Trabalho 2

Opção 01

MEC 2403 - Otimização, Algoritmos e Aplicações na Engenharia Mecânica

Gustavo Henrique Gomes dos Santos
gustavohgs@gmail.com

Professor: Ivan Menezes

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

Trabalho 2

MEC 2403 - Otimização, Algoritmos e Aplicações na Engenharia Mecânica

Gustavo Henrique Gomes dos Santos

junho de 2023

1 Introdução

1.1 Objetivos

Esse trabalho tem como objetivo a implementação e teste dos seguintes métodos indiretos de otimização com restrição :

1. Penalidade
2. Barreira

Para a etapa de sequência de otimização sem restrição, da qual esses métodos fazem uso, foram utilizados os métodos implementados no trabalho 1. São eles:

1. Univariante
2. Powell
3. Steepest Descent
4. Fletcher-Reeves
5. BFGS
6. Newton-Raphson

Estes métodos, por sua vez, fazem uso dos seguintes algoritmos de busca unidimensional também implementados em trabalhos anteriores :

1. Passo constante
2. Seção áurea

2 Implementação

Foi utilizada a linguagem de programação Python para elaboração deste trabalho. A implementação consiste de um arquivo com o código principal, um segundo arquivo com os algoritmos dos métodos de otimização com restrição, um terceiro arquivo que encapsula os métodos e a convergência da otimização sem restrição e um quarto arquivo com os algoritmos da busca unidimensional.

2.1 Código Principal

O código principal trata das definições do ponto inicial, função a ser minimizada e as restrições de igualdade e desigualdade. A escolha de quais métodos OCR e OSR serão utilizados e a definição dos parâmetros numéricos também são feitos no código principal. Além disso, o controle de passos da otimização OC e a verificação de convergência.

Os seguintes pacotes foram utilizados na implementação do código principal, já inclusos os arquivos .py com as implementações dos métodos OSR, busca unidimensional e métodos OCR.

```
import numpy as np
import matplotlib.pyplot as plt
import osr_methods as osr
import line_search_methods as lsm
import ocr_methods as ocr
```

Definição do ponto inicial, que irá variar em cada teste realizado com diferentes funções e método OCR.

```
x = np.array([3., 2.])
```

Escolha dos métodos de OCR e OSR.

```
# Metodos OCR
# 1 - Penalidade
# 2 - Barreira
metodo_ocr = 1

if (metodo_ocr == 1):
    n_met_ocr = "Penalidade"
elif (metodo_ocr == 2):
    n_met_ocr = "Barreira"

# Metodos OSR
# 1 - Univariante
# 2 - Powell
# 3 - Stepest Descent
# 4 - Newton-Raphson
# 5 - Fletcher-Reeves
# 6 - BFGS
metodo_osr = 4

if (metodo_osr == 1):
    n_met = 'Univariante'
elif (metodo_osr == 2):
    n_met = 'Powell'
elif (metodo_osr == 3):
    n_met = 'Steepest Descent'
elif (metodo_osr == 4):
    n_met = 'Newton-Raphson'
elif (metodo_osr == 5):
    n_met = 'Fletcher-Reeves'
elif (metodo_osr == 6):
    n_met = 'BFGS'
```

Controle numérico

```
# numero maximo de iteracoes na OSR
maxiter = 1000

# tolerancia para convergencia do gradiente na OSR
tol_conv = 1E-6

# tolerancia para a busca unidirecional na OSR
tol_search = 1E-7

# delta alpha do passo constante na OSR
line_step = 1E-2

#epsilon da maquina
eps = 1E-10

#parametros ocr
if metodo_ocr == 1:
    #penalidade
    r = 1
    beta = 10
elif metodo_ocr == 2:
    #barreira
    r = 10
    beta = 0.1

#tolerancia OCR
```

```
tol = 1E-6

ctrl_num_osr = [maxiter, tol_conv, tol_search, line_step, eps]
```

Definição de $f(\overrightarrow{x})$, $\overrightarrow{\nabla} f(\overrightarrow{x})$ e $\mathbf{H}(\overrightarrow{x})$. As reticências no código abaixo serão substituiídas pelos valores adequados a cada função.

```
def f(x):
  return ...

def grad_f(x):
  return ...

def hess_f(x):
  hess = np.zeros((2,2), dtype=float)
  hess[0,:] = ...
  hess[1,:] = ...
  return hess
```

