betar(i); -chb; -b3*shb-gamar(i)*chb];
   x(1:3) = A(1:3,1:3)\B(1:3);
   dL = L/100;
   xp = 0:dL:L;
   % Modos de vibração
   for j = 1 : length(xp)
      y = betar(i)*xp(j);
      Fi(i,j) = x(1)*sin(y) + x(2)*cos(y) + x(3)*sinh(y) + x(4)*cosh(y);
   end
   Fil(i) = betar(i)*(x(1)+x(3));
   % Normalização
   aux1 = 0;
   for j = 1: length(xp)-1
      aux1 = aux1 + ro*dL*((Fi(i,j)+Fi(i,j+1))/2)^2;

   end
   delta = aux1 + Ih*(Fil(i)^2) + mp*(Fi(i,1)^2);
   x = x/sqrt(delta);
   for j = 1 : length(xp)
      y = betar(i)*xp(j);
      Fi(i,j) = x(1)*sin(y) + x(2)*cos(y) + x(3)*sinh(y) + x(4)*cosh(y);
   end
   %figure
   %plot(xp,Fi(i,:))
   Fil(i) = betar(i)*(x(1)+x(3));
end

Itotal = Ih + ro*(L^3/3) + mp*L^2;

% Cálculo das matrizes

A = [ 0 1 0 0 0 0 ; 0 0 0 0 0 0 ; 0 0 0 1 0 0 ; 0 0 -w(1)^2 0 0 0 ; 0 0 0 0 0 1 ; 0 0 0 0
-w(2)^2 0 ];
B = [ 0 ; 1/Itotal ; 0 ; Fil(1) ; 0 ; Fil(2) ];

[ u v ] = size(Fi);
C = [ 1 0 0 0 0 0 ; 0 0 -Fi(1,u) 0 -Fi(2,u) 0 ];
D(2,1) = 0;
```

3)PO.M

```
% Função para geração da população inicial
function fu = Po(mi,n)
for i = 1: mi
   for j = 1:n
      aux = rand(1);
      if aux <= 0.5
         Pop(i,j,1) = 1*rand(1);
         Pop(i,j,2) = 0.1*randn(1);
      else
         if aux <= 0.9
            Pop(i,j,1) = 10*rand(1);
            Pop(i,j,2) = 1*randn(1);
         else
            Pop(i,j,1) = 100*rand(1);
            Pop(i,j,2) = 10*randn(1);
         end
      end
   end
end
fu = Pop;
```

4)R.M

Igual ao utilizado por alocação de pólos.

5)PAI.M

Igual ao utilizado por alocação de pólos.

6)M.M

Igual ao utilizado por alocação de pólos.

7)AGRUPA.M

Igual ao utilizado por alocação de pólos.

8)S_PID2.M

```
% Operador de seleção, problema de minimização
function fu = s_pid2(populacao,mi)
global A B C D tempo Ktodos Valortodos Mptodos tstodos finaltodos mediatodos picotodos
iteracao modelo
interrompe = 1;
[ linha coluna prof ] = size(populacao);
for i = 1: linha
   adaptacao(i) = f_pid_final(populacao,i);
end
adaptacao
auxiliar = populacao;
melhor = acha_menor(adaptacao);
%adaptacao
if adaptacao(melhor) > 300000 % para a simulacao caso a melhor delas de mais que 3000
    break
```

```
end
K = populacao(melhor,:,1);
Ktodos(iteracao,tempo,:) = populacao(melhor,:,1);
Valortodos(iteracao,tempo) = adaptacao(melhor);
moptions = simset('SrcWorkspace','current');
sim(modelo, 8, moptions) % Devolve as variáveis tout e y
Mptodos(iteracao,tempo) = abs(sobressinal(tout,saida(:,1)));
tstodos(iteracao,tempo) = tempo_s(tout,saida(:,1));
finaltodos(iteracao,tempo) = valor_final(saida(:,1));
mediatodos(iteracao,tempo) = soma_des(saida(:,2),tout); %deslocamento médio no tempo
picotodos(iteracao,tempo) = des_pico(saida(:,2));
Valortodos(iteracao,tempo)
for i = 1: mi
   posmenor = acha_menor(adaptacao);
   Pop(i,:,:) = auxiliar(posmenor,:,:);
   adaptacao(posmenor) = inf;
end
fu = Pop;
```

9)F_PID_FINAL.M

