IMPLEMENTAÇÃO DO PROBLEMA OTIMIZAÇÃO BASEADO EM *CONSTRAINTS* EM PYTHON PARA MODULAÇÃO *SYNCHRONOUS OPTIMAL PULSE-WIDTH*

TIARLES GUTERRES[1]

SUMÁRIO

1 Introdução 2
2 Desenvolvimento 2
3 Resultados e Discussão 5

LISTA DE FIGURAS

Figura 1 Captura da página online de documentação da função *fmincon*. Disponível em https://www.mathworks.com/help/optim/ug/fmincon.html. 3
Figura 2 Um exemplo de otimização utilizando a função *minimize* do pacote *scipy.optimize*. . 4
Figura 3 Código da otimização feita para o problema da *SO Modulation*. 6
Figura 4 Resultados da otimização em cada iteração para (a) $x0 = [10° \ 30° \ 50°]$ e (b) $x0 = [13.36° \ 26.73° \ 53°]$ com índice de modulação igual à 0.8. 7
Figura 5 Resultados da otimização em cada índice de modulação para (a) $x0 = [10° \ 30° \ 85°]$ e (b) $x0 = [13.36° \ 26.73° \ 53°]$ com índice de modulação variando entre 0.1 e 1.2. . . 7

[1] *Grupo de Eletrônica de Potência e Controle (GEPOC), UFSM, Santa Maria, Brasil*

1 INTRODUÇÃO

O método de modulação síncrona otimizada (*Synchronous Optimal Modulation*) é baseada em ângulos de referência pré-estabelecidos a partir de um processo de otimização numérica. A otimização utiliza uma função custo e que, neste trabalho, considera a simetria de um quarto de onda do sinal modulante ($f(t)$) que se dá através da expansão da função $f(t)$ na série de Fourier (equação (1)).

$$f(t) = a_0 + \sum_{h=1}^{\infty} a_h \cos\left(h\frac{2\pi}{T}t\right) + b_h \sin\left(h\frac{2\pi}{T}t\right) \tag{1}$$

DEDUÇÃO DA FUNÇÃO CUSTO. A função a ser analisada para a extração da função custo considera que a função é periódica ($f(t) = f(t+T)$) e possui simetria de meia onda (*HWS*) e de um quarto de onda (*QWS*), ou seja:

$$f(t) = -f\left(t \pm \frac{T}{2}\right) \quad \text{para } HWS \text{ e } QWS\text{, e ainda}$$
$$|f(t)| = \left|f\left(\frac{T}{2} - t\right)\right| \quad \text{para } QWS.$$

Para a $f(t)$ com *QWS* os coeficientes da série de Fourier que solucionam a equação (1) podem ser dados por:

$$a_0 = a_h = 0, \forall\, h \in R \text{ e}$$
$$b_h = \frac{4}{\pi}\int_0^{\frac{\pi}{2}} f(\omega t)\sin(h\omega t)d\omega t. \tag{2}$$

Considerando θ os ângulos a serem definidos para a modulação da referência a equação (2) pode ser reescrita como:

$$b_h = \frac{4E}{\pi h}\sum_{i=1}^{N}(-1)^{i+1}\cos(h\theta_i), \tag{3}$$

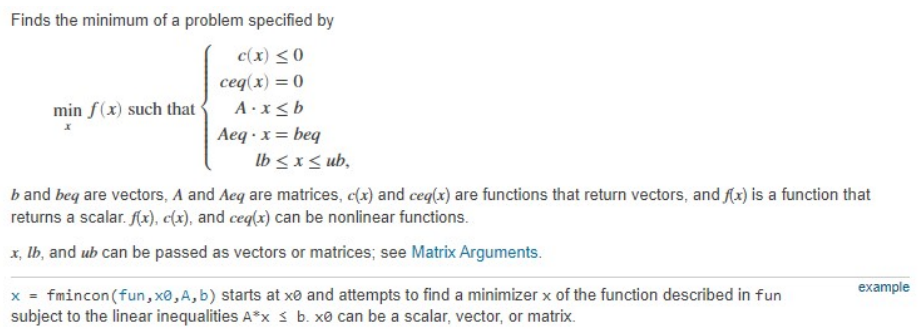
onde E é a tensão no barramento DC do conversor e N é o número de ângulos escolhidos.

