CODIFICAÇÃO POR PÓLOS E ZEROS PARA O PROJETO DE FILTROS DIGITAIS DE RESPOSTA INFINITA AO IMPULSO UTILIZANDO OTIMIZAÇÃO POR ENXAME DE PARTÍCULAS

Daniel Rossato de Oliveira*, Heitor Silvério Lopes*

* *Universidade Tecnológica Federal do Paraná (UTFPR)*
Av. Sete de Setembro 3165
Curitiba, Paraná, Brasil

Emails: rossato.daniel@gmail.com, heitorslopes@gmail.com

Abstract— IIR digital filter design is classic problem in Electrical Engineering. There are many specific methods in literature, but most of these methods are not flexible, preventing one from fixing all parameters. Therefore, this problem is also solved in literature with Evolutionary Computing techniques. Several works were found with this approach, but in almost all, the focus was in the evolutionary algorithm rather than on the filter problem, using this only as a benchmark function. Moreover, the filter stability - prerequisite for the filter application - is neglected in almost all of them. So, this work proposes a new codification, which maps a search sub-space made only by stable solutions. This approach is compared with the most common codification in the literature, and experiments showed a much better performance obtained with the proposed approach.

Keywords— Evolutionary Computing, Digital Filters, PSO.

Resumo— O projeto de filtros digitais do tipo IIR é um problema clássico da engenharia. Existem vários métodos especialistas consagrados na literatura, mas a maior parte destes métodos não é flexível, impedindo a especificação de todos os parâmetros importantes de um filtro. Por isso, este problema também é atacado na literatura através de técnicas de computação evolucionária. Diversos estudos foram encontrados com esta abordagem, mas em quase todos, o enfoque era dado no algoritmo evolucionário em vez de no filtro, utilizando este apenas como uma função de *benchmark*. Além disso, a estabilidade do filtro - condição imprescindível para a utilização do mesmo - é desprezada em quase todos os trabalhos. Este trabalho propõe uma nova codificação, que mapeia um sub-espaço de busca composto apenas pelas soluções estáveis. Esta abordagem é comparada com a codificação mais utilizada na literatura, e os resultados mostram um desempenho muito superior obtido pela abordagem proposta.

Palavras-chave— Computação Evolucionária, Filtros Digitais, PSO.

1 Introdução

O processamento de sinais é uma área da Engenharia Elétrica que permeia praticamente todas as outras áreas. Sistemas embarcados processam sinais de dispositivos de entrada, sistemas de controle processam os sinais de sensores e atuadores, etc. Dentro da área de processamento de sinais, a parte que lida com sinais digitais é uma das mais presentes em nosso dia-a-dia, já que a maior parte da análise e manipulação de dados é feita hoje por dispositivos digitais.

Os filtros digitais são uma das bases do processamento digital de sinais. Os filtros são classificados conforme diversas características, como o tipo de sua resposta, seu comportamento quando submetido a diferentes sinais, sua linearidade, etc. Uma das principais características é a duração da resposta ao impulso unitário de um filtro. Esta pode ser finita ou infinita, gerando filtros chamados respectivamente de FIR (*Finite Impulse Response*) ou IIR (*Infinite Impulse Response*).

A principal diferença na implementação destes filtros é que os do tipo IIR possuem realimentação, isto é, os resultados em cada instante na saída afetam o valor de saída nos instantes seguintes. O filtro FIR não possui realimentação. Esta característica tem várias consequências, sendo as principais: dificuldade de obter resposta em fase linear, tendência à instabilidade, e maior fator de *roll-off*. Este fator é a "velocidade" de transição entre as bandas de passagem e rejeição de um filtro. Como um filtro IIR consegue um *roll-off* muito mais acentuado que um FIR que utilize o mesmo processamento, os filtros IIR são mais utilizados.

Existem diversos métodos de projeto de filtros digitais, cada um com suas particularidades. Porém, a maioria não é muito flexível: não é possível especificar todos os parâmetros desejados, pois alguns serão resultado do cálculo. Eles também priorizam o atendimento de uma das faixas em detrimento da outra, gerando um desequilíbrio. Se o projetista fornecer um conjunto de especificações impossível de ser alcançado, os métodos tradicionais fornecerão um filtro que obedeça aos critérios em uma das faixas, enquanto a outra fica muito longe do exigido.

Por isso, é muito estudado na literatura o projeto de filtros digitais através de técnicas de otimização heurística, dentre elas a Computação Evolucionária (CE). Os métodos de CE são algoritmos bioinspirados, isto é, baseados em algum processo biológico ou natural que formam meta-heurísticas para a resolução de problemas de otimização numérica ou combinatória. Dentre as técnicas mais conhecidas, estão o Algo-

ritmo Genético (AG) (Goldberg, 1989), Otimização por Enxame de Partículas (PSO) (Kennedy and Eberhart, 1995), Programação Genética (GP) (Koza, 1992), entre outras. Neste campo estão sempre sendo propostas novas técnicas como o *Harmony Search* (HS) (Geem et al., 2001), Otimização por Enxame Bioluminescente (BSO) (de Oliveira, Parpinelli and Lopes, 2011),*Bat Algorithm* (BA) (Yang, 2010), Evolução Diferencial (DE) (Price et al., 2005), Otimização por Colônia de Formigas (ACO) (Dorigo and Stützle, 2004), dentre várias outras.