```
% Função objetiva
function fu = f_pid_final(populacao,individuo)
global A B C D modelo %moptions %tout saida
moptions = simset('SrcWorkspace','Current');
interrompe = 1;
for o = 1: 6
   K(o) = populacao(individuo,o,1);
end
restricao = restricoes(populacao,individuo);
if restricao == 1
   fu = inf;
else
   sim(modelo,8,moptions)
   Mp = abs(sobressinal(tout,saida(:,1)));
   ts = tempo_s(tout,saida(:,1));
   dt = abs(ts - 4);
   final = valor_final(saida(:,1));
   dm = abs(Mp-0.01);
   df = abs(final - 15);
   des = soma_des(saida(:,2),tout); %deslocamento médio no tempo
   pico = des_pico(saida(:,2));
   fu = 2*dm + dt + 10*pico + 200*df + 100*des;
   mtorque = acha_maior(torque);
   if abs(torque(mtorque)) > 1.5
      fu = fu*10;

   end
end
```

10)ACHA_MENOR.M

Igual ao utilizado por alocação de pólos.

11)SOBRESSINAL.M

Igual ao utilizado por alocação de pólos.

12)TEMPO_S.M

Igual ao utilizado por alocação de pólos.

13)VALOR_FINAL.M

Igual ao utilizado por alocação de pólos.

14)SOMA_DES.M

Igual ao utilizado por alocação de pólos.

15)DES_PICO.M

Igual ao utilizado por alocação de pólos.

16)T_PID2.M

```
% Critério de parada
function fu = t_pid2(tempo,tempomax)
para = 0;
if tempo == tempomax
   para = 1;
end
fu = para;
```

17)GRAFICO_PID.M

```
% Script para graficos da situacao inicial fixa
global modelo
modelo = 'braco_pid'
total = 5;

for a = 5: 5%total

 numero = 40;
 maior = acha_menor(Valortodos(a,:))
 Mp = Mptodos(a,maior)
 ts = tstodos(a,maior)
 media = mediatodos(a,maior)
 final = finaltodos(a,maior)
 pico = picotodos(a,maior)
 Iteracoes = linspace(1,numero,numero);
 clear esperadomp
 clear esperadots
 clear esperadofinal
 for i = 1: numero
    esperadomp(i) = 0.1;
    esperadots(i) = 1;
    esperadofinal(i) = 15;
 end
 figure
 plot(Iteracoes,Valortodos(a,:),Iteracoes,Valortodos(a,:),'*')
 %title(['Iteracao: ' num2str(a)] )
 xlabel('Iterações')
 ylabel('Valor da função de avaiação')
 % axis([1 numero 0.3 0.75])
 grid

 figure
 subplot(2,2,1)
 plot(Iteracoes,picotodos(a,:),Iteracoes,picotodos(a,:),'*')%
 %title(['Iteracao: ' num2str(a)] )
```

```
 xlabel('Iterações')
 ylabel('Deslocamento da ponta do braço (m)')
 %axis([1 40 0.01 0.05 ])
 %axis([1 10 0.024 0.034 ])

 subplot(2,2,2)
 plot(Iteracoes,finaltodos(a,:),Iteracoes,finaltodos(a,:),'*',Iteracoes,esperadofinal,'-
')
 %title(['Iteracao: ' num2str(a)] )
 xlabel('Iterações')
 ylabel('Valor Final de Teta (graus)')
 %axis([1 40 14.5 15.5])
 axis([1 30 14.99 15.01])

 subplot(2,2,3)
 plot(Iteracoes,tstodos(a,:),Iteracoes,tstodos(a,:),'*',Iteracoes,esperadots,'-')
 %title(['Iteracao: ' num2str(a)] )
 xlabel('Iterações')
 ylabel('Tempo de Acomodação (s)')
 %axis([5 40 0.8 1.25])
 %axis([1 40 0.995 1.015])

 subplot(2,2,4)
 plot(Iteracoes,Mptodos(a,:),Iteracoes,Mptodos(a,:),'*',Iteracoes,esperadomp,'-')
 %title(['Iteracao: ' num2str(a)] )
 xlabel('Iterações')
 ylabel('Sobressinal')
 %axis([1 40 0.05 0.5])
 %axis([1 10 0.0 0.3])

