Trabalho 02

MEC 2403 - Otimização e Algoritmos para Engenharia Mecânica

Pedro Henrique Cardoso Paulo
pedrorjpaulo.phcp@gmail.com

Professor: Ivan Menezes

Departamento de Engenharia Mecânica
PUC-RJ Pontifícia Universidade Católica do Rio de Janeiro
junho de 2023

Trabalho 02

MEC 2403 - Otimização e Algoritmos para Engenharia Mecânica

Pedro Henrique Cardoso Paulo

junho de 2023

1 Introdução

1.1 Objetivos

Esse é o entregável da Trabalho 02 da disciplina MEC 2403 - Otimização e Algoritmos para Engenharia Mecânica. Esse trabalho tem como objetivos:

1. Expandir os métodos de otimização sem restrição (OSR) implementados na Lista 02 para a tarefa de otimização com restrição (OCR)
2. Implementar métodos de penalidade e barreira para restrições
3. Realizar visualização dos resultados finais

Para atingir esse objetivo, este trabalho consistirá na realização de dois exercícios. O primeiro consistirá na aplicação dos métodos de otimização a duas funções distintas, sendo a primeira quadrática e a segunda não. O segundo exercício consistirá na solução de um problema de inspiração física, capaz de ser solucionado pelo método dos elementos finitos.

1.2 Links úteis

Nesta seção são listados alguns links e referências úteis para se entender o trabalho desempenhado.

1. Apostila de programação matemática da disciplina
2. GitHub usado para essa disciplina
3. Notebook com o código para as figuras desse relatório
4. Notebook com o código para derivação simbólica das funções mais complexas
5. Pasta com os códigos a serem aproveitados em todas as listas

2 Materiais e métodos

Nesta seção serão descritos os principais métodos de otimização aplicados neste estudo, tal como detalhes de sua implementação em código Python.

2.1 Otimização sem restrição (OSR)

Os métodos de otimização sem restrição, tal como os detalhes de suas implementações forma descritos no Trabalho 01. Para mais detalhes sobre esses métodos, referenciar o trabalho anterior.

2.2 Otimização com restrições (OCR)

O problema de otimização com restrições consiste em encontrar o mínimo de uma função que respeita um conjunto de restrições pré-determinadas. Um exemplo de problema OCR é dado na equação 1

$$\begin{cases} \min f(\mathbf{x}) \\ h_i(\mathbf{x}) = 0, i \in 1, 2, 3, ..., n_h \\ c_j(\mathbf{x}) \leq 0, j \in 1, 2, 3, ..., n_c \end{cases} . \tag{1}$$

A solução deste problema passa por encontrar o mínimo da função f respeitando as restrições de igualdade h_i e de desigualdade c_j. Embora técnicas analíticas como multiplicadores de Lagrange possam ser plicadas para esse conjunto de problemas, sua aplicação pode ser ineficiente para aplicações numéricas, de modo que métodos como a Penlidade e a Barreira são preferíveis. A estrutura geral para aplicação desses métodos segue a seguinte sequência de passos:

1. Define-se os valores iniciais de $\mathbf{x}_0$, r_p^0 e r_b^0
2. Aplica-se um método de OSR para uma função nova definida $\Phi(\mathbf{x}) = f(\mathbf{x}) + r_p^i p_p(\mathbf{x}) + r_b^i p_b(\mathbf{x})$, determinando assim $\mathbf{x}_{i+1}$
3. Para o ponto de resultado, calcula-se a métrica de erro $r_p^i p_p(\mathbf{x}_{i+1}) + r_b^i p_b(\mathbf{x}_{i+1})$

   Caso o erro seja inferior a uma tolerência, encerramos o processo e retornamos $\mathbf{x}_{i+1}$

   Caso contrário, atualizamos $r_p^{i+1} = \beta_p r_p^i$, $r_b^{i+1} = \beta_b r_b^i$ e retornamos para o item 2. Os valores de β_p e β_b são constantes ao longo da solução

Nas próximas seções serão descritos brevemente os métodos de Penalidade e Barreira.

2.2.1 Método da Penalidade

O método da Penalidade soma a função otimizadora o termo descrito na equação 2

$$p_p(\mathbf{x}) = \sum_i^{n_h} (h(\mathbf{x}))^2 + \sum_j^{n_c} (\max(c(\mathbf{x}), 0))^2 . \tag{2}$$

Dentre as características principais desse método temos a convergência facilitada para pontos iniciais for da região viável (embora pontos internos possam também convergir) e uma tendência de decrescimento da função p_p conforme o ponto se aproxima de atender às restrições, o que demanda que r_p^i tenha uma tendência de crescimento ao longo das iterações (i.e., $\beta_p > 1.0$). Tende a ser um método estável, convergindo para um ponto de mínimo na maioria dos casos.