Definição das restrições $c_l(\overrightarrow{x})$, $\overrightarrow{\nabla} c_l(\overrightarrow{x})$, $h_k(\overrightarrow{x})$, $\overrightarrow{\nabla} h_k(\overrightarrow{x})$, $\mathbf{W}_{c_l}(\overrightarrow{x})$ e $\mathbf{W}_{h_k}(\overrightarrow{x})$, sendo W a hessiana da restrição. As reticências no código abaixo serão substituídas pelos valores adequados a cada função. Podem ser definidas quantas restrições forem desejadas, apesar do exemplo abaixo estar com apenas uma de desigualdade e uma de igualdade.

```
def h1(x):
  return ...

def grad_h1(x):
  return ...

def hess_h1(x):
  hess = np.zeros((2,2), dtype=float)
  hess[0,:] = ...
  hess[1,:] = ...
  return hess

def c1(x):
  return ...

def grad_c1(x):
  return ...

def hess_c1(x):
  hess = np.zeros((2,2), dtype=float)
  hess[0,:] = ...
  hess[1,:] = ...
  return hess
```

Agrupamento das restrições em listas(ou arrays) para servirem de input para o restante do programa. E montagem de um array auxiliar para montagem da função ϕ no caso do método OCR de penalidade.

```
h_list = [h1]
grad_h_list = [grad_h1]
hess_h_list = [hess_h1]

c_list = [c1]
grad_c_list = [grad_c1]
hess_c_list = [hess_c1]

#para o metodo de penalidade
#controle de quais cls irao montar a phi
c_mont = []
if metodo_ocr == 1:
    for c in c_list:
        if c(x) > 0:
            c_mont.append(1)
        else:
            c_mont.append(0)
```

```
params = [f, grad_f, hess_f, h_list, grad_h_list, hess_h_list, c_list, grad_c_list,
                                                    hess_c_list, c_mont]
```

Verificação de convergência.

1. Penalidade : $\frac{1}{2} r_p^k p(\overrightarrow{x}^{k+1}) < tol$, sendo $p(\overrightarrow{x}) = \sum_{k=1}^{m} h_k^2(\overrightarrow{x}) + \sum_{l=1}^{p} \{max[0, c_l(\overrightarrow{x})]\}^2$
2. Barreira : $r_b^k b(\overrightarrow{x}^{k+1}) < tol$, sendo $b = \sum_{l=1}^{p} -\frac{1}{c_l(\overrightarrow{x})}$

A cada iteração do loop no código abaixo, o valor de r é atualizado pelo fator β, e os parâmetros necessários para montagem da função ϕ pelos métodos da penalidade e da barreira também são atualizados. Após cada atualização e verificação de convergência, a otimização sem restrição é chamada, e esse processo é repetido até que seja atingido o critério de convergência da otimização com restrição.

Para o método da barreira também é feita uma avaliação se o ponto final da iteração continua dentro das restrições. Caso negativo, o passo é refeito com um passo reduzido no método do passo constante da busca unidimensional.

```
if metodo_ocr == 1:
  parc = (1/2)*r*ocr.p_penal(x, params)
elif metodo_ocr == 2:
  parc = r*ocr.b_bar(x, params)

listP_OCR = []
listP_OCR.append(x)

listResultsOSR = []

passos_OCR = 0
redo = 0
print(n_met)
while(parc > tol):
  passos_OCR = passos_OCR + 1
  if passos_OCR > 1:
      r = beta*r
      if metodo_ocr == 1:
          params[-1] = []
          for c in c_list:
              if c(x) > 0:
                  params[-1].append(1)
              else:
                  params[-1].append(0)
  listP_OSR, passos_OSR, conv_OSR, flag_conv_OSR, tempoExec_OSR = osr.osr_ctrl(x, params, r,
                                                      ctrl_num_osr, metodo_ocr, metodo_osr)

  if metodo_ocr == 2:
      redo = 0
      for c in c_list:
          if c(listP_OSR[-1]) > 0:
              redo = 1
              break
  if (redo == 0):
      ctrl_num_osr[3] = line_step
      x = listP_OSR[-1]
      listP_OCR.append(x)
      listResultsOSR.append([listP_OSR, params, r, metodo_ocr, metodo_osr])
      if metodo_ocr == 1:
          parc = (1/2)*r*ocr.p_penal(x, params)
      elif metodo_ocr == 2:
          parc = r*ocr.b_bar(x, params)
      print(f'{passos_OCR}: x={x}, r={r:.4e}, passos={passos_OSR}, conv_OCR={parc:.4e},
                                                      conv_OSR={conv_OSR:.4e}, tempo={
                                                      tempoExec_OSR}')
  elif (redo == 1):
      print(f'Refazendo passo {passos_OCR} com delta alpha = {0.1*ctrl_num_osr[3]}')
      passos_OCR = passos_OCR - 1
      r = r/beta
      ctrl_num_osr[3] = 0.1*ctrl_num_osr[3]
```