 %Simulação para melhor resultado
 a
 maior
 for i = 1: 6
    K(i) = Ktodos(a,maior,i);
 end
 K
 moptions = simset('SrcWorkspace','current');
 sim(modelo, 8, moptions)
 figure
 plot(tout,saida(:,2))
 xlabel('Tempo (s)')
 ylabel('Deslocamento na ponta do braço (m)')
 grid
 %title({'Iteracao: ' num2str(a)] )
 figure
 plot(tout,180*saida(:,1)/pi)
 xlabel('Tempo (s)')
 ylabel('Teta (graus)')
 grid
 %title(['Iteracao: ' num2str(a)] )
 figure
 plot(tout,torque)
 xlabel('Tempo (s)')
 ylabel('Torque (Nm)')
 grid
 %title(['Iteracao: ' num2str(a)] )

end

valor_final = verifica_pid2(K)
```

IV) Código para otimização do PID com outra função objetiva.

O único arquivo que sofre alteração com relação à implementação anterior é o seguinte:

```
% Função objetiva
function fu = f_pid_final(populacao,individuo)
global A B C D modelo %moptions %tout saida
moptions = simset('SrcWorkspace','Current');
interrompe = 1;
for o = 1: 6
   K(o) = populacao(individuo,o,1);
end
restricao = restricoes(populacao,individuo);
if restricao == 1
   fu = inf;
else
   sim(modelo,8,moptions)
   Mp = abs(sobressinal(tout,saida(:,1)));
   ts = tempo_s(tout,saida(:,1));
   dt = abs(ts - 4);
   final = valor_final(saida(:,1));
   dm = abs(Mp-0.001);
   df = abs(final - 15);
   des = soma_des(saida(:,2),tout); %deslocamento médio no tempo
   pico = des_pico(saida(:,2));
   fu = 20*dm + dt + 100*pico + 2000*df + 100*des;
   mtorque = acha_maior(torque);
   if abs(torque(mtorque)) > 1.5
      fu = fu*10;

   end
end
```

V) Códigos para otimização das funções de busca da otimização para os ganhos dos PIDs.

Os arquivos utilizados são os mesmos utilizados na implementação anterior, ocorrendo modificações no seguinte arquivo:

1)PID_BRACO.M

```
% Programa principal - PID
clear
global A B C D tout saida moptions tempo Valortodos Ktodos picotodos Mptodos tstodos
finaltodos mediatodos iteracao modelo
acha_matrizes_pid
%open_system('braco_pid') %Linha utilizada caso o modelo ja exista. Na verdade nao
precisa abrir para simular

% Parâmetros para o algorítimo
mi = 5;         % mi indivíduos por geração
lambda = 15;    % lambda > mi, lambda filhos gerados
n = 6;          % seis variáveis a serem otimizadas
delta = 1.8;      % Parâmetro para a mutação - passo para a variação de sigma
moptions = simset('DstWorkspace','base');
tempomax = 40;
fim = 0;
```

```
modelo = 'sub_braco';
interrompe = 1;

for iteracao = 1: 5

   fim = 0;
   clear Pt
   clear Ptlinha
   %clear K
   % Processo de optimização
   %Pt = Po(mi,n) % Geração da população inicial - Chute inicial aleatório
   Pt = Pofixo(mi,n)
   %Ptlinha = Pt;
   Ptlinha = m(r(Pt,lambda),delta);  % Geração dos lambda descendentes
   tempo = 1
   Ptlinha = agrupa(Pt,Ptlinha);
   while  fim == 0
      %clear Pt;
      Pt = s_pid2(Ptlinha,mi)
      % Pt = s_pidteste(Ptlinha,mi);
      Ptlinha = m(r(Pt,lambda),delta);
      tempo = tempo + 1
      Ptlinha = agrupa(Pt,Ptlinha);
      fim = t_pid2(tempo,tempomax);
   end
  for i = 1: lambda + mi
   adaptacao(i) = f_pid_final(Ptlinha,i);
  end
  melhor = acha_menor(adaptacao);
  if adaptacao(melhor) > 1000  % para a simulacao caso a melhor delas de mais que 200
       break
  end
  K = Ptlinha(melhor,:,1);
  Ktodos(iteracao,tempo,:) = Ptlinha(melhor,:,1);
  Valortodos(iteracao,tempo) = adaptacao(melhor);
  moptions = simset('SrcWorkspace','current');
  sim(modelo, 8, moptions) % Devolve as variáveis tout e y
  Mptodos(iteracao,tempo) = abs(sobressinal(tout,saida(:,1)));
  tstodos(iteracao,tempo) = tempo_s(tout,saida(:,1));
  finaltodos(iteracao,tempo) = valor_final(saida(:,1));
  mediatodos(iteracao,tempo) = soma_des(saida(:,2),tout); %deslocamento médio no tempo
  picotodos(iteracao,tempo) = des_pico(saida(:,2));
  Valortodos(iteracao,tempo)
end