Quando se busca otimizar um problema com métodos de CE, uma das escolhas mais importantes é a codificação do problema, que é a forma de representação numérica de uma solução candidata. Usualmente, para o projeto de filtros digitais, cada solução é representada pelos coeficientes do filtro (Pan, 2011; Chen and Chen, 2006; Gao et al., 2008). Porém, esta abordagem gera alguns problemas que podem ser resolvidos com uma codificação melhor.

O objetivo deste trabalho é apresentar uma nova codificação para o projeto de filtros digitais com métodos de Computação Evolucionária, e comparar o desempenho obtido com a codificação mais utilizada na literatura.

Na Seção 2 é feita uma breve revisão teórica sobre os filtros digitais IIR, o algoritmo PSO, e os trabalhos da literatura que utilizam técnicas de CE para o projeto de filtros digitais. Na Seção 3, é apresentada a nova abordagem proposta, suas implicações e vantagens esperadas, assim como os experimentos realizados para avaliar seu desempenho. Por fim, a Seção 4 mostra os resultados obtidos, assim como as conclusões tiradas a partir destes.

2 Fundamentação Teórica

2.1 Filtros Digitais

Filtros digitais são sistemas que efetuam uma série de operações matemáticas em um sinal digital discreto, de modo a amplificar ou atenuar determinadas características deste sinal (Diniz et al., 2004). Como são geralmente sistemas lineares invariantes no deslocamento (LSI), diversas propriedades observadas no estudo destes sistemas são aplicáveis neste caso. Entre as mais importantes estão a total representação do sistema através de sua resposta ao impulso, e o critério para a avaliação da estabilidade.

Um sistema LSI pode ser representado por uma equação de diferenças (Hamming, 1997), no formato mostrado na Equação 1.

$$\sum_{i=0}^{N} a_i y(n-i) - \sum_{l=0}^{M} b_i x(n-l) = 0 \qquad (1)$$

Em geral, deseja-se calcular a saída atual do sistema, chamada de $y(n)$. Isolando-se $y(n)$ na Equação 1, obtém-se a Equação 2 (considerando-se $a_0 = 1$).

$$y(n) = \frac{\sum_{l=0}^{M} b_i x(n-l)}{\sum_{i=1}^{N} a_i y(n-i)} \qquad (2)$$

Como essa equação descreve totalmente o sistema (Hamming, 1997) e tanto os valores x quanto y são de amostras alheias ao sistema em si, tem-se que o sistema também pode ser totalmente caracterizado pelos vetores de coeficientes a_k e b_k, que multiplicam respectivamente as amostras de saída e de entrada.

Estes vetores e equações descrevem o funcionamento do filtro no domínio do tempo discreto. Este formato é importante para a implementação do filtro, mas para o projeto e análise do mesmo, é muito mais usual a utilização da representação da resposta do filtro no domínio da frequência. É nele que são especificados os ganhos e faixas do filtro. Para se calcular esta resposta em frequência, utiliza-se a Equação 3 (Oppenheim and Schafer, 1989).

$$H(e^{j\omega}) = \frac{\sum_{k=0}^{M-1} b(k) e^{-j\omega k}}{\sum_{l=0}^{N-1} a(l) e^{-j\omega l}} \qquad (3)$$

Portanto, é a partir dos vetores a_k e b_k que se calcula a resposta em frequência de um filtro, e partir desta resposta em frequência que se avalia a adequação dele ao projeto em questão.

2.2 Otimização por Enxame de Partículas

A Otimização por Enxame de Partículas (PSO) foi proposta por Kennedy and Eberhart (1995), e é um algoritmo inspirado no movimento sincronizado de cardumes de peixes e bandos de pássaros. Neste algoritmo, cada solução candidata é representada por uma partícula, cuja posição em um espaço N-dimensional é a representação numérica desta solução. Cada partícula possui também velocidade própria e inércia, isto é, a força da influência da velocidade atual na mudança de velocidade na próxima iteração.

A navegação do grupo de partículas, também chamado de enxame, é guiada pela melhor solução encontrada pelo enxame como um todo, chamado de máximo social ou *gbest*, pela melhor solução já encontrada pela própria partícula, chamado de máximo cognitivo ou *lbest*, e pela velocidade anterior da partícula. O peso de cada uma destas soluções na alteração de velocidade e posição das partículas é dado pelas constantes c_g e c_l, que regem respectivamente a influência de *gbest* e *lbest*. A inércia é dada por w, e o tamanho do enxame ou número de partículas é chamado de M. Algumas implementações utilizam também o *pbest*, que é o

melhor resultado encontrado pelo grupo do qual a partícula faz parte.