2.2 Métodos OCR

Os algoritmos referentes ao uso dos métdos OCR da penalidade e da barreira foram implementados em um arquivo denominado ocr_methods.py.

O seguinte pacote é necessário nesse arquivo:

```
import numpy as np
```

2.2.1 Método de Penalidade

Pseudo-Função Objetivo:

$\phi(\overrightarrow{x}, r_p) = f(\overrightarrow{x}) + \frac{1}{2} r_p \sum_{k=1}^{m} h_k^2(\overrightarrow{x}) + \frac{1}{2} r_p \sum_{l=1}^{p} \{max[0, c_l(\overrightarrow{x})]\}^2$

Cálculo do gradiente:

Sendo $p(\overrightarrow{x}) = \sum_{k=1}^{m} h_k^2(\overrightarrow{x}) + \sum_{l=1}^{p} \{max[0, c_l(\overrightarrow{x})]\}^2$, temos então que $\overrightarrow{\nabla}\phi(\overrightarrow{x}, r_p) = \overrightarrow{\nabla} f(\overrightarrow{x}) + \frac{1}{2} r_p \overrightarrow{\nabla} p(\overrightarrow{x})$.

$\overrightarrow{\nabla} p(\overrightarrow{x}) = 2 \sum_{k=1}^{m} \{h_k(\overrightarrow{x}) \overrightarrow{\nabla} h_k(\overrightarrow{x})\} + 2 \sum_{l=1}^{p} \{c_l^f(\overrightarrow{x}) c_l(\overrightarrow{x}) \overrightarrow{\nabla} c_l(\overrightarrow{x})\}$, sendo $c_l^f(\overrightarrow{x}) = \begin{cases} 1, & \text{se } c_l(\overrightarrow{x}) > 0 \\ 0, & \text{se } c_l(\overrightarrow{x}) \leq 0 \end{cases}$

Cálculo da Hessiana:

$H_\phi(\overrightarrow{x}) = H_f(\overrightarrow{x}) + \frac{1}{2} r_p H_p(\overrightarrow{x})$

$H_{p_{ixj}} = \frac{\partial^2 p}{\partial x_i \partial x_j} = 2 \sum_{k=1}^{m} \{\frac{\partial h_k}{\partial x_j} \frac{\partial h_k}{\partial x_i}\} + 2 \sum_{k=1}^{m} \{h_k \frac{\partial^2 h_k}{\partial x_j \partial x_i}\} + 2 \sum_{l=1}^{p} \{c_l^f \frac{\partial c_l}{\partial x_j} \frac{\partial c_l}{\partial x_i}\} + 2 \sum_{l=1}^{p} \{c_l^f c_l \frac{\partial^2 c_l}{\partial x_j \partial x_i}\}$