grafico_pid
```

VI) Códigos para otimização das funções de busca do PID

Alguns códigos utilizados foram os mesmos utilizados na otimização simples dos ganhos do PID. Os diferentes são:

1)GERAL_PESOS.M

```
clear
% Algotitmo para obtencao de valores para a funcao objetiva
global A B C D moptions Valortodosp Pesostodos Ktodos modelo Ptlinha Kauxiliar%Valortodos
Ktodos Mptodos tstodos finaltodos mediatodos picotodos
moptions = simset('SrcWorkspace','current');
acha_matrizes_pid % Calcula os valores de A, B, C e D
modelo = 'sub_braco';
```

```
mip = 4;
lambdap = 6;
np = 7;
deltap = 2;
tempomaxp = 10;
fimp = 0;
Pesostodos(1,np) = 0;
Valortodosp(tempomaxp) = 0;
Ktodos(1,6) = 0;

Ptp = Pop(mip,np);
Ptlinhap = m_p(r(Ptp,lambdap),deltap);
tempop = 1
Ptlinhap = agrupa(Ptp,Ptlinhap);  % mi + lambda ES
fimp = t_pesos(tempop,tempomaxp);
for i = 1: lambdap + mip
   adaptacaop(i) = f_pesos(Ptlinhap,i);
end
melhorp = acha_menor(adaptacaop);
Ktodos(tempop,:) = Kauxiliar(melhorp,:)
Pesostodos(tempop,:) = Ptlinhap(melhorp,:,1)
Valortodosp(tempop) = adaptacaop(melhorp)
while fimp == 0
   clear Kauxiliar
   tempop = tempop + 1
   Ptp = s_pesos(Ptlinhap,mip,tempop);
   Ptlinhap = m_p(r(Ptp,lambdap),deltap);
   Ptlinhap = agrupa(Ptp,Ptlinhap);
   fimp = t_pesos(tempop,tempomaxp);
end
```

2)POP.M

```
% Função para geração da população inicial de pesos
function fu = Pop(mi,n)

for j = 1:n
   Pop(1,j,2) = (rand(1)-rand(1))^2;  % Valor do parâmetro estratégico sigma
end
Pop(1,:,1) = [ 2 1 10 200 100 0.1 1 ];
for i = 2: mi
   for j = 1:n
      Pop(i,j,1) = rand(1);  % Valor da variável objetiva geracao aleatoria
      Pop(i,j,2) = (rand(1)-rand(1))^2;  % Valor do parâmetro estratégico sigma
   end
end
fu = Pop;
```

3)M_P.M

```
% Operador de mutação otimizacao dos pesos
function fu = m(populacao,delta)
[ linha coluna prof ] = size(populacao);
for i = 1:linha
   for j = 1:coluna
      Pop(i,j,2) = populacao(i,j,2)*exp(delta*randn(1));
      Pop(i,j,1) = abs(populacao(i,j,1) + Pop(i,j,2)*randn(1));
   end
end
fu = Pop;
```

4)T_PESOS.M

```
% Critério de parada para processo de pesos
```

```
function fu = t_pesos(tempo,tempomax)
para = 0;
if tempo == tempomax
   para = 1;
end
fu = para;
```

5)F_PESOS.M

```
% Função objetiva para analise dos pesos da funcao objetiva
function fu = f_pesos(populacao,individuo)
global A B C D moptions modelo Ktodos Kauxiliar% Valortodos Ktodos Mptodos tstodos
finaltodos mediatodos picotodos
restricaop = restricoesp(populacao,individuo);
if restricaop == 1
   fu = inf;
else
   for o = 1: 7
      Pesos(o) = populacao(individuo,o,1);
   end
   mi = 2;
   lambda = 10;
   n = 6;
   delta = 1.8;
   tempomax = 20;
   fim = 0;