2 DESENVOLVIMENTO

O desenvolvimento começa a partir da necessidade dos métodos que consigam rastrear e otimizar a função objetivo (*objective function*), como a mostrada na equação (3) e a respeitando as *constraints* do sistema definidas, considerando $N = 3$, por:

$$0 < \theta_1 < \theta_2 < \theta_3 < \frac{\pi}{2}.$$

OTIMIZAÇÃO VIA MINIMIZAÇÃO PERANTE *constraints*. O método fmincon($\cdot$) do software MATLAB é parametrizada como mostra a Figura 1. Podemos observar que a forma mais simples de utilizar o método é a passando para ele a função que se deseja otimizar (*fun*, com mostra a Figura), o chute inicial (*xo*) e uma relação matricial ($Ax \leqslant b$) que carrega a informação de *constraints*, que são algumas das regras ao qual a otimização deve obedecer no processo.



Finds the minimum of a problem specified by

$$\min_x f(x) \text{ such that } \begin{cases} c(x) \leq 0 \\ ceq(x) = 0 \\ A \cdot x \leq b \\ Aeq \cdot x = beq \\ lb \leq x \leq ub, \end{cases}$$

b and beq are vectors, A and Aeq are matrices, $c(x)$ and $ceq(x)$ are functions that return vectors, and $f(x)$ is a function that returns a scalar. $f(x)$, $c(x)$, and $ceq(x)$ can be nonlinear functions.

x, lb, and ub can be passed as vectors or matrices; see Matrix Arguments.

`x = fmincon(fun,x0,A,b)` starts at `x0` and attempts to find a minimizer `x` of the function described in `fun` subject to the linear inequalities `A*x ≤ b`. `x0` can be a scalar, vector, or matrix. example

Figura 1: Captura da página online de documentação da função *fmincon*. Disponível em https://www.mathworks.com/help/optim/ug/fmincon.html.

OTIMIZAÇÃO EM *python*. *Python* oferece algumas ferramentas de otimização semelhantes a função *fmincon*. O pacote *NLopt* oferece uma interface para trabalhar com uma biblioteca em C com alguns métodos lá desenvolvidos, porém ela não possui um padrão pythônico[1] para a instalação e desenvolvimento. Outra alternativa (e que será aplicada para esta atividade) é o uso da função *minimize* do pacote *optimize* da biblioteca *SciPy*. A função *minimize(·)* aceita *constraints* lineares e não-lineares, logo ela pode ser utilizada para o problema de otimização da modulação síncrona.

UM EXEMPLO DE OTIMIZAÇÃO UTILIZANDO *minimize*. A Figura 2 mostra um exemplo de otimização considerando o sistema:

$$\min_x f(x) \quad \text{subject to} \begin{cases} f(x) & = & x_1x_4(x_1+x_2+x_3)+x_3 & & \\ c_1(x) & = & x_1x_2x_3x_4 & \geqslant & 25 \\ c_2(x) & = & x_1^2+x_2^2+x_3^2+x_4^2 & = & 40 \\ 1 & \leqslant & x_1,x_2,x_3,x_4 & \leqslant & 5 \end{cases} \tag{4}$$