Interessante notar que $\frac{\partial c_l}{\partial x_j}$, $\frac{\partial c_l}{\partial x_i}$, $\frac{\partial h_k}{\partial x_j}$ e $\frac{\partial h_k}{\partial x_i}$ são componentes conhecidos de $\overrightarrow{\nabla} h_k$ e $\overrightarrow{\nabla} c_l$. Além disso, $\frac{\partial^2 c_l}{\partial x_j \partial x_i}$ e $\frac{\partial^2 h_k}{\partial x_j \partial x_i}$ são termos das hessianas das restrições e que também são conhecidos. Dessa forma, temos todos os inputs necessários para cálculo da hessiana de $p(\overrightarrow{x})$ e por conseguinte a hessiana de $\phi(\overrightarrow{x}, r)$.

```
#Metodo da Penalidade
def p_penal(x, params):
    #leitura dos parametros
    h_list = params[3]
    c_list = params[6]
    c_mont = params[9]

    p = 0
    for h in h_list:
        p = p + (h(x))**2

    for i in np.arange(len(c_list)):
        p = p + c_mont[i]*c_list[i](x)**2

    return  p

def phi_penal(x, params, r):
    #leitura dos parametros
    f = params[0]
    h_list = params[3]
    c_list = params[6]
    c_mont = params[9]

    p = 0
    for h in h_list:
        p = p + (h(x))**2

    for i in np.arange(len(c_list)):
        p = p + c_mont[i]*c_list[i](x)**2

    return f(x) + (1/2)*r*p

def grad_phi_penal(x, params, r):
    #leitura dos parametros
    grad_f = params[1]
    h_list = params[3]
    grad_h_list = params[4]
    c_list = params[6]
    grad_c_list = params[7]
    c_mont = params[9]

    dimens = x.size
```

```
    grad_p = np.zeros(dimens, dtype=float)

    for i in np.arange(len(h_list)):
        grad_p = grad_p + 2*h_list[i](x)*grad_h_list[i](x)
    for j in np.arange(len(c_list)):
        grad_p = grad_p + 2*c_mont[j]*c_list[j](x)*grad_c_list[j](x)

    return grad_f(x) + (1/2)*r*grad_p

def hess_phi_penal(x, params, r):
    #leitura dos parametros
    hess_f = params[2]
    h_list = params[3]
    grad_h_list = params[4]
    hess_h_list = params[5]
    c_list = params[6]
    grad_c_list = params[7]
    hess_c_list = params[8]
    c_mont = params[9]

    dimens = x.size
    hessian_p = np.zeros((dimens, dimens), dtype=float)

    for i in np.arange(dimens):
        for j in np.arange(dimens):
            for k in np.arange(len(grad_h_list)):
                hessian_p[i,j] = hessian_p[i,j] + 2*grad_h_list[k](x)[i]*grad_h_list[k](x)[j
                                                                                           ]
            for l in np.arange(len(grad_cl_list)):
                hessian_p[i,j] = hessian_p[i,j] + 2*c_mont[l]*grad_c_list[l](x)[j]*
                                                                grad_c_list[l](x)[i]

    for k in np.arange(len(h_list)):
        hessian_p = hessian_p + 2*h_list[k](x)*hess_h_list[k](x)

    for k in np.arange(len(c_list)):
        hessian_p = hessian_p + 2*c_mont[k]*c_list[k](x)*hess_c_list[k](x)

    return hess_f(x) + (1/2)*r*hessian_p
```

2.2.2 Método de Barreira

Pseudo-Função Objetivo:

$\phi(\overrightarrow{x}, r_b) = f(\overrightarrow{x}) + r_b \sum_{l=1}^{m} -\frac{1}{c_l(\overrightarrow{x})}$

Cálculo do gradiente:

Sendo $b(\overrightarrow{x}) = \sum_{l=1}^{m} -\frac{1}{c_l(\overrightarrow{x})}$, temos então que $\overrightarrow{\nabla}\phi(\overrightarrow{x}, r_b) = \overrightarrow{\nabla} f(\overrightarrow{x}) + r_b \overrightarrow{\nabla} b(\overrightarrow{x})$.

$\overrightarrow{\nabla} b = \sum_{l=1}^{m} \frac{\overrightarrow{\nabla} c_l}{c_l^2}$

Cálculo da Hessiana:

$H_\phi(\overrightarrow{x}) = H_f(\overrightarrow{x}) + r_b H_b(\overrightarrow{x})$

$H_{b_{ixj}} = -2\sum_{l=1}^{m}\{\frac{1}{c_l^3}\frac{\partial c_l}{\partial x_i}\frac{\partial c_l}{\partial x_j}\} + \sum_{l=1}^{m}\{\frac{1}{c_l^2}\frac{\partial^2 c_l}{\partial x_i \partial x_j}\}$