   Pt = Po(mi,n);
   Ptlinha = m(r(Pt,lambda),delta);
   tempo = 1;
   Ptlinha = agrupa(Pt,Ptlinha);  % mi + lambda ES
   fim = t_pid2(tempo,tempomax);
   while fim == 0
      Pt = s_pidp(Ptlinha,mi,Pesos);
      Ptlinha = m(r(Pt,lambda),delta);
      Ptlinha = agrupa(Pt,Ptlinha);
      tempo = tempo + 1;
      fim = t_pid2(tempo,tempomax);
   end
   % Quando termina pega os resultados com relacao à função objetiva original
   for i = 1: mi + lambda
      adaptacao(i) = f_pidv(Ptlinha,i);
   end
   melhor = acha_menor(adaptacao);
   Kauxiliar(individuo,:) = Ptlinha(melhor,:,1);
   fu = adaptacao(melhor);
end
```

6)S_PIDP.M

```
% Operador de seleção, problema de minimização
function fu = s_pidp(populacao,mi,Pesos)
global A B C D moptions modelo
[ linha coluna prof ] = size(populacao);
for i = 1: linha
   adaptacao(i) = f_pidp(populacao,i,Pesos);
end
auxiliar = populacao;
for i = 1: mi
   posmenor = acha_menor(adaptacao);
   Pop(i,:,:) = auxiliar(posmenor,:,:);
   adaptacao(posmenor) = inf;
end
fu = Pop;
```

7)F_PIDP.M

```
% Função objetiva
function fu = f_pidp(populacao,individuo,Pesos)
global A B C D moptions modelo
interrompe = 1;
for o = 1: 6
   K(o) = populacao(individuo,o,1);
end
restricao = restricoes(populacao,individuo);
if restricao == 1
   fu = inf;
else
   sim(modelo,4,moptions)
   Mp = abs(sobressinal(tout,saida(:,1)));
   ts = tempo_s(tout,saida(:,1));
   dm = abs(Mp-Pesos(6));%0.1);
   dt = abs(ts - Pesos(7));%1);
   final = valor_final(saida(:,1));
   df = abs(final - 15);
   des = soma_des(saida(:,2),tout); %deslocamento médio no tempo
   pico = des_pico(saida(:,2));
   fu = Pesos(1)*dm + Pesos(2)*dt + Pesos(3)*pico + Pesos(4)*df + Pesos(5)*des;
end
```

8)F_PIDV.M

```
% Função objetiva
function fu = f_pidv(populacao,individuo)
global A B C D modelo %moptions %tout saida
moptions = simset('SrcWorkspace','Current');
interrompe = 1;
for o = 1: 6
   K(o) = populacao(individuo,o,1);
end
restricao = restricoes(populacao,individuo);
if restricao == 1
   fu = inf;
else
   sim(modelo,4,moptions)
   Mp = abs(sobressinal(tout,saida(:,1)));
   ts = tempo_s(tout,saida(:,1));
   dm = abs(Mp-0.1);
   dt = abs(ts - 1);
   final = valor_final(saida(:,1));
   df = abs(final - 15);
   des = soma_des(saida(:,2),tout); %deslocamento médio no tempo
   pico = des_pico(saida(:,2));
   %fu = 2*dm + 5*dt + 10*pico + 1500*df + 1000*des;
   fu = 2*dm + dt + 10*pico + 200*df + 100*des;
   mtorque = acha_maior(torque);
   if abs(torque(mtorque)) > 1.5
      fu = fu*10;
   end
end
```

9)S_PESOS.M

```
% Operador de seleção, problema de minimização
function fu = s_pesos(populacao,mi,tempop)
global A B C D moptions Pesostodos Valortodosp Kauxiliar Ktodos
[ linha coluna prof ] = size(populacao);
for i = 1: linha
   adaptacaop(i) = f_pesos(populacao,i);
end
melhorp = acha_menor(adaptacaop)
Ktodos(tempop,:) = Kauxiliar(melhorp,:)
```

```
Pesostodos(tempop,:) = populacao(melhorp,:,1)
Valortodosp(tempop) = adaptacaop(melhorp)
auxiliar = populacao;
for i = 1: mi
   posmenor = acha_menor(adaptacaop);
   Pop(i,:,:) = auxiliar(posmenor,:,:);
   adaptacaop(posmenor) = inf;
end
fu = Pop;
```