As partículas são inicializadas com velocidade e posições aleatórias, no início da execução. A Figura 1 descreve os passos tomados em cada iteração, buscando-se otimizar uma função $f(\vec{x})$.

```
1  Configura parâmetros: M, c_g, c_l, w;
2  Inicializa de forma aleatória o vetor de
   posição das partículas x_i;
3  Inicializa de forma aleatória o vetor de
   velocidade das partículas v_i;
4  Inicializa o vetor lbest_i = x_i;
5  Encontra o ótimo global gbest;
6  while Critério de parada não for satisfeito
   do
7      for cada partícula i do
8          //atualiza velocidade;
9          v_i ← w · v_i + c_g · rand · (gbest −
           x_i) + c_l · rand · (lbest − x_i);
10         //atualiza posição;
11         x_i ← x_i + v_i;
12         if f(x_i) > f(lbest_i) then
13             //atualiza lbest;
14             lbest_i ← x_i;
15         end if
16         if f(x_i) > f(gbest) then
17             //atualiza gbest;
18             gbest ← x_i;
19         end if
20     end for
21 end while
22 Pós-processamento e visualização de dados;
```

Figura 1: Otimização por Enxame de Partículas (PSO)

A equação mostrada na linha 9 do Algoritmo 1 diz respeito à velocidade da partícula, que é o núcleo da heurística do PSO. Seu efeito é levar a partícula a ser atraída ao mesmo tempo para o *gbest* e para o *lbest*, balanceando a relação entre busca global e busca local, respectivamente, pelas constantes c_g e c_l.

2.3 Trabalhos Relacionados

No artigo de Chen and Chen (2006), é proposta uma modificação no algoritmo PSO para a evolução de filtros digitais IIR. Esta modificação busca calibrar a busca do algoritmo alterando a ênfase entre local e global. O objetivo é fazer com que partículas com boas soluções efetuem uma busca mais local, enquanto partículas com baixo *fitness*, que provavelmente estão em vizinhanças de baixa qualidade, efetuem sua busca de forma global, para explorar melhor o espaço. Para avaliar a qualidade destas modificações, foi analisado o desempenho dos algoritmos em problemas de *benchmarking*, e na identificação de filtros IIR. Os resultados encontrados são melhores do que os de outras variações do PSO inclusive o canônico, mas não é feita uma comparação com métodos clássicos de cálculo de filtro.

Já em Gao et al. (2008), a modificação feita é chamada de CPSO, ou *Chaotic* PSO. Nela, é inserido um fator caótico em vez de aleatório no cálculo da velocidade das partículas. Além disto, são geradas várias populações iniciais em vez de apenas uma, e a melhor delas é selecionada para ser trabalhada pelo algoritmo. O resultado do CPSO é comparado apenas com o PSO canônico e com a resposta ideal de cada filtro. Ou seja, como não é mostrado o desempenho de um método como Chebyshev ou Butterworth em comparação com o obtido pelo CPSO, não é possível saber se há alguma vantagem nesta proposta, já que o custo computacional é obviamente maior. Porém, o CPSO mostrou desempenho melhor do que o PSO canônico.

A introdução de fatores caóticos é novamente utilizada em Jia et al. (2009). Aqui, é utilizada a chamada mutação caótica, em que a melhor partícula de cada iteração sofre mutação de forma caótica, para melhorar a busca local do algoritmo. O resultado é comparado com o do PSO canônico, assim como com o dos métodos de Butterworth, Chebyshev tipo I e Chebyshev tipo II. Embora o CPSO tenha sido superado apenas pelo Chebyshev tipo II, apenas um filtro é utilizado para este teste. Sendo assim, é difícil dizer se o método realmente é satisfatório.

Outra variação do PSO utilizada para o cálculo de filtros IIR é o PSO-QI, proposta em Luite and Venayagamoorthy (2008). Nela, as partículas do PSO são tratadas como partículas da física quântica, em que velocidade e posição não podem ser determinadas com precisão ao mesmo tempo. Desta forma, as partículas são representadas por uma função de densidade de probabilidade, que resulta na probabilidade de uma partícula estar em determinada posição num determinado instante. Esta abordagem mostrou resultados melhores que o PSO, mas não foi comparada com métodos fortes de cálculos de filtros.

Um estudo mais profundo da evolução de filtros é feita em Pan (2011). Aqui, a evolução de filtros IIR é testada utilizando PSO, AG, Evolução Diferencial (DE) e uma versão modificada do AG chamada de *Improved Genetic Algorithm* (IGA). O autor deduz um método de verificação de estabilidade que exige menos esforço computacional, e compara o desempenho de todos os algoritmos. Porém, apenas filtros de segunda ordem são utilizados nos experimentos, tornando os problemas simples de serem resolvidos por otimização numérica. Outro problema é a ausência de comparação com os métodos clássicos, que daria uma real

medida da qualidade dos resultados obtidos pelos algoritmos.

