Instituto Tecnológico de Aeronáutica – ITA
Controle para Sistemas Computacionais – CMC-12
Tutorial sobre Otimização para Controle

Professor: Marcos Ricardo Omena de Albuquerque Maximo

10 de agosto de 2020

1 Introdução a Otimização

De modo geral, um problema de otimização consiste em encontrar o $\mathbf{x}$ ($\mathbf{x}$ pode ser um vetor) que minimiza (ou maximiza) uma determinada função (ou campo escalar) $J(\mathbf{x})$, a qual dá-se o nome de "função objetivo". Em geral, a convenção adotada é a de minimizar a função de objetivo, caso em que esta recebe o nome de função de custo. Note que convencionar minimização não produz limitação, pois maximizar $J(\mathbf{x})$ consiste em minimizar $-J(\mathbf{x})$. Também é comum adotar-se restrições sobre os valores permitidos para $\mathbf{x}$. Em notação matemática, um problema de otimização (de mínimo) é escrito da seguinte forma:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & J(\mathbf{x}) \\ \text{sujeita a} \quad & g_i(\mathbf{x}) \leq 0,\ i = 1, \ldots, m. \\ & h_i(\mathbf{x}) = 0,\ j = 1, \ldots, p. \end{aligned} \tag{1}$$

Perceba que a "dificuldade" do problema de otimização depende das expressões da função de custo e das restrições. Em casos mais simples, o problema de otimização pode ser resolvido analiticamente através do conhecido método de derivar e igualar a zero (no caso de haver restrições, usa-se multiplicadores de Lagrange). Porém, em muitos casos a solução analítica não é possível e é necessário o uso de um algoritmo numérico.

Também definem-se várias classes de problemas de otimização, de acordo com as estruturas da função de custo e das restrições. Por exemplo, a classe conhecida como "Programação Linear" assume função de custo linear e restrições lineares. Note que identificar a classe de certo problema de otimização é útil, pois há algoritmos especificamente desenvolvidos para determinadas classes de problemas de otimização. Estes algoritmos se aproveitam da estrutura do problema para resolvê-lo de forma mais eficiente.

Há ainda algoritmos de otimização que não assumem nada sobre $J(\mathbf{x})$, nem mesmo que esta possa ser escrita na forma de equação matemática. Para estes algoritmos, basta que dado um certo $\mathbf{x}$, seja possível calcular $J(\mathbf{x})$. Assim, o cálculo de $J(\mathbf{x})$ pode consistir até mesmo de um algoritmo computacional complexo ou da realização de um experimento real. Todavia, em problemas com funções de custo que possuem múltiplos mínimos (locais), é comum algoritmos de otimização terem dificuldades para encontrar o verdadeiro mínimo global e convergirem para um mínimo local.

2 Otimização com o MATLAB

No MATLAB, a função `fminsearch` realiza otimização sem restrições. O algoritmo usado pelo `fminsearch` é Nelder-Mead. Para uma rápida explicação sobre o Nelder-Mead, assista à videoaula entregue juntamente com esse tutorial (opcional). Este algoritmo não assume nenhuma estrutura para $J(\mathbf{x})$, portanto pode ser usado com praticamente qualquer função de custo, porém costuma enfrentar problemas de convergência para um mínimo local. O Nelder-Mead requer um chute inicial $\mathbf{x}_0$ que é onde ele começa a busca. Na prática, o algoritmo é muito sensível a este chute inicial e a convergência para o mínimo global é muito mais fácil se o chute inicial estiver próximo deste.

Para problemas de projeto de controlador, que em geral envolvem poucas variáveis, a função `fminsearch` costuma ser adequada. Uma boa ideia para chute inicial é resolver analiticamente uma versão simplificada do problema e então usar a solução analítica como chute inicial. Por exemplo, no caso de uma função de transferência complicada, pode-se ignorar inicialmente dinâmicas mais rápidas de modo a tornar a função de transferência simples o suficiente para permitir uma solução analítica, então esta solução analítica é usada como chute inicial para o `fminsearch`, que opera usando a dinâmica completa da planta, sem aproximações.