2.2.2 Método da Barreira

O método da Barreira soma a função otimizadora o termo descrito na equação 3

$$p_b(\mathbf{x}) = -\sum_j^{n_c} \frac{1}{c(\mathbf{x})} . \tag{3}$$

As principais características desse método é sua aplicação únicamente em condições de contorno de inequação e o faato deste convergir apenas para pontos iniciais dentro da região viável. Condições de igualdade podem ser aplicadas por meio do método de penalidade em casos de uso do método da barreira, porém a necessidade de um ponto interno é essencial, havendo necessidade da criação de proteções de código para garantir que durante a busca linear não haja possibilidade de um ponto fora da região viável. Como a tendência de p_b é de crescimento perto da fronteira, devemos garantir que r_b^i tenha tendência de decréscimo (i.e., $\beta_b < 1.0$).

2.3 Implementação

Em adição aos módulos já desenvolvidos e exemplificados no Trabalho 01, foram elaborados os seguintes módulos ccomplementares:

- `constraints.py`: Arquivo contendo as classes que simulam as restrições das funções. Herdam da classe `SpecialFunction`
- `constropt.py`: Arquivo que armazena o código para otimização com restrição. Recebem objetos das classes `GenericOptimizer` e `GenericStep`

Abaixo é apresentado o código do otimizador com restrição, elaborado para esse trabalho.

```
class ConstrainedOptimizer:

def __init__(self, tol, max_iter=200, verbose=False):

    self.tol = tol
    self.max_iter = max_iter
    self.verbose = verbose
    self.clear_cache()

def clear_cache(self):
    self.cache_x = []
    self.iter = 0

def __call__(self, function:ConstrainedSpecialFunction, p_initial, optimizer:GenericOptimizer,
                                                    step:GenericStep):

    self.clear_cache()
    function.start()
    x = p_initial
    errorflag = False
    while self.iter < self.max_iter:
        self.iter += 1
        if self.verbose: print(f'    Beginning iteration: {self.iter}')
        self.cache_x.append(x)
        da_original = step.get_da()
        da = step.get_da()
        x_new = optimizer(function, x, step)
        while not function.check_validity(x_new):
            da = da/10.0
            if self.verbose: print(f'       Original delta alpha too big. trying delta alpha =
                                                                {da}')
            if da <= self.tol:
                errorflag = True
                break
            step.set_da(da)
            x_new = optimizer(function, x, step)
        step.set_da(da_original)
        x = x_new
        if self.verbose: print(f'    Ending iteration: {self.iter}. Final point: {x[0]},{x[1]}
                                                    , loss: {function.loss(*x)}')
        if function.loss(*x) <= self.tol:
            break
        if errorflag:
            print('    A step value inferior to the tolerance is needed in order to solve the
                                                        problem. The iteractive process
                                                        stopped. The best solution is
                                                        returned')
            break
        function.step()
    self.cache_x.append(x)
    return x
```

3 Resultados

Todos os resultados apresentados foram gerados com códigos Python. A tolerância aplicada para a convergência OCR foi de 10^{-6}, mas com o objetivo de facilitar a convergência geral dos problemas, os códigos de OSR e busca linear utilizaram tolerâncias de 10^{-7} e 10^{-8}, respectivamente. Todos os gradientes e hessianas das funções e restrições foram calculados analíticamente e convertidos em funções Python com o auxílio da biblioteca `sympy`.

3.1 Questão 01

Os resultados finais da Questão 01 foram sumarizados na Tabela 1. É possível ver que, para esse problema, todos os métodos de OSR retornaram essencialmente o mesmo ponto final tanto para o método da Penalidade quanto o da Barreira. Além disso, os métodos de OSR para Penalidade e Barreira seguiram essencialmente o mesmo caminho, com basicamente o memo número de passos, o que é esperado dado que todos os métodos de OSR devem, a cada busca de Φ, retornar o mesmo ponto de mínimo.