Em Yu et al. (2009), é proposto um PSO com esquema adaptativo para o momento de inércia. Nele, cada partícula possui um momento de inércia próprio, baseado em seu *fitness* e corrigido por uma função sigmóide. De forma similar à proposta de Chen and Chen (2006), o objetivo é dar ênfase à busca local nas proximidades das melhores soluções encontradas, e utilizar as partículas em vizinhanças ruins para explorar o espaço de busca de forma mais global. O algoritmo proposto é chamado de AIW-PSO, e apresentou resultados melhores do que o PSO canônico. Porém, o resultado não foi comparado com filtros obtidos pelos métodos clássicos.

O trabalho de Zhang et al. (2010) propõe um esquema de redistribuição de partículas em caso de convergência prematura. Esta redistribuição nada mais é do que a reinicialização das partículas, chamada em outros trabalhos de explosão. Neste artigo, o critério utilizado para a detecção de convergência é a distância média entre cada partícula e o melhor resultado encontrado. O resultado desta proposta, chamada de RPSO, é comparada com o PSO canônico, QPSO, CPSO e o algoritmo de Cultura Diferencial (DC). O RPSO alcançou melhores resultados, mas estes não foram comparados com métodos fortes.

Observou-se em todos estes trabalhos que usualmente o projeto de filtros digitais é utilizado apenas como um problema de *benchmark*, em vez de um problema de engenharia. Raramente os resultados são comparados com os obtidos com um método especialista, e a estabilidade do filtro - condição essencial para sua utilização - também costuma ser desprezada.

3 Metodologia

Ao aplicar-se métodos de Computação Evolucionária na resolução de problemas numéricos, duas escolhas possuem o maior impacto na eficácia do algoritmo: o cálculo de *fitness* e a codificação. O primeiro é a forma de mensurar a qualidade de uma dada solução, e a segunda é a forma de representação desta dada solução.

Em filtros digitais, o cálculo de *fitness* mais usual é através do erro quadrático médio (MSE) da resposta em frequência do filtro em relação à resposta ideal. Há outras formas de avaliar a qualidade de um filtro (de Oliveira, Scalabrin, Teodoro, Parpinelli and Lopes, 2011; de Oliveira, 2011), mas a discussão destas não está no escopo deste trabalho. Portanto, para que os resultados das modificações propostas nestes trabalhos possam ser comparados com resultados encontrados na literatura, o cálculo de *fitness* será feito pela MSE, como descrito na Equação 4.

$$f(\vec{x}) = \sum_{i=1}^{N} \frac{|Y(i) - X(i)|^2}{N} \tag{4}$$

onde $f(\vec{x})$ é o *fitness* da solução $\vec{x}$, $Y(i)$ é a resposta do filtro ideal, $X(i)$ é a resposta em frequência de $\vec{x}$, e N é o número de amostras a ser considerado nas respostas em frequência.

A codificação utilizada em todos os trabalhos encontrados na literatura foi a codificação por coeficientes. Ou seja, cada filtro é representado pelos vetores a_k e b_k utilizados na implementação do filtro, como mostrado na Equação 2. A vantagem desta codificação é que o cálculo da resposta em frequência vem diretamente deste vetores. Portanto, são necessários poucos cálculos para comparar a resposta do filtro proposto com a resposta do filtro ideal.

Porém, esta abordagem apresenta alguns problemas. O primeiro é a faixa de valores e a precisão de cada coeficiente. Os valores dos coeficientes podem ser extremamente díspares, com alguns sendo 100 vezes maiores que outros. É possível normalizar os coeficientes, mas isso pode implicar em uma necessidade elevada de precisão.

Em algoritmos discretos como o AG tradicional, esse aumento de precisão acarreta num grande aumento do espaço de busca, elevando a complexidade do problema, o esforço computacional e o tempo necessário para concluir o projeto do filtro. Além disso, este aumento do espaço de busca pode levar o algoritmo a ficar "preso" em mínimos locais mais facilmente.

Além da dificuldade de se estabelecer os limites e precisão do espaço de busca, há um problema muito maior, que é a estabilidade do filtro. Um filtro IIR é estável quando todas as raízes do polinômio formado pelo vetor a_k tem módulo menor do que 1, ou seja, estão dentro do círculo de raio unitário no plano z. Portanto, para descobrir se um filtro é estável ou não – e portanto, utilizável ou não – é necessário calcular as raízes deste polinômio.

A estabilidade do filtro é um critério comumente chamado em CE de restrição (Goldberg, 1989). A princípio, a função de *fitness* não leva em conta o atendimento ou não à restrição. Para isso, em geral são utilizadas penalidades, que são redutores de *fitness*. Porém, esta solução não é muito boa: nada garante que ao final do processo evolutivo haverá um filtro estável, e a forma de aplicação destas penalidades necessita de estudos. Seria possível zerar o *fitness* das soluções inválidas, mas esta abordagem costuma atrapalhar a evolução. O método que costuma apresentar melhores resultados é a de punir proporcionalmente à gravidade da desobediência à restrição. Por exemplo, poderia ser avaliado o número de pólos causadores de instabilidade, ou a distância deles ao círculo de raio unitário, entre outros. Ainda as-

sim, nada garantiria que um filtro estável seria atingido ao final do algoritmo.