$\frac{\partial c_l}{\partial x_j}$ e $\frac{\partial c_l}{\partial x_i}$ são componentes conhecidos de $\overrightarrow{\nabla} c_l$. Além disso, $\frac{\partial^2 c_l}{\partial x_j \partial x_i}$ são termos da hessiana das restrições e que também são conhecidos. Dessa forma, temos todos os inputs necessários para cálculo da hessiana de $b(\overrightarrow{x})$ e por conseguinte a hessiana de $\phi(\overrightarrow{x}, r)$.

```
#### Metodo da Barreira
def phi_bar(x, params, r):
    #leitura dos parametros
    f = params[0]
    c_list = params[6]

    b = 0
    for c in c_list:
```

```
        b = b - 1/cl(x)

    return f(x) + r*b

def b_bar(x, params):
    #leitura dos parametros
    c_list = params[6]

    b = 0
    for c in c_list:
        b = b - 1/c(x)

    return b

def grad_phi_bar(x, params, r):
    #leitura dos parametros
    grad_f = params[1]
    c_list = params[6]
    grad_c_list = params[7]

    dimens = x.size
    grad_b = np.zeros(dimens, dtype=float)

    for i in np.arange(len(c_list)):
        grad_b = grad_b + (c_list[i](x))**(-2)*grad_c_list[i](x)

    return grad_f(x) + r*grad_b

def hess_phi_bar(x, params, r):
    #leitura dos parametros
    hess_f = params[2]
    c_list = params[6]
    grad_c_list = params[7]
    hess_c_list = params[8]

    dimens = x.size
    hessian_b = np.zeros((dimens, dimens), dtype=float)

    for i in np.arange(dimens):
        for j in np.arange(dimens):
            for k in np.arange(len(c_list)):
                hessian_b[i,j] = hessian_b[i,j] - 2*((c_list[k](x))**(-3))*grad_c_list[k](x)
                                                                    [i]*grad_c_list[k](x)[j]

    for k in np.arange(len(cl_list)):
        hessian_b = hessian_b + ((c_list[k](x))**(-2))*hess_c_list[k](x)

    return hess_f(x) + r*hessian_b
```

2.3 Métodos e Otimização OSR

Os algoritmos de otimização sem restrição, Univariante, Powell, Steepest Descent, Newton-Raphson, Flecther-Reeves e BFGS foram implementados em um arquivo denominado osr_methods.py. Uma função chamada osr_ctrl também está presente nesse arquivo e faz a interface entre o código principal e os métodos de OSR. O código principal chama essa função a cada iteração OCR, e ela por sua vez faz todo o tratamento da OSR, chamando as funções ϕ do módulo OCR apresentado na seção anterior E retornando os resultados para o código principal.