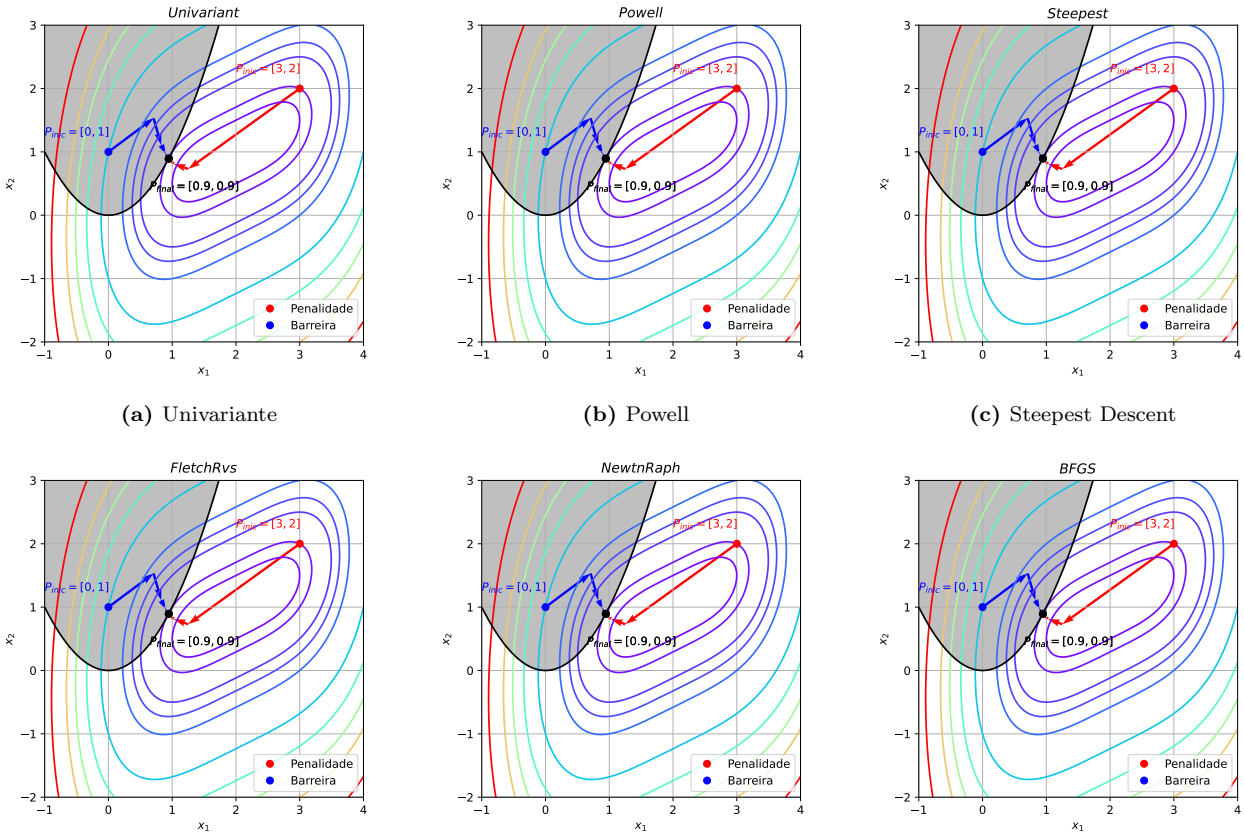
Três problemas específicos foram detectados na realização deste exercício: casos em que o método da barreira atingia pontos fora do domínio, casos em que Powell retornava um vetor de busca nulo e casos em que métodos como o BFGS retornavam vetores inválidos por matriz singular. Os três casos foram tratados por meio da

	Penalidade			Barreira		
Método	**Ponto de mínimo**	**n$_{passos}$**	**t (s)**	**Ponto de mínimo**	**n$_{passos}$**	**t (s)**
Univariante	$[0.942897, 0.889055]^T$	8	0.687	$[0.947491, 0.897739]^T$	13	1.475
Powell	$[0.945580, 0.894121]^T$	8	0.500	$[0.945550, 0.894063]^T$	15	0.619
Steepest Descent	$[0.945543, 0.894051]^T$	8	0.631	$[0.945652, 0.894255]^T$	14	1.617
Fletcher-Reeves	$[0.945584, 0.894129]^T$	8	0.648	$[0.945592, 0.894145]^T$	13	1.307
Newton-Raphson	$[0.945583, 0.894127]^T$	8	0.475	$[0.945583, 0.894126]^T$	15	1.729
BFGS	$[0.945583, 0.894127]^T$	8	0.651	$[0.945584, 0.894127]^T$	15	0.482

Tabela 1: Resumo dos resultados obtidos para a Questão 01

adição de proteções no código para evitar a parada do processo e garantir que o ponto retornado sempre ficasse dentro do domínio. Vale resslatar que para este caso a maioria dos problemas ocorreu apenas no método da barreira.