3.1 Codificação por pólos e zeros

Melhor do que aplicar penalidades, é alterar a codificação de forma que somente soluções válidas sejam representáveis. Ou seja, o novo espaço de busca é um sub-espaço do original, contendo todas e apenas as soluções estáveis possíveis. Neste problema, isso é possível representando os filtros através de seus pólos e zeros, ao invés de seus coeficientes. Desta forma, limitando-se o módulo dos pólos e zeros em 1, garante-se que apenas filtros estáveis sejam representáveis, e todos eles estão contidos no espaço de busca. A Figura 2 mostra um exemplo de diagrama de pólos e zeros de um filtro.

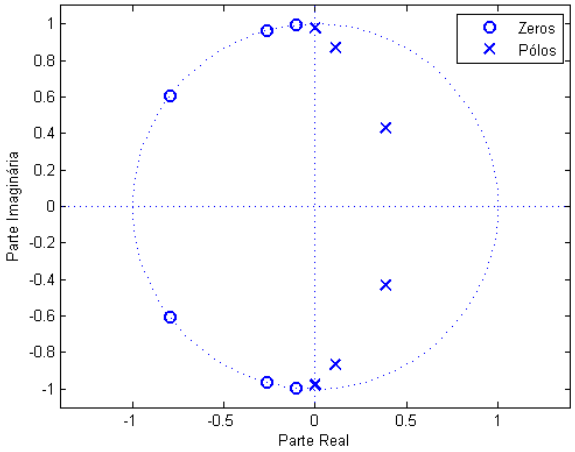


Figura 2: Diagrama de pólos e zeros

Em um primeiro momento, a solução mais óbvia seria representar cada pólo/zero por seu módulo e ângulo. Porém, para que um filtro seja realizável, todos os seus coeficientes deve ser números reais. Para que esta condição seja alcançada, todos os pólos e zeros do filtro que sejam complexos necessitam de outro pólo ou zero conjugado. Por exemplo, um filtro com um pólo em $0,3 + j0,4$ só seria realizável se também houvesse outro pólo em $0,3 - j0,4$.

Com isso, há um problema equivalente ao da codificação por coeficientes: apenas uma fração do espaço de busca é válido. Seria novamente necessário aplicar penalidades, o que anularia as vantagens geradas pela nova representação. A solução encontrada também é relacionada à codificação. Basta que cada pólo ou zero tenha seu conjugado implícito. Ou seja, se a codificação diz que o filtro possui um pólo em $0,4 + j0,6$, ele automaticamente possui outro pólo em $0,4 - j0,6$. Dessa forma, qualquer valor representável na codificação gerará um filtro realizável. A Figura 3 mostra como isto se organiza.

A primeira metade dos valores diz respeito aos pólos, e a segunda metade diz respeito aos zeros.

Figura 3: Codificação proposta

Cada par de valores *Mxn;Axn* equivale respectivamente ao módulo e ângulo de um pólo ou zero. Como citado anteriormente, cada pólo/zero tem seu conjugado implícito, então cada par de valores reprenta um par de pólos ou um par de zeros.

Os valores de *Mxn* são limitados entre 0 e 1, para garantir a estabilidade do filtro. Os ângulos poderiam variar entre $-\pi$ e π, mas isto não é necessário pelo seguinte motivo: qualquer valor de ângulo negativo teria o mesmo efeito de um ângulo positivo. Isto ocorre pelo conjugado implícito, já que o ângulo do conjugado de um número complexo é o ângulo original com o sinal invertido. Logo, representar cada valor Axn entre 0 e π já é suficiente. Para fins de padronização, este valor será normalizado entre 0 e 1. Desta forma, todo o vetor é composto por valores normalizados entre 0 e 1.

Como não é possível calcular a resposta em frequência do filtro diretamente através dos pólos e zeros, é necessário primeiro converter estes valores nos coeficientes da função de transferência. Esta conversão é feita através da Equação 5.

$$H(z) = \Pi_{i=1}^{N} \frac{(z - a_i)}{(z - p_i)} \tag{5}$$

onde a_k são os zeros e p_k são os pólos. A partir do vetor codificação, é possivel demonstrar que a conversão fica como na Equação 6.

$$H(z) = \prod_{i=1}^{\frac{N}{2}} \frac{z^2 + z \cdot MZ_i \cdot 2 \cdot cos(\pi \cdot AZ_i) + MZ_i^2}{z^2 + z \cdot MP_i \cdot 2 \cdot cos(\pi \cdot AP_i) + MP_i^2} \tag{6}$$

onde cada par módulo/ângulo gera um par de pólos ou zeros, ou seja, uma seção de segunda ordem do filtro. O produto destas seções é o filtro completo.