Esses códigos foram discutidos com mais detalhes no trabalho 1.

```
import numpy as np
import osr_methods as osr
import line_search_methods as lsm
import ocr_methods as ocr
from timeit import default_timer as timer

def univariante(passo, dimens):
    #indice do vetor = (resto da divisao do passo pela dimensao) - 1
    #primeira posicao do vetor no python tem indice 0
    indice = passo%dimens - 1

    if (indice == -1) :
        #indice = -1 indica que se trata da ultima posicao do array
        #no pyton esse indice eh o tamanho do vetor - 1
        indice = dimens - 1
```

```
    #define a direcao canonica a ser utilizada
    ek = np.zeros(dimens)
    ek[indice] = 1

    return ek

def powell(P, P0, direcoes, passos, ciclos, dimens):
    #indice do vetor = (resto da divisao do passo pela dimensao) - 1
    #primeira posicao do vetor no python tem indice 0
    indice = passos%(dimens + 1) - 1

    if (indice == -1):
        #indice = -1 indica que se trata da ultima posicao do array
        #no pyton esse indice eh o tamanho do vetor - 1
        #direcao n + 1 do ciclo = Patual - P0
        dir = P - P0
        direcoes[dimens - 1] = dir
    elif (indice == 0):
        #indice = 0 significa que vamos usar a primeira direcao do conjunto
        #representa o inicio de um novo ciclo
        ciclos = ciclos + 1

        if (ciclos%(dimens+2) == 0):
            #se ciclo for multipl de dimens + 2, conjunto de direcoes = canonicas
            direcoes = np.eye(dimens, dtype=float)
        P0 = P.copy()
        dir = direcoes[indice].copy()

    else:
        dir = direcoes[indice].copy()
        direcoes[indice-1] = dir

    return dir, direcoes, P0, ciclos

def newtonRaphson(grad_P, hessian_f):
    return -np.linalg.inv(hessian_f).dot(grad_P)

def steepestDescent(grad):
    return -grad

def fletcherReeves(dir_last, grad, grad_last, passo):
    if passo == 1:
        grad_last = grad.copy()
        return -grad, grad_last
    else:
        beta = (np.linalg.norm(grad)/np.linalg.norm(grad_last))**2
        grad_last = grad.copy()
        return -grad + beta*dir_last, grad_last

def bfgs(P, P_last, grad, grad_last, S_last, passo, dimens):
    if (passo == 1):
        dir = -S_last.dot(grad)
    else:
        delta_x_k = P - P_last
        delta_g_k = grad - grad_last

        #para o numpy, vetor 1-D linha e vetor coluna sao a mesma coisa (nao e necessrio
                                                                transpor)
        #matrizes
        A = np.outer(delta_x_k, np.transpose(delta_x_k))
        B = S_last.dot(np.outer(delta_g_k, np.transpose(delta_x_k)))
        C = np.outer(delta_x_k, np.transpose(S_last.dot(delta_g_k)))

        #Escalares
        d = np.transpose(delta_x_k).dot(delta_g_k)
        e = np.transpose(delta_g_k).dot(S_last.dot(delta_g_k))

        S = S_last + (d + e)*A/(d**2) - (B + C)/d
        dir = -S.dot(grad)
        S_last = S.copy()
    P_last = P
    grad_last = grad
    return dir, P_last, grad_last, S_last

def osr_ctrl(P0, params, r, ctrl_num, metodo_ocr, metodo_osr):
    #controle numerico
```

```
maxiter = ctrl_num[0]
tol_conv = ctrl_num[1]
tol_search = ctrl_num[2]
line_step = ctrl_num[3]
eps = ctrl_num[4]

metodo = metodo_osr

#inicializacoes auxiliares dos metodos de OSR
passos = 0
dimens = P0.size
Pmin = P0.copy()
listPmin = []
listPmin.append(Pmin)

if metodo_ocr == 1:
    grad = ocr.grad_phi_penal(Pmin, params, r)
elif metodo_ocr == 2:
    grad = ocr.grad_phi_bar(Pmin, params, r)

norm_grad = np.linalg.norm(grad)
flag_conv = True

if (metodo == 2):
    direcoes = np.eye(dimens, dtype=float)
    ciclos = 0
    P1 = P0.copy()
elif (metodo == 5):
    #o metodo recebe a direcao anterior
    #inicializo a direcao com um vetor de zeros mas que nunca e usado
    #uso apenas para enviar como parametro na primeira iteracao do metodo, o qual
                                                    atualiza o valor de dir para a
                                                    iteracao seguinte
    dir = np.zeros((1, dimens))
    grad_last = grad.copy()
elif(metodo == 6):
    S_last = np.eye(dimens)
    grad_last = grad.copy()
    P_last = P0.copy()

#calculo do Pmin
start = timer()
while (norm_grad > tol_conv):
    if (passos == maxiter):
        flag_conv = False
        break
    passos = passos + 1
    if (metodo == 1):
        dir = osr.univariante(passos, dimens)
    elif (metodo == 2):
        dir, direcoes, P1, ciclos = osr.powell(Pmin, P1, direcoes,passos, ciclos, dimens
                                                                                    )
    elif (metodo == 3):
        dir = osr.steepestDescent(grad)
    elif (metodo == 4):
        if metodo_ocr == 1:
            hess = ocr.hess_phi_penal(Pmin, params, r)
        elif metodo_ocr == 2:
            hess = ocr.hess_phi_bar(Pmin, params,r)
        dir = osr.newtonRaphson(grad, hess)
    elif (metodo == 5):
        dir, grad_last = osr.fletcherReeves(dir, grad, grad_last, passos)
    elif (metodo == 6):
        dir, P_last, grad_last, S_last = osr.bfgs(Pmin, P_last, grad, grad_last, S_last,
                                                                    passos, dimens)
    dir = dir/np.linalg.norm(dir)
    intervalo = lsm.passo_cte(dir, Pmin, params, r, metodo_ocr, eps, line_step)
    alpha = lsm.secao_aurea(intervalo, dir, Pmin, params, r, metodo_ocr, tol_search)
    Pmin = Pmin + alpha*dir
    listPmin.append(Pmin)


    if metodo_ocr == 1:
        grad = ocr.grad_phi_penal(Pmin, params, r)
    elif metodo_ocr == 2:
        grad = ocr.grad_phi_bar(Pmin, params, r)
```

```
        norm_grad = np.linalg.norm(grad)

    end = timer()
    tempoExec = end - start

    return listPmin, passos, norm_grad, flag_conv, tempoExec
```