A Figura 1 apresenta os resultados em gráfico. Novamente, fica evidente que o caminho percorrido, independente do método OSR, foi extremamente similar. Ressalta-se que a região viável nas figuras é marcada com cinza, para todas as restrições.



(a) Univariante **(b)** Powell **(c)** Steepest Descent

(d) Fletcher-Reeves **(e)** Newton-Raphson **(f)** BFGS

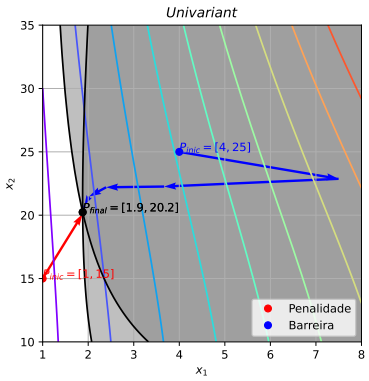
Figura 1: Resultados gráficos para a Questão 01

3.2 Questão 02

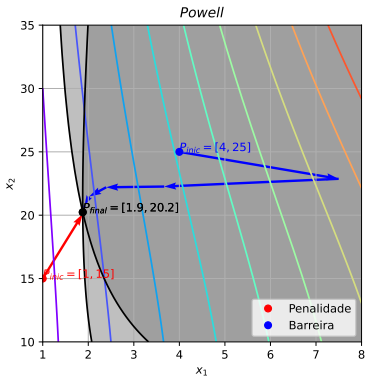
Os resultados obtidos para a Questão 02 são exibidos em forma tabular na Tabela 2 e graficamente na Figura 2. As conclusões obtidas neste caso são similares às obtidas na Questão 01.

	Penalidade			Barreira		
Método	**Ponto de mínimo**	$\mathbf{n}_{passos}$	**t (s)**	**Ponto de mínimo**	$\mathbf{n}_{passos}$	**t (s)**
Univariante	$[1.878357, 20.236907]^T$	6	1.336	$[1.878355, 20.236914]^T$	15	2.902
Powell	$[1.878357, 20.236907]^T$	6	1.087	$[1.878356, 20.236914]^T$	16	0.810
Steepest Descent	$[1.878357, 20.236905]^T$	6	1.300	$[1.878355, 20.237137]^T$	15	3.074
Fletcher-Reeves	$[1.878357, 20.236908]^T$	6	1.369	$[1.878357, 20.236927]^T$	15	2.710
Newton-Raphson	$[1.878357, 20.236907]^T$	6	1.493	$[1.878357, 20.236908]^T$	16	3.068
BFGS	$[1.878357, 20.236907]^T$	6	0.917	$[1.878357, 20.236914]^T$	16	0.860

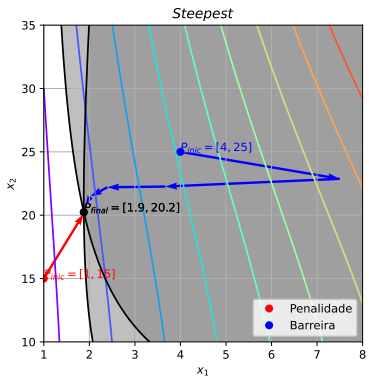
Tabela 2: Resumo dos resultados obtidos para a Questão 02