Com os polinômios da função de transferência (ou seja, os vetores de coeficientes a_k e b_k) calculados, já é possível descobrir a curva de resposta em frequência, e com isso calcular o *fitness*. Porém, mais um passo é necessário. Quando se especifica os ganhos de um filtro, o que realmente importa é a diferença entre os ganhos na banda de passagem e na banda de rejeição, pois é trivial modificar ambos: basta multiplicar a função de transferência pelo ganho desejado. Portanto, os filtros calculados terão seu ganho normalizado antes do cálculo do *fitness*. O ganho geral do filtro

sera dividido pelo ganho máximo, resultando em um ganho máximo de $1V/V$ ou 0dB.

4 Experimentos e Resultados

Para avaliar o desempenho da codificação proposta, foram utilizados 3 especificações diferentes de filtros, retirados da literatura (Diniz et al., 2004). A especificação de cada um está mostrada na Tabela 1. Esta tabela mostra os 6 conjuntos de experimentos realizados, que são os 3 filtros com as 2 diferentes codificações.

Foi utilizado o PSO canônico com os parâmetros padrão ($c_g = 2,0$; $c_l = 2,0$; $\omega = 0,5$), com tamanho do enxame $M = 100$ particulas e número de iterações $N = 1000$. Cada configuração foi executada 30 vezes, para que o desempenho médio seja obtido. Isso é necessário pois o PSO é um algoritmo estocástico, e portanto cada execução fornece resultados diferentes. Além disso, a função de *fitness* foi normalizada e transformada em maximização, já que o PSO canônico é maximizador. A transformação utilizada é mostrada na Equação 7.

$$fitness(\vec{x}) = \frac{1}{1 + MSE(\vec{x})} \tag{7}$$

onde $\vec{x}$ é a solução proposta, e MSE é o erro quadrático médio da resposta em frequência em relação à resposta ideal do filtro especificado.

Para a codificação por coeficientes, é necessário aplicar penalidades para a instabilidade. A função destas penalidades é guiar o algoritmo para soluções estáveis, já que as instáveis terão *fitness* mais baixo. A função de penalidade utilizada é mostrada na Equação 8.

$$p_1 = \frac{1}{1 + N_p} \tag{8}$$

onde $p1$ é o fator a multiplicar o *fitness* da solução, e N_p é o número de pólos causadores de instabilidade, isto é, com módulo maior do que 1.

4.1 Resultados

Os resultados obtidos pelos experimentos são mostrados na Tabela 2. A coluna referente ao *fitness* máximo mostra o melhor resultado encontrado em todas as 30 rodadas, *fitness* mínimo é o pior resultado encontrado ao final das rodadas, e *fitness* médio é o resultado médio encontrado nos testes, mostrado juntamente com o desvio padrão. A coluna ρ mostra a probabilidade de que as médias sejam as mesmas pelo teste de t-Student. Ou seja, expressa a chance de que a diferença observada na média do *fitness* obtido seja puramente aleatória e não causada pela codificação diferente. A coluna "Diferença" mostra o intervalo em que se encontra a diferença no *fitness* médio obtido pela mudança de codificação, com 95% de certeza.

As Figuras 4, 5 e 6 mostram as curvas de *fitness* obtidas pelas duas abordagens. A Figura 4 mostra a evolução média da melhor partícula nos experimentos 1 e 2. É possível perceber que quando representadas por coeficientes, as soluções tendem a convergir mais rápido, mas a qualidade dos resultados obtidos é bem mais baixa se houver tempo suficiente para evoluírem as soluções representadas por pólos e zeros. O ponto de cruzamento dos *fitness* é próximo da iteração 200.

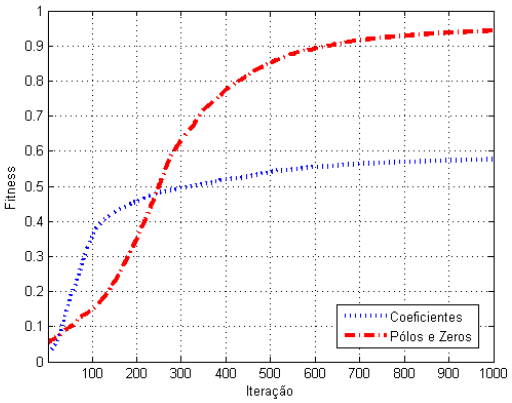


Figura 4: Média da evolução do fitness máximo para os experimentos 1 e 2

Na Figura 5 percebe-se que o algoritmo com a representação por coeficientes convergiu muito rapidamente, mas para soluções de qualidade muito baixa. Já com os pólos e zeros, o algoritmo sequer chegou a convergir dentro do limite estipulado de iterações, pois nota-se uma inclinação na curva de *fitness* que sugere um potencial ainda considerável de melhora da solução. Ainda assim, o contraste na qualidade entre as soluções é visível.