1 Padrão pythônico (ou em inglês *Pythonic way*) é como se chama a simplicidade de instalação e o uso de pacotes para a interação com a linguagem Python.

```
In [1]: import numpy as np
        from scipy.optimize import minimize

In [2]: def objective(x):
            x1, x2, x3, x4 = x
            return x1*x4*(x1+x2+x3)+x3

        def constraint1(x):
            x1, x2, x3, x4 = x
            return x1*x2*x3*x4-25.

        def constraint2(x):
            x1, x2, x3, x4 = x

            return x1**2 + x2**2 + x3**2 + x4**2 - 40

In [3]: x0 = [1,5,5,1]

        b = (1.0,5.0)
        bnds = (b,b,b,b)
        con1 = {'type': 'ineq', 'fun': constraint1}
        con2 = {'type': 'eq', 'fun': constraint2}
        cons = [con1, con2]

In [4]: sol = minimize(objective, x0, method='SLSQP', bounds=bnds, constraints=cons); sol

Out[4]:      fun: 17.01401724556073
             jac: array([14.57227039,  1.37940764,  2.37940764,  9.56415081])
         message: 'Optimization terminated successfully.'
            nfev: 30
             nit: 5
            njev: 5
          status: 0
         success: True
               x: array([1.        , 4.74299607, 3.82115466, 1.37940764])
```

Figura 2: Um exemplo de otimização utilizando a função *minimize* do pacote *scipy.optimize*.

Na primeira célula a biblioteca *NumPy* e a função *minimize* foram importadas. Na segunda célula a função $f(x)$ da equação (4) foi chamada de *objective(x)* no código *Python*. As funções $c_1(x)$ e $c_2(x)$ são as *constraint1(x)* e *constraint2(x)*.

Na terceira célula os parâmetros da função *minimize* foram configurados, note que as *constraints* foram equacionalmente isoladas na segunda célula e nesta as utilizamos definindo-as ao seu tipo de relação: *'ineq'* para uma desigualdade e *'eq'* para uma igualdade. Os limites, chamados de *bnds* no código (do inglês *boundaries*) são os valores que as variáveis podem assumir.

Na quarta célula, a função de otimização é declarada e é escolhido um método de otimização, chamado *Sequential Least SQuares Programming* ou *'SLSQP'*[2].

Como resultado final da otimização ela se mostra com sucesso (*sucess: True*, na saída da célula), ou seja, obteve-se variáveis que satisfaçam tando as *boudaries* quanto as *constraints* resultando nos valores de $x_{opt} = [1\ 4.74\ 3.82\ 1.38]$ solucionando a função $\text{objective}(x_{opt}) = 17.01$.

O PROBLEMA DE OTIMIZAÇÃO A SER RESOLVIDO. O problema de otimização a ser resolvido é mostrado na equação (5):

$$\min_x f(x) \quad \text{subject to} \quad \begin{cases} f(x) = (\cos(3\theta_1) - \cos(3\theta_2) + \cos(3\theta_3))^2 + (\cos(5\theta_1) - \cos(5\theta_2) + \cos(5\theta_3))^2 \\ c(x) = \cos(\theta_1) - \cos(\theta_2) + \cos(\theta_3) = m_a \\ Ax \leqslant b \\ 0 < \theta_1 < \theta_2 < \theta_3 < \frac{\pi}{2} \end{cases} \tag{5}$$

em que a relação linear $Ax \leqslant b$ parte da matriz A e os vetores b e x que são dados por:

2 Esse é o único método que a função *minimize* aceita que utiliza tanto *boundaries* quanto *constraints* ao mesmo tempo.

$$A = \begin{bmatrix} -1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}, \; b = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{\pi}{2} \end{bmatrix} \text{ e } x = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}.$$

Este problema de otimização seria facilmente solucionado com a função *fmincon* do MATLAB, porém para a utilização pela função *minimize* em Python esse sistema deve ser transformado. A relação linear que define as *constraints* do sistema deve ser resolvida a fim de se obter quatro *constraints* mais a que já estava estabelecida na equação $c(x)$ do sistema (5). Logo, este pode ser descrito como:

$$\min_x f(x) \text{ subject to } \begin{cases} f(x) = (\cos(3\theta_1) - \cos(3\theta_2) + \cos(3\theta_3))^2 + (\cos(5\theta_1) - \cos(5\theta_2) + \cos(5\theta_3))^2 \\ c_{1a}(x) = \cos(\theta_1) - \cos(\theta_2) + \cos(\theta_3) = m_a \\ c_{1b}(x) = -\theta_1 \geqslant 0 \\ c_{2b}(x) = \theta_1 \leqslant \theta_2 \\ c_{3b}(x) = \theta_2 \leqslant \theta_3 \\ c_{4b}(x) = \theta_3 \leqslant \frac{\pi}{2} \\ 0 < \theta_1 < \theta_2 < \theta_3 < \frac{\pi}{2} \end{cases}, \quad (6)$$

em que c_{1a}, c_{1b}, c_{2b}, c_{3b} e c_{4b} formarão a lista de *constraints* do sistema.

3 RESULTADOS E DISCUSSÃO

APLICANDO O NOVO MODELO DE OTIMIZAÇÃO UTILIZANDO A FUNÇÃO *minimize* EM *python*. A Figura 3 mostra o trecho de código semelhante ao da Figura 2 agora com o problema de otimização para o *Synchronous Optimal Modulation* descrito na equação (3).

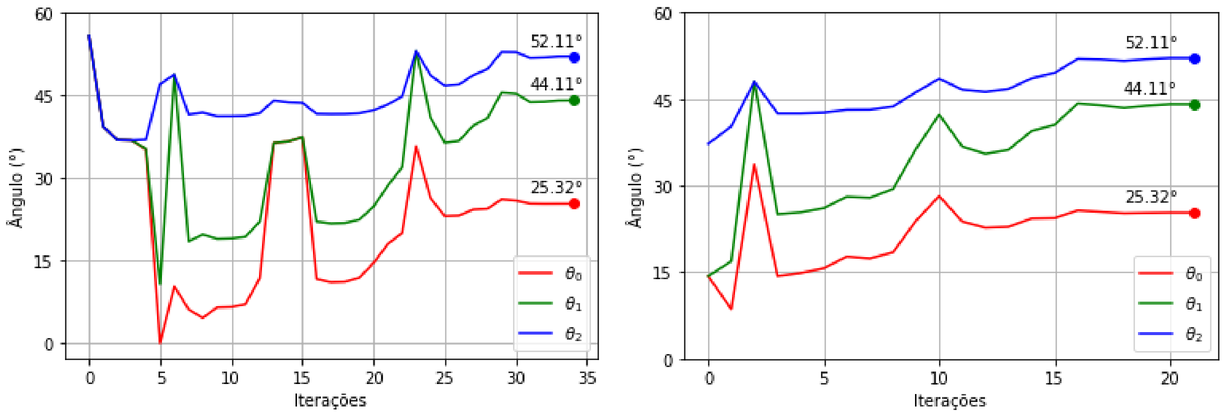
```
In [3]: def objective(theta):
            th1, th2, th3 = theta
            return (np.cos(3*th1) - np.cos(3*th2) + np.cos(3*th3))**2 + (np.cos(5*th1) - np.cos(5*th2) + np.cos(5*th3))**2

        ma = .8

        def constraint1a(theta):
            th1, th2, th3 = theta
            return ma - np.cos(th1) + np.cos(th2) - np.cos(th3)

        def constraint1b(theta):
            th1, th2, th3 = theta
            return th1

        def constraint2b(theta):
            th1, th2, th3 = theta
            return th2 - th1

        def constraint3b(theta):
            th1, th2, th3 = theta
            return th3 - th2

        def constraint4b(theta):
            th1, th2, th3 = theta
            return np.pi/2 - th3

In [4]: x0 = [np.radians(10), np.radians(30), np.radians(85)]

        b = (0., np.pi/2)
        bnds = (b,b,b)
        con1a = {'type': 'eq', 'fun': constraint1a}
        con1b = {'type': 'ineq', 'fun': constraint1b}
        con2b = {'type': 'ineq', 'fun': constraint2b}
        con3b = {'type': 'ineq', 'fun': constraint3b}
        con4b = {'type': 'ineq', 'fun': constraint4b}

        cons = [con1a, con1b, con2b, con3b, con4b]

In [5]: lista_x = []
        sol = minimize(objective, x0, method='SLSQP', bounds=bnds, constraints=cons, callback=get_x); sol

Out[5]:      fun: 2.4350199461850785e-10
             jac: array([ 0.00208428, -0.00096291,  0.00020863])
         message: 'Optimization terminated successfully.'
            nfev: 191
             nit: 35
            njev: 35
          status: 0
         success: True
               x: array([0.4418938 , 0.76980053, 0.90954345])
```