Para exemplificar o uso do `fminsearch`, iremos resolver um problema de otimização simples. Considere a seguinte função de duas variáveis:

$$J(x_1, x_2) = (x_1 - 1)^2 + (x_2 - 2)^2 \tag{2}$$

O mínimo desta função pode ser determinado trivialmente como $(1, 2)$. Resolvendo o problema no MATLAB, obtemos:

```
>> J = @(x) (x(1) - 1)^2 + (x(2) - 2)^2;
>> x0 = [0; 0];
>> xMin = fminsearch(f, x0)

xMin =

    1.0000
    2.0000
```

Opcionalmente, podemos usar uma função defina em um `.m`. Por exemplo, no problema acima, pode-se definir `funcaoCusto.m` da seguinte forma:

```
function J = funcaoCusto(x)

J = (x(1) - 1)^2 + (x(2) - 2)^2;

end
```

Desse modo, para resolver o problema, escreve-se:

```
>> f = @(x) funcaoCusto(x);
>> x0 = [0; 0];
>> opcoes = optimset('Display', 'iter'); % para acompanhar a otimizacao
>> xMin = fminsearch(f, x0, opcoes)

 Iteration   Func-count     min f(x)         Procedure
     0           1               5
```

8	1	3	4.999	initial simplex
9	2	5	4.99775	expand
10	3	7	4.99613	expand
11	4	9	4.99282	expand
12	5	11	4.98791	expand
13	6	13	4.97885	expand
14	7	15	4.96455	expand
15	8	17	4.93934	expand
16	9	19	4.89835	expand
17	10	21	4.82804	expand
18	11	23	4.71266	expand
19	12	25	4.51924	expand
20	13	27	4.20589	expand
21	14	29	3.70209	expand
22	15	31	2.93729	expand
23	16	33	1.86901	expand
24	17	35	0.72356	expand
25	18	37	0.421353	reflect
26	19	38	0.421353	reflect
27	20	40	0.421353	contract inside
28	21	42	0.39397	contract inside
29	22	43	0.39397	reflect
30	23	45	0.381766	contract inside
31	24	46	0.381766	reflect
32	25	48	0.381766	contract inside
33	26	50	0.381432	reflect
34	27	52	0.374318	expand
35	28	53	0.374318	reflect
36	29	55	0.364366	expand
37	30	57	0.364366	contract inside
38	31	59	0.349134	expand
39	32	61	0.342406	expand
40	33	63	0.301371	expand
41	34	64	0.301371	reflect
42	35	66	0.238672	expand
43	36	68	0.20157	expand
44	37	70	0.0821063	expand
45	38	72	0.0780134	expand
46	39	74	0.0158599	reflect
47	40	76	0.0158599	contract inside
48	41	78	0.00306375	reflect
49	42	80	0.00306375	contract inside
50	43	82	0.00306375	contract inside
51	44	84	0.00124684	contract outside
52	45	86	0.000127141	contract inside
53	46	88	0.000127141	contract inside
54	47	89	0.000127141	reflect
55	48	91	0.000127141	contract inside
56	49	93	0.000100509	contract outside
57	50	95	1.24708e-05	contract inside
58	51	97	1.24708e-05	contract inside
59	52	99	4.77714e-06	contract outside
60	53	101	4.77714e-06	contract inside
61	54	103	1.05322e-06	contract inside
62	55	105	1.05322e-06	contract inside

```
    56          107      9.02411e-07         contract inside
    57          109      3.21827e-07         contract inside
    58          111      5.77616e-08         contract inside
    59          113      5.77616e-08         contract inside
    60          115      5.21748e-08         reflect
    61          117      4.08498e-08         contract inside
    62          119      2.92757e-09         contract inside
    63          121      2.92757e-09         contract inside
    64          123      2.92757e-09         contract inside
    65          125      2.92757e-09         contract inside
    66          127      1.86918e-09         contract inside

Optimization terminated:
 the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-04
 and F(X) satisfies the convergence criteria using OPTIONS.TolFun of 1.000000e-04


xMin =

    1.0000
    2.0000
```

Perceba que o algoritmo Nelder-Mead convergiu para a solução ótima $(1, 2)$ em 66 iterações. Além disso, atingiu um custo de apenas $1.86918 \cdot 10^{-9}$. O MATLAB também mostra qual operação do Nelder-Mead foi executada em cada iteração.