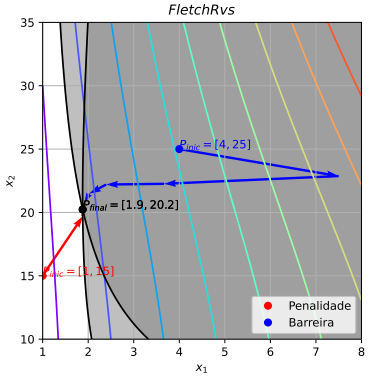
(a) Univariante



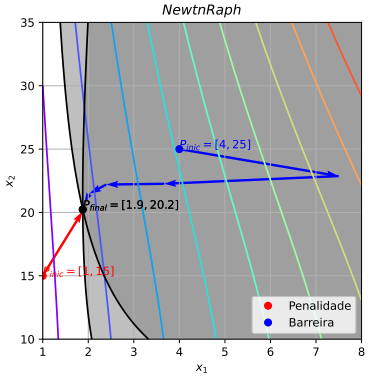
(b) Powell



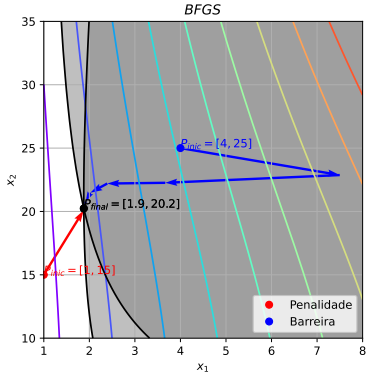
(c) Steepest Descent



(d) Fletcher-Reeves



(e) Newton-Raphson



(f) BFGS

Figura 2: Resultados gráficos para a Questão 02

As funções de restrição aplicadas nesse caso são mostradas a seguir. Cabe ressaltar que a formulação do problema foi retirada dos slides e material de aula, com derivadas analíticas obtidas com a ajuda do `sympy`.

```
cfs = []

def cf(d, H):

    return P * np.sqrt(H**2 + B**2) / (np.pi * d * t * H) - sigma_y

def gradf(d, H):

    return np.array([-P*np.sqrt(B**2 + H**2)/(H*d**2*np.pi*t), P/(d*np.pi*t*np.sqrt(B**2 + H
                                                **2)) - P*np.sqrt(B**2 + H**2)/(H**2*d*np
                                                .pi*t)])

def hessf(d, H):

    return np.array([[2*P*np.sqrt(B**2 + H**2)/(H*d**3*np.pi*t), -P/(d**2*np.pi*t*np.sqrt(B
                                                **2 + H**2)) + P*np.sqrt(B**2 + H**2)/(H
                                                **2*d**2*np.pi*t)], [-P/(d**2*np.pi*t*np.
                                                sqrt(B**2 + H**2)) + P*np.sqrt(B**2 + H**
                                                2)/(H**2*d**2*np.pi*t), -H*P/(d*np.pi*t*(
                                                B**2 + H**2)**(3/2)) - P/(H*d*np.pi*t*np.
                                                sqrt(B**2 + H**2)) + 2*P*np.sqrt(B**2 + H
                                                **2)/(H**3*d*np.pi*t)]])

cf = functions.AnalyticalSpecialFunction(cf, gradf, hessf)
cfs.append(cf)

def cf(d, H):

    return P * np.sqrt(H**2 + B**2) / (np.pi * d * t * H) - np.pi **2 * E * (d**2+t**2) / (8
                                                .0*(H**2 + B**2))

def gradf(d, H):

    return np.array([-2*E*d*np.pi**2/(8.0*B**2 + 8.0*H**2) - P*np.sqrt(B**2 + H**2)/(H*d**2*
                                                np.pi*t), 0.25*E*H*np.pi**2*(d**2 + t**2)
                                                /(B**2 + H**2)**2 + P/(d*np.pi*t*np.sqrt(
                                                B**2 + H**2)) - P*np.sqrt(B**2 + H**2)/(H
                                                **2*d*np.pi*t)])

def hessf(d, H):

    return np.array([[-2*E*np.pi**2/(8.0*B**2 + 8.0*H**2) + 2*P*np.sqrt(B**2 + H**2)/(H*d**3
                                                *np.pi*t), 0.5*E*H*d*np.pi**2/(B**2 + H**
                                                2)**2 - P/(d**2*np.pi*t*np.sqrt(B**2 + H
                                                **2)) + P*np.sqrt(B**2 + H**2)/(H**2*d**2
                                                *np.pi*t)], [0.5*E*H*d*np.pi**2/(B**2 + H
                                                **2)**2 - P/(d**2*np.pi*t*np.sqrt(B**2 +
                                                H**2)) + P*np.sqrt(B**2 + H**2)/(H**2*d**
                                                2*np.pi*t), -1.0*E*H**2*np.pi**2*(d**2 +
                                                t**2)/(B**2 + H**2)**3 + 0.25*E*np.pi**2*
                                                (d**2 + t**2)/(B**2 + H**2)**2 - H*P/(d*
                                                np.pi*t*(B**2 + H**2)**(3/2)) - P/(H*d*np
                                                .pi*t*np.sqrt(B**2 + H**2)) + 2*P*np.sqrt
                                                (B**2 + H**2)/(H**3*d*np.pi*t)]])

cf = functions.AnalyticalSpecialFunction(cf, gradf, hessf)
cfs.append(cf)
```

3.3 Questão 03

O principal desafio da Questão 03 é o pequeno tamanho de sua zona viável, que consiste basicamente no diminuto espaço entre as duas restrições circulares, tal como ilustrado na Figura 3. Embora para o método de Penalidade isso não represente um desafio, para o método de Barreira essa característica é um problema, que inviabiliza o ponto inicial proposto no enunciado. Como forma de poder usar o método de barreira neste problema, um novo ponto inicial $[15.05, 4.8]^T$ foi proposto e usado. Demais parâmetros foram mantidos conforme o enunciado.