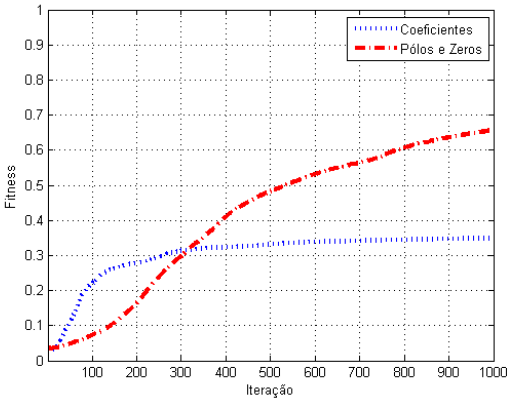


Figura 5: Média da evolução do fitness máximo para os experimentos 3 e 4

Nos experimentos 5 e 6, ambas as codificações parecem ainda ter um certo potencial de melhora ao final das iterações, mas com pólos e zeros o desempenho é muito melhor. Este é um filtro mais difícil do que os experimentos anteriores, pois re-

Exp. N^o	Filtro N^o	Codificação	Ordem	ω_p	ω_r	A_p [dB]	A_r[dB]
1	1	Coeficientes	6	0,45	0,5	1	-30
2	1	Pólos e Zeros	6	0,45	0,5	1	-30
3	2	Coeficientes	6	0,8	0,85	1	-30
4	2	Pólos e Zeros	6	0,8	0,85	1	-30
5	3	Coeficientes	4	0,08	0,1	1	-30
6	3	Pólos e Zeros	4	0,08	0,1	1	-30

Tabela 1: Experimentos de codificação

Exp. N^o	*Fitness* máximo	*Fitness* mínimo	*Fitness* médio	ρ	Diferença
1	0,7904	0,3114	0,5769 ± 0,1449	8,45e-14	(0,3101; 0,4237)
2	0,9713	0,8434	0,9438 ± 0,0299		
3	0,5588	0,1564	0,2182 ± 0,1066	1,36e-9	(0,3759; 0,6065)
4	0,8850	0,1631	0,7094 ± 0,2843		
5	0,7371	0,0689	0,3494 ± 0,1586	3,89e-8	(0,2222; 0,3923)
6	0,8841	0,2010	0,7775 ± 0,2062		

Tabela 2: Resultados dos experimentos de codificação

quer um *roll-off* muito acentuado. Isto explica a necessidade de um tempo maior para a convergência, mas a Figura 6 deixa implícito que a codificação por coeficientes não chegará perto de atingir o resultado obtido com pólos e zeros, mesmo com um grande número de iterações.

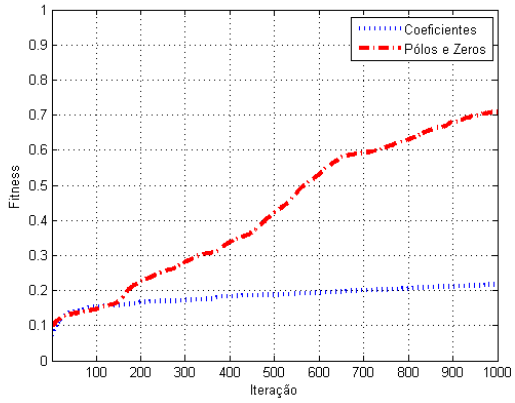


Figura 6: Média da evolução do fitness máximo para os experimentos 5 e 6

A Figura 7 mostra os filtros médios obtidos pelas duas abordagens. O filtro médio foi definido como sendo o filtro obtido cujo *fitness* mais se aproximou do *fitness* médio obtido em todas as rodadas. Desta forma tem-se um filtro existente cujo desempenho reflete o desempenho do algoritmo como um todo.

5 Conclusões

A abordagem proposta mostrou resultados muito melhores do que a codificação tradicional. Trabalhos na literatura forneceram resultados aparentemente melhores em um primeiro momento, mas praticamente nenhum leva em consideração a estabilidade do filtro. Com a nova codificação, é impossível obter um filtro instável.

Mesmo sem a realização de um ajuste dos parâmetros do PSO, os filtros encontrados tiveram respostas satisfatórias. Trabalhos futuros investigarão o desempenho desta nova codificação em comparação com métodos especialistas, assim como melhores conjuntos de parâmetros do PSO.

Outros algoritmos evolucionários também devem ser avaliados, assim como os respectivos parâmetros de configuração que forneçam os melhores resultados.