3 Projeto de Controlador Usando Otimização

Considere um motor elétrico com os seguintes parâmetros:

- Momento de inércia do rotor: $J = 0,01\ kgm^2$.
- Constante de atrito viscoso: $b = 0,1\ Nms$.
- Constante de torque do motor: $K_t = 0,01\ NmA^{-1}$.
- Resistência elétrica: $R = 1\ \Omega$.
- Indutância elétrica: $L = 0,5 \cdot 10^{-3}\ mH$.

A função de transferência da planta é

$$\frac{\dot{\Theta}(s)}{V(s)} = \frac{K_t}{(Js + b)(Ls + R) + K_t^2} \tag{3}$$

Deseja-se erro nulo em regime para entrada degrau. Assim, vamos usar um controlador PI com ganhos K_p e K_i, a função de transferência em malha fechada do sistema fica:

$$G_f(s) = \frac{K_p K_t s + K_i K_t}{JLs^3 + (JR + Lb)\, s^2 + (Rb + K_t^2 + K_p K_t)\, s + K_i K_t} \tag{4}$$

Considere os seguintes requisitos para o sistema:

- Banda passante (de ganho) $\omega_{b,req} = 3\ rad/s$: frequência no diagrama de Bode de ganho em que há uma queda de $-3\ dB$ ($\approx \sqrt{2}/2$).
- Pico de ressonância $M_{r,req} = 0{,}3546\ dB$: pico no diagrama de Bode.

Como o sistema é de terceira ordem e há um zero devido ao uso de PI, é complicado determinar K_p e K_i analiticamente, portanto usa-se um algoritmo de otimização através da função `fminsearch` do MATLAB. Seja $\omega_{b,req}$ e $M_{r,req}$ os requisitos de banda passante e de pico de ressonância, respectivamente, pode-se definir a função de custo como

$$J([K_p, K_i]^T) = \left(\omega_{b,req} - \omega_b([K_p, K_i]^T)\right)^2 + \left(M_{r,req} - M_r([K_p, K_i]^T)\right)^2 \tag{5}$$

em que $\omega_b\left([K_p, K_i]^T\right)$ e $M_r\left([K_p, K_i]^T\right)$ são a banda passante e o pico de ressonância do sistema, respectivamente, calculados numericamente através do diagrama de Bode de $G_f(s)$ para o valor de $[K_p, K_i]^T$ em questão.

Para o chute inicial, resolve-se uma versão simplificada do problema. Inicialmente, considere que a dinâmica da corrente é muito rápida e pode ser desprezada, i.e. $L \approx 0$. Com isso, a função de transferência em malha fechada é simplificada para

$$G_{f,aprox}(s) = \frac{K_p K_t s + K_i K_t}{JRs^2 + (Rb + K_t^2 + K_p K_t)\, s + K_i K_t} \tag{6}$$

Ignorando a presença do zero, tem-se um sistema de segunda ordem padrão, de modo que

$$\omega_n^2 = \frac{K_i K_t}{JR}, \tag{7}$$

$$2\xi\omega_n = \frac{Rb + K_t^2 + K_p K_t}{JR}. \tag{8}$$

Portanto:

$$K_p = \frac{2\xi\omega_n JR - Rb - K_t^2}{K_t}, \tag{9}$$

$$K_i = \frac{JR\omega_n^2}{K_t}. \tag{10}$$

Além disso, como tem-se um sistema de segunda ordem padrão, pode-se usar as seguintes fórmulas para determinar ω_n e ξ a partir dos requisitos:

$$\omega_b = \omega_n\sqrt{1 - 2\xi^2 + \sqrt{2 - 4\xi^2 + 4\xi^4}} \Rightarrow \omega_n = \frac{\omega_b}{\sqrt{1 - 2\xi^2 + \sqrt{2 - 4\xi^2 + 4\xi^4}}}, \tag{11}$$

$$M_r = \frac{1}{2\xi\sqrt{1 - \xi^2}} \Rightarrow \xi = \sqrt{\frac{1}{2}\left(1 - \frac{\sqrt{M_r^2 - 1}}{M_r}\right)}. \tag{12}$$

Finalmente, mostra-se como implementar o método de projeto de controlador no MATLAB. A função `projetarControladorPIMotor` (vide Listagem 1) projeta o controlador conforme o método descrito, enquanto o script `rodarOtimizacao.m` (vide Listagem 2) resolve o problema com os parâmetros considerados no início desta seção. Durante a otimização, o MATLAB mostra resultados obtidos em cada iteração para que se verifique que o algoritmo está convergindo (vide Listagem 3). Além disso, perceba que ao descomentar a linha 44 da função `projetarControladorPIMotor`, o algoritmo converge para uma solução ruim, o que mostra como o chute inicial usando a solução analítica é importante. A Figura 1 apresenta o diagrama de Bode em malha fechada do sistema projetado para comprovar atendimento aos requisitos. Perceba como o sistema projetado por otimização atende perfeitamente aos requisitos!

Listagem 1: Função `projetarControladorPIMotor`.

```
1  function [Kp, Ki] = projetarControladorPIMotor(requisitos, planta)
2  % [Kp, Ki] projeta um controlador PI para um motor eletrico.
3  % A struct requisitos eh dada por:
4  % requisitos.wb: requisito de banda passante.
5  % requisitos.Mr: requisito de pico de ressonancia (em dB).
6  % A struct planta tem os seguintes parametros:
7  % planta.J: inercia.
8  % planta.b: constante de atrito viscoso.
9  % planta.Kt: constante de torque.
10 % planta.R: resistencia.
11 % planta.L: indutancia.
12 % As saidas sao:
13 % Kp: ganho proporcional do controlador.
14 % Ki: ganho integrativo do controlador.
15
16 %% Coletando parametros
17
18 J = planta.J;
19 b = planta.b;
20 Kt = planta.Kt;
21 R = planta.R;
22
23 %% Convertendo os requisitos para wn e xi
24
25 wbReq = requisitos.wb;
26 MrReq = requisitos.Mr;
27 MrReq = 10^(MrReq/20); % pois MrReq esta em dB
28
29 xiReq = sqrt((1 / 2) * (1 - sqrt(MrReq^2 - 1) / MrReq));
30 wnReq = wbReq / sqrt(1 - 2 * xiReq^2 + sqrt(2 - 4 * xiReq^2 + 4 * xiReq^4));
31
32 %% Resolvendo problema simplificado (chute inicial)
33
34 Kp0 = (2 * xiReq * wnReq * J * R - R * b - Kt^2) / Kt;
35 Ki0 = J * R * wnReq^2 / Kt;
36
37 x0 = [Kp0, Ki0];
38
39 %% Resolvendo problema de otimizacao
40
```

```
41 opcoes = optimset('Display', 'iter'); % para imprimir informacoes da iteracao
42
43 J = @(x) funcaoCusto(requisitos, planta, x);
44 % x0 = [40; 40];
45 xOtimo = fminsearch(J, x0, opcoes);
46 Kp = xOtimo(1);
47 Ki = xOtimo(2);
48
49 end
50
51 function J = funcaoCusto(requisitos, planta, parametros)
52 % J = funcaoCusto(requisitos, planta, parametros) calcula o custo associado
53 % a um vetor de parametros [Kp; Ki]. A struct requisitos eh dada por:
54 % requisitos.wb: requisito de banda passante.
55 % requisitos.Mr: requisito de pico de ressonancia (em dB).
56 % A struct planta tem os seguintes parametros:
57 % planta.J: inercia.
58 % planta.b: constante de atrito viscoso.
59 % planta.Kt: constante de torque.
60 % planta.R: resistencia.
61 % planta.L: indutancia.
62
63 %% Coletando parametros
64
65 Kp = parametros(1);
66 Ki = parametros(2);
67
68 J = planta.J;
69 b = planta.b;
70 Kt = planta.Kt;
71 R = planta.R;
72 L = planta.L;
73
74 wbReq = requisitos.wb;
75 MrReq = requisitos.Mr;
76
77 %% Definindo funcao de transferencia de malha fechada
78
79 s = tf('s');
80
81 Gf = (Kp * Kt * s + Ki * Kt) / (J * L * s^3 + (J * R + L * b) * s^2 + (R * b + Kt^2 +
       Kp * Kt) * s + Ki * Kt);
82
83 w = 1e-2:1e-2:1000;
84
85 mag = bode(Gf, w);
86 mag = mag(:);
87 magdB = 20 * log10(mag);
88
89 wb = interp1(magdB, w, -3);
90 Mr = 20 * log10(max(mag));
91
92 J = (wbReq - wb)^2 + (MrReq - Mr)^2;
93
94 end
```