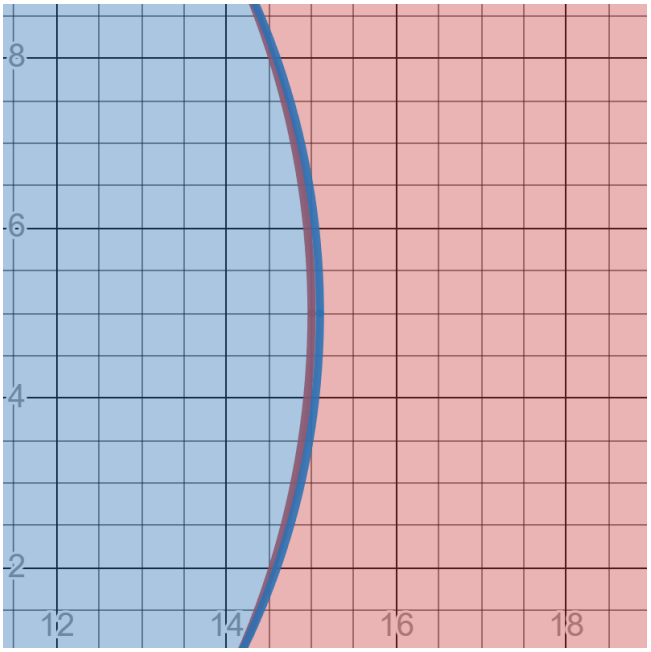
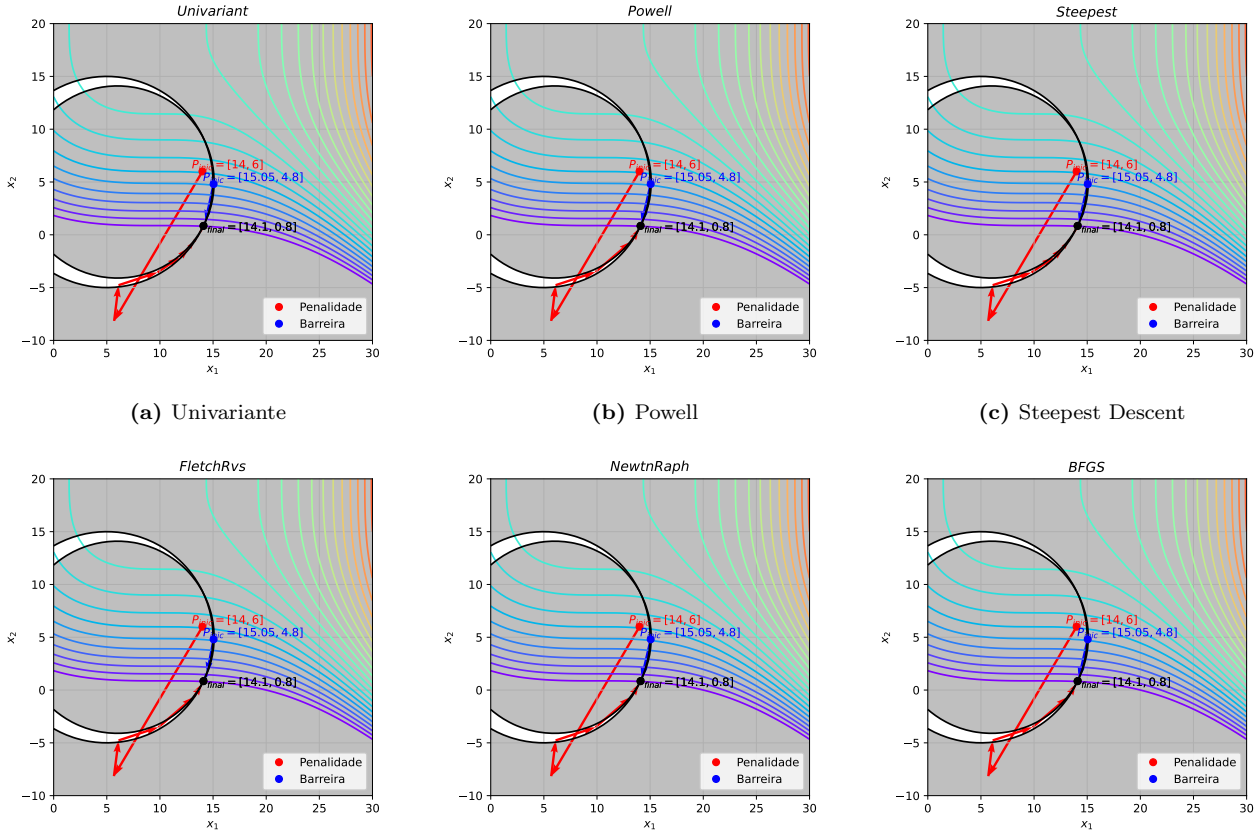