Figura 3: Código da otimização feita para o problema da *SO Modulation*.

A primeira célula da Figura 3 mostra as funções *objective* e *constraints* descritas em Python assim como mostra o sistema na equação (6), considerando um índice de modulação $ma = .8$. Na segunda célula são preparados os parâmetros da função *minimize* com o chute inicial e a construção da lista de contraints.

Na terceira célula podemos ver o resultado final da otimização, os ângulos iniciais $x_0 = [10° \ 30° \ 85°]$ no fim das 35 iterações foram obtidos os valores, em graus, $x_{35} = [25° \ 44° \ 52°]$ mais uma vez dentro das *constraints* e das *boundaries* fornecidas a função de minimização.

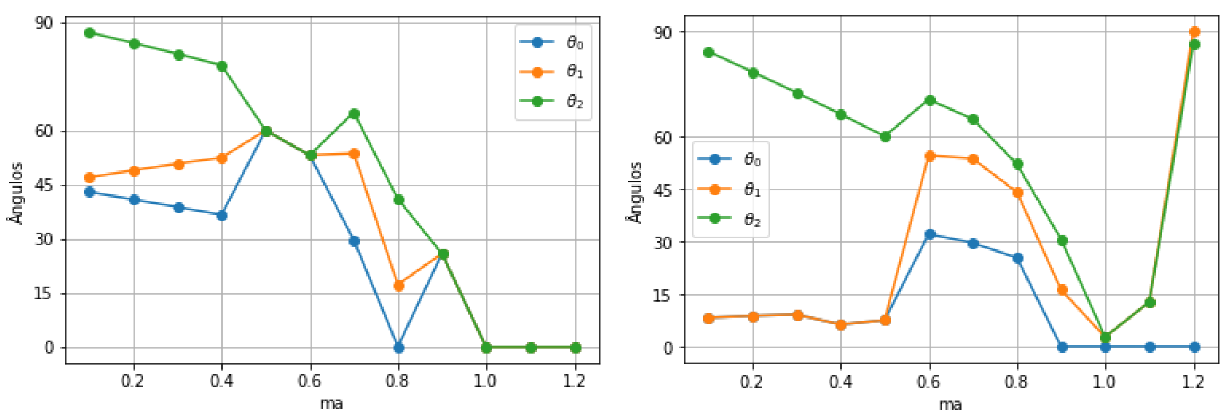
Nesta figura podemos analisar ainda um novo parâmetro na função minimize chamado de *callback*. A função atribuida ao *callback* é chamada a cada iteração da otimização, para podermos acompanhar o processo de otimização, este parâmetro foi setado com a função *get_x* que adicionou a uma lista os valores de x em cada iteração. O resultado da evolução pode ser visto na Figura 4.



(a) (b)

Figura 4: Resultados da otimização em cada iteração para (a) x0 = [10° 30° 50°] e (b) x0 = [13.36° 26.73° 53°] com índice de modulação igual à 0.8.

Os resultados da Figura 5 mostram a otimização considerando diferentes valores do índice de modulação e para diferentes valores de partida.



(a) (b)

Figura 5: Resultados da otimização em cada índice de modulação para (a) x0 = [10° 30° 85°] e (b) x0 = [13.36° 26.73° 53°] com índice de modulação variando entre 0.1 e 1.2.