Listagem 2: Script `rodarOtimizacao.m`.

```
%% Definindo parametros

requisitos.wb = 3;
requisitos.Mr = 0.3546;
planta.J = 0.01;
planta.b = 0.01;
planta.Kt = 0.01;
planta.R = 1;
planta.L = 0.5 * 10^-3;

%% Projetando o controlador por otimizacao

[Kp, Ki] = projetarControladorPIMotor(requisitos, planta);

%% Reconstruindo o sistema para verificacao de atendimento aos requisitos

s = tf('s');

J = planta.J;
b = planta.b;
Kt = planta.Kt;
R = planta.R;
L = planta.L;

Gf = (Kp * Kt * s + Ki * Kt) / (J * L * s^3 + (J * R + L * b) * s^2 + (R * b + Kt^2 +
    Kp * Kt) * s + Ki * Kt);

%% Tracando graficos

t = 0:1e-2:5;
y = step(Gf, t);
plot(t, y, 'LineWidth', 2);
xlabel('Tempo (s)', 'FontSize', 14);
ylabel('Velocidade (rad/s)', 'FontSize', 14);
set(gca, 'Fontsize', 14)
title('Resposta ao Degrau', 'FontSize', 14);
grid on;
% print -depsc2 degrau_otimizacao.eps

figure;
w = 1e-2:1e-2:10;
mag = bode(Gf, w);
mag = mag(:);
magdB = 20 * log10(mag);
semilogx(w, magdB, 'LineWidth', 2);
wb = interp1(magdB, w, -3);
Mr = 20 * log10(max(mag));
xlabel('Frequencia (rad/s)', 'FontSize', 14);
ylabel('Magnitude (dB)', 'FontSize', 14);
title(sprintf('Diagrama de Bode\nBanda = %.2f, Pico de Res. = %.2f', wb, Mr));
set(gca, 'FontSize', 14);
grid on;
% print -depsc2 bode_otimizacao.eps
```