2.4 Busca Unidirecional

Os algoritmos dos métodos do Passo Constante e da Seção Áurea foram implementados em um arquivo denominado line_search_methods.py. Códigos discutidos no trabalho 1. Pequena adaptação realizada para trabalhar com as funções ϕ dos métodos OCR.

```
import ocr_methods as ocr
import numpy as np

def passo_cte(direcao, P0, params, r, metodo_ocr, eps = 1E-8, step = 0.01):
    #line search pelo metodo do passo constante

    #define o sentido correto de busca
    if metodo_ocr == 1:
        f1 = ocr.phi_penal(P0 - eps*(direcao/np.linalg.norm(direcao)), params, r)
        f2 = ocr.phi_penal(P0 + eps*(direcao/np.linalg.norm(direcao)), params, r)
    elif metodo_ocr == 2:
        f1 = ocr.phi_bar(P0 - eps*(direcao/np.linalg.norm(direcao)), params, r)
        f2 = ocr.phi_bar(P0 + eps*(direcao/np.linalg.norm(direcao)), params, r)

    if (f1 > f2):
        sentido_busca = direcao.copy()
        flag = 0
    else:
        sentido_busca = -direcao.copy()
        flag = 1

    P = P0.copy()
    P_next = P + step*sentido_busca
    alpha = 0

    if metodo_ocr == 1:
        f1 = ocr.phi_penal(P, params, r)
        f2 = ocr.phi_penal(P_next, params, r)
    elif metodo_ocr == 2:
        f1 = ocr.phi_bar(P, params, r)
        f2 = ocr.phi_bar(P_next, params, r)

    while (f1 > f2):
        alpha = alpha + step
        P = P0 + alpha*sentido_busca
        P_next = P0 + (alpha+step)*sentido_busca

        if metodo_ocr == 1:
            f1 = ocr.phi_penal(P, params, r)
            f2 = ocr.phi_penal(P_next, params, r)
            f_eps = ocr.phi_penal(P - eps*(sentido_busca/np.linalg.norm(sentido_busca)),
                                                        params, r)
        elif metodo_ocr == 2:
            f1 = ocr.phi_bar(P, params, r)
            f2 = ocr.phi_bar(P_next, params, r)
            f_eps = ocr.phi_bar(P - eps*(sentido_busca/np.linalg.norm(sentido_busca)),
                                                        params, r)

        if (f_eps < f1):
            alpha = alpha - step
            break

    intervalo = np.array([alpha, alpha + step])

    if(flag == 1):
        intervalo = -intervalo

    #retorna o intervalo de busca = [alpha min, alpha min + step]
    return intervalo

def secao_aurea(intervalo, direcao, P0, params, r, metodo_ocr, tol=0.00001):
```

```
    #line search pelo metodo da secao aurea

    #verifica o sentido da busca
    if(intervalo[1] < 0):
        intervalo = -intervalo
        sentido_busca = -direcao.copy()
        flag = 1
    else:
        sentido_busca = direcao.copy()
        flag = 0

    #atribui os limites superior e inferior da busca a variaveis internas do metodo
    alpha_upper = intervalo[1]
    alpha_lower = intervalo[0]
    beta = alpha_upper - alpha_lower

    #razao aurea
    Ra = (np.sqrt(5)-1)/2

    # define os pontos de analise de f com base na razao aurea
    alpha_e = alpha_lower + (1-Ra)*beta
    alpha_d = alpha_lower + Ra*beta

    #primeira iteracao avalia f nos 2 pontos selecionados pela razao aurea
    if metodo_ocr == 1:
        f1 = ocr.phi_penal(P0 + alpha_e*sentido_busca, params, r)
        f2 = ocr.phi_penal(P0 + alpha_d*sentido_busca, params, r)
    elif metodo_ocr == 2:
        f1 = ocr.phi_bar(P0 + alpha_e*sentido_busca, params, r)
        f2 = ocr.phi_bar(P0 + alpha_d*sentido_busca, params, r)

    #loop enquanto a convergencia nao for obtida
    while (beta > tol):
        if (f1 > f2):
            #caso positivo, define novo intervalo variando de alpha_e ate alpha_upper
            # e aproveita os valores anteriores de alpha_d e f2 como novos alpha_e e f1
            alpha_lower = alpha_e
            f1 = f2
            alpha_e = alpha_d

            #calcula novo alpha_d e f2=f(alpha_d)
            beta = alpha_upper - alpha_lower
            alpha_d = alpha_lower + Ra*beta

            if metodo_ocr == 1:
                f2 = ocr.phi_penal(P0 + alpha_d*sentido_busca, params, r)
            elif metodo_ocr == 2:
                f2 = ocr.phi_bar(P0 + alpha_d*sentido_busca, params, r)

        else:
            #caso negativo, define novo intervalo variando de alpha_lower ate alpha_d
            # e aproveita os valores anteriores de alpha_e e f1 como novos alpha_d e f2
            alpha_upper = alpha_d
            f2 = f1
            alpha_d = alpha_e

            #calcula novo alpha_e e f1=f(alpha_e)
            beta = alpha_upper - alpha_lower
            alpha_e = alpha_lower + (1-Ra)*beta

            if metodo_ocr == 1:
                f1 = ocr.phi_penal(P0 + alpha_e*sentido_busca, params, r)
            elif metodo_ocr == 2:
                f1 = ocr.phi_bar(P0 + alpha_e*sentido_busca, params, r)

    # calcula Pmin e alpha min apos convergencia
    alpha_med = (alpha_lower + alpha_upper)/2
    alpha_min = alpha_med

    if (flag == 1):
        alpha_min = -alpha_min

    return alpha_min
```