Figura 3: Região possível para as soluções da Questão 03

Os resultados obtidos para a Questão 03 são exibidos em forma tabular na Tabela 3 e graficamente na Figura 4. As conclusões obtidas neste caso são similares às obtidas na Questão 01 e 02.

	Penalidade			**Barreira**		
Método	**Ponto de mínimo**	$\mathbf{n}_{passos}$	**t (s)**	**Ponto de mínimo**	$\mathbf{n}_{passos}$	**t (s)**
Univariante	$[14.094999, 0.842959]^T$	50	7.826	$[14.095016, 0.842995]^T$	10	1.607
Powell	$[14.095000, 0.842961]^T$	12	2.007	$[14.095023, 0.843006]^T$	9	0.805
Steepest Descent	$[14.095026, 0.843012]^T$	43	6.768	$[14.095157, 0.843304]^T$	8	1.399
Fletcher-Reeves	$[14.095001, 0.842963]^T$	14	2.507	$[14.095009, 0.842980]^T$	9	1.903
Newton-Raphson	$[14.095000, 0.842961]^T$	50	8.233	$[14.095000, 0.842960]^T$	13	2.355
BFGS	$[14.095000, 0.842961]^T$	13	2.026	$[14.095001, 0.842961]^T$	12	0.518

Tabela 3: Resumo dos resultados obtidos para a Questão 03



(a) Univariante **(b)** Powell **(c)** Steepest Descent

(d) Fletcher-Reeves **(e)** Newton-Raphson **(f)** BFGS

Figura 4: Resultados gráficos para a Questão 03

3.4 Questão 04

O problema proposto para a Questão 04 é o mais complexo da lista em termos de domínio e restrições. Baseado nas equações do enunciado e natureza do problema algumas mudanças no conjunto de restrições foram feitas, a saber:

- As restrições de máximo para R e t foram removidas, dado que a função objetivo já tende a levar para valores menores
- Foi criada uma restrição que forçasse $R \geq t$, para evitar soluções em que a espessura fosse superior ao raio

Ainda no escopo do enunciado da Questão 04, foi solicitada a seleção de um conjunto de condições iniciais ($\mathbf{x}_0$, r^0 e β). Após alguns testes preliminares, os parâmetros para cada caso foram selecionados. Os valores usados estão sumarizados na Tabela 4.

Método	$\mathbf{x}_0$	r^0	β
Penalidade	$[0.2, 0.15]^T$	10^5	1×10^3
Barreira	$[0.5, 0.025]^T$	10^1	5×10^{-1}