Listagem 3: Execução da otimização no MATLAB.

```
>> rodarOtimizacao

 Iteration   Func-count     min f(x)         Procedure
     0            1          2.00476
     1            3          1.84367         initial simplex
     2            5          1.07211         expand
     3            7          0.72653         expand
     4            9         0.148222         expand
     5           10         0.148222         reflect
     6           12         0.141555         reflect
     7           13         0.141555         reflect
     8           15         0.136615         contract inside
     9           16         0.136615         reflect
    10           18         0.125895         contract inside
    11           19         0.125895         reflect
    12           21         0.125895         contract inside
    13           23         0.125895         contract inside
    14           25         0.125837         contract inside
    15           27         0.125783         contract inside
    16           29         0.125768         contract outside
    17           31         0.125768         contract inside
    18           33         0.125745         expand
    19           35         0.106726         expand
    20           37        0.0209237         expand
    21           39       0.00825353         reflect
    22           41       0.00825353         contract inside
    23           43       0.00825353         contract outside
    24           45       0.00825353         contract inside
    25           47       0.00717266         contract inside
    26           49       0.00695314         contract inside
    27           51       0.00681084         reflect
    28           53       0.00669511         contract inside
    29           55       0.00598008         expand
    30           56       0.00598008         reflect
    31           58       0.00492038         expand
    32           59       0.00492038         reflect
    33           61       0.00283692         expand
    34           62       0.00283692         reflect
    35           64      0.000839829         expand
    36           66      0.000455462         expand
    37           67      0.000455462         reflect
    38           69      4.37565e-05         contract inside
    39           71      4.37565e-05         contract inside
    40           73      4.37565e-05         contract inside
    41           75      3.43872e-05         contract inside
    42           77      2.60827e-05         contract outside
    43           79      7.95516e-06         contract inside
    44           81      1.81955e-06         contract inside
    45           83      1.68201e-06         contract outside
    46           85       8.1323e-07         contract inside
    47           87      3.44222e-07         contract inside
    48           89      3.44222e-07         contract inside
    49           91      6.58326e-08         contract inside
    50           92      6.58326e-08         reflect
```

```
    51            94       6.58326e-08         contract inside
    52            96       3.83338e-08         contract inside
    53            98       9.04479e-09         contract inside
    54           100        7.5517e-09         contract inside
    55           102       3.18526e-09         contract outside
    56           104        3.8587e-10         contract inside
    57           106        3.8587e-10         contract outside
    58           108        3.8587e-10         contract inside

Optimization terminated:
 the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-04
 and F(X) satisfies the convergence criteria using OPTIONS.TolFun of 1.000000e-04
```

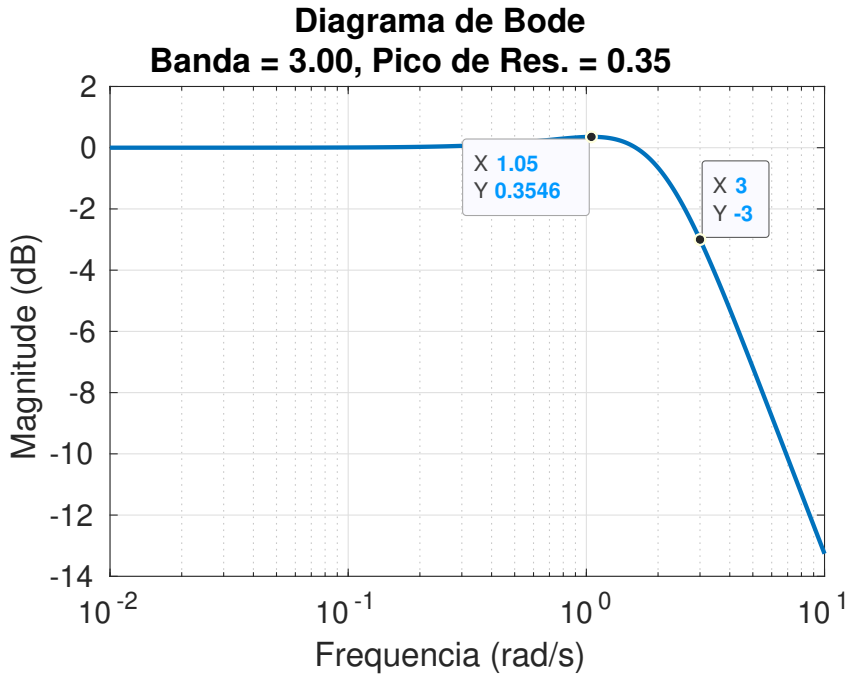

Figura 1: Diagrama de Bode de malha fechada do sistema projetado com otimização.