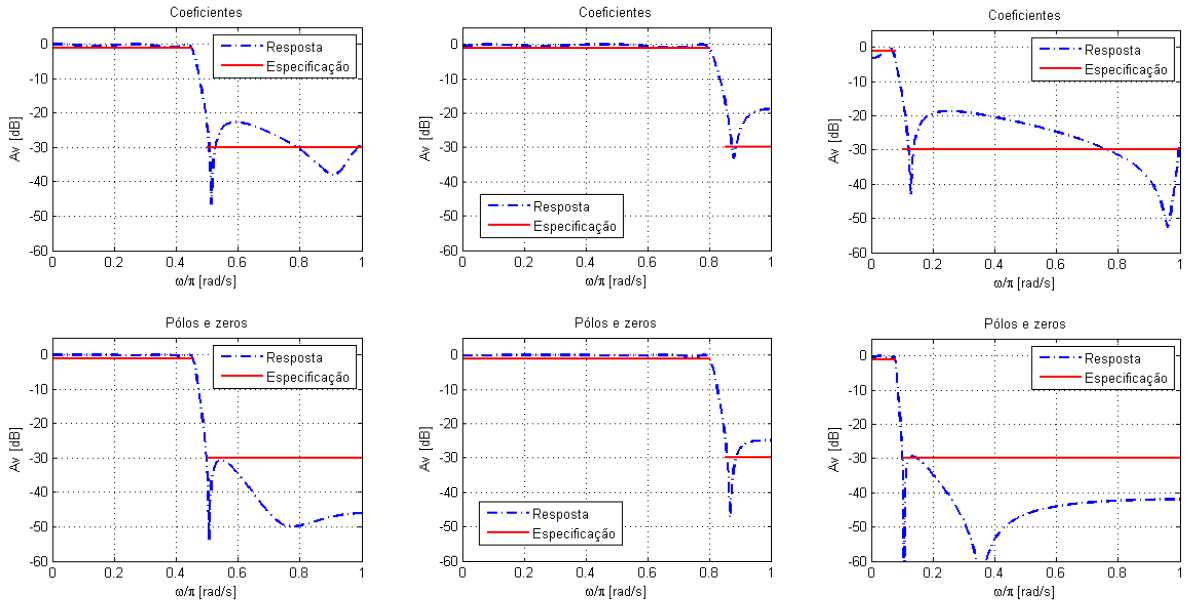
Referências

Chen, H.-C. and Chen, O. T.-C. (2006). Particle swarm optimization incorporating a preferential velocity-updating mechanism and its applications in IIR filter design, *IEEE International Conference on Systems, Man, and Cybernetics* **1**: 1190–1195.

de Oliveira, D. R. (2011). *Projeto de filtros digitais iir com técnicas de computação evolucionária*, Master's thesis, Universidade Tecnológica Federal do Paraná.

de Oliveira, D. R., Parpinelli, R. S. and Lopes, H. S. (2011). Bioluminescent swarm optimization algorithm, *in* E. Kita (ed.), *Evolutionary Algorithms*, InTech, pp. 69–84.

de Oliveira, D. R., Scalabrin, M. H., Teodoro, F. R., Parpinelli, R. S. and Lopes, H. S. (2011). Comparação de técnicas de computação evolucionária para o projetos de filtros digitais de resposta finita ao impulso, *in* H. S. Lopes and R. Takahashi (eds), *Computação Evolucionária em Problemas de Engenharia*, Omnipax, pp. 107–128.



(a) Filtro 1 (b) Filtro 2 (c) Filtro 3

Figura 7: Filtros Obtidos

Diniz, P., da Silva, E. and Netto, S. (2004). *Processamento Digital de Sinais – Projeto e Análise de Sistemas*, Bookman.

Dorigo, M. and Stützle, T. (2004). *Ant Colony Optimization*, MIT Press.

Gao, Y., Li, Y. and Qian, H. (2008). The design of IIR digital filter based on chaos particle swarm optimization algorithm, *Second International Conference on Genetic and Evolutionary Computing* **1**: 303–306.

Geem, Z., Kim, J. and Loganathan, G. (2001). A new heuristic optimization algorithm: Harmony search, *Simulation* **76**(2): 60–68.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA.

Hamming, R. (1997). *Digital Filters*, 3^{rd} edn, Dover Publications, Mineola, NY.

Jia, D., Jiao, Y. and Zhang, J. (2009). Satisfactory design of IIR digital filter based on chaotic mutation particle swarm optimization, *Third International Conference on Genetic and Evolutionary Computing* **1**: 48–51.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization, *Proc. of IEEE International Conference on Neural Networks* **4**: 1942–1948.

Koza, J. (1992). *Genetic Programming: on the programming of computers by means of natural selection*, Cambridge: MIT Press.

Luite, B. and Venayagamoorthy, G. K. (2008). Particle swarm optimization with quantum infusion for the design, *IEEE Swarm Intelligence Symposium* **1**: 1–8.

Oppenheim, A. and Schafer, R. (1989). *Discrete-Time Signal Processing*, Prentice-Hall.

Pan, S.-T. (2011). Evolutionary computation on programmable robust IIR filter pole-placement design, *IEEE Transactions on Instrumentation and Measurement* **60**: 1469–1479.

Price, K., Storn, R. and Lampinen, J. (2005). *Differential Evolution: A Practical Approach to Global Optimization*, Springer.

Yang, X.-S. (2010). A new meta-heuristic bat-inspired algorithm, *Natured Inspired Cooperative Strategies for Optimization* **1**: 65–74.

Yu, X., Liu, J. and Li, H. (2009). An adaptive inertia weight particle swarm optimization algorithm for IIR digital filter, *International Conference on Artificial Intelligence and Computational Intelligence.* **1**: 114–118.

Zhang, X., Jia, P. and Guo, J. (2010). An improved particle swarm optimizer for IIR digital filter design, *International Conference on Intelligent Systems and Knowledge Engineering* **1**: 191–196.