Tabela 4: Condições iniciais para a Questão 04

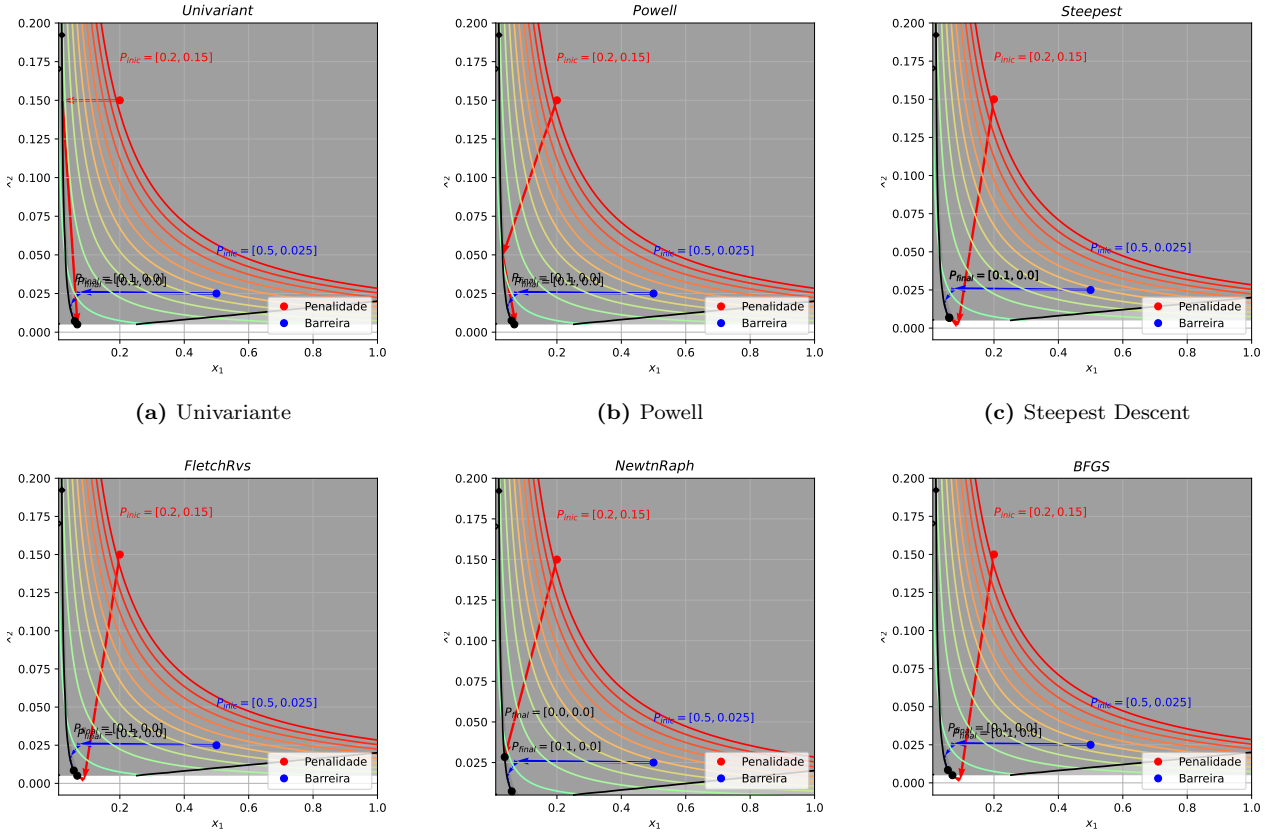
Os resultados obtidos para a Questão 04 estão sumarizados na Tabela 5. Diferentemente das demais questões neste trabalho, para este caso os métodos de Penalidade e Barreira retornaram valores diferentes. Comparando os resultados para o método de Barreira, é possível ver que todos os métodos OSR aplicados retornaram pontos relativamente próximos, o que está de acordo com o esperado uma vez que todos os métodos, em cada passo, buscaram retornar o mínimo da mesma função. O número de passos da otimização OCR para o caso de barreira é evidência também disso. Já no caso do método de Penalidade, embora a maioria dos métodos OSR tenham retornado essencialmente o mesmo ponto (tendo este ponto, inclusive, um valor de peso a ser minimizado inferior ao ponto obtido na Penalidade), vale notar que dois métodos (Steepest e Newton) retornaram pontos muito distintos dos demais. Enquanto o ponto obtido pelo Steepest Descent tem a seu favor ser próximo do obtido

pelos métodos quando aplicado o método da Barreira, o ponto obtido pelo método de Newton é claramente um erro numérico, em que pelas características do método após um passo a convergência foi atingida.

	Penalidade			**Barreira**		
Método	**Ponto de mínimo**	$\mathbf{n}_{passos}$	**t (s)**	**Ponto de mínimo**	$\mathbf{n}_{passos}$	**t (s)**
Univariante	$[0.067747, 0.005000]^T$	3	1.337	$[0.060146, 0.007125]^T$	10	3.430
Powell	$[0.067748, 0.005000]^T$	4	1.566	$[0.059181, 0.007477]^T$	9	1.302
Steepest Descent	$[0.061623, 0.006629]^T$	2	1.036	$[0.060623, 0.006959]^T$	10	3.466
Fletcher-Reeves	$[0.067747, 0.005000]^T$	3	1.353	$[0.056907, 0.008404]^T$	8	2.756
Newton-Raphson	$[0.037870, 0.028403]^T$	1	0.531	$[0.059698, 0.007286]^T$	9	3.836
BFGS	$[0.071021, 0.005000]^T$	5	2.369	$[0.056884, 0.008414]^T$	8	1.381

Tabela 5: Resumo dos resultados obtidos para a Questão 04

A Figura 5 corrobora os comentários anteriormente feitos ao mostrar os resultados da Questão 04 de forma gráfica.



(a) Univariante **(b)** Powell **(c)** Steepest Descent

(d) Fletcher-Reeves **(e)** Newton-Raphson **(f)** BFGS

Figura 5: Resultados gráficos para a Questão 04