3 Teste da Implementação

3.1 Problema 1

$$\begin{cases} \textbf{Min} & f(x_1, x_2) = (x_1 - 2)^4 + (x_1 - 2x_2)^2 \\ \textbf{s.t.:} & x_1^2 - x_2 \leq 0 \end{cases}$$

Obs.: Adotar $r_p^0 = 1$, $\beta = 10$ e $x^0 = \{3, 2\}$ para o método de penalidade e $r_b^0 = 10$, $\beta = 0.1$ e $x^0 = \{0, 1\}$ para o método de barreira.

$\overrightarrow{\nabla} f(\overrightarrow{x}) = \{4(x_1 - 2)^3 + 2(x_1 - 2x_2), -4(x_1 - 2x_2)\}$

$H_{f_{1x1}} = 12(x_1 - 2)^2 + 2, \quad H_{f_{1x2}} = -4, \quad H_{f_{2x1}} = -4, \quad H_{f_{2x2}} = 8$

Definição da função, seu gradiente e sua hessiana, no código principal:

```
def f(x):
    return (x[0]-2)**4 + (x[0] - 2*x[1])**2

def grad_f(x):
    return np.array([4*(x[0]-2)**3 + 2*(x[0] - 2*x[1]), 2*(x[0] - 2*x[1])*(-2)])

def hess_f(x):
    hess = np.zeros((2,2), dtype=float)
    hess[0,:] = np.array([12*(x[0]-2)**2 + 2, -4.])
    hess[1,:] = np.array([-4., 8.])
    return hess
```

$c(\overrightarrow{x}) = x_1^2 - x_2$

$\overrightarrow{\nabla} c(\overrightarrow{x}) = \{2x_1, -1\}$

$H_{c_{1x1}} = 2, \quad H_{c_{1x2}} = 0, \quad H_{c_{2x1}} = 0, \quad H_{c_{2x2}} = 0$

Definição da restrição, seu gradiente e sua hessiana, no código principal:

```
def c1(x):
    return x[0]**2 - x[1]

def grad_c1(x):
    return np.array([2*x[0], -1.])

def hess_c1(x):
    hess = np.zeros((2,2), dtype=float)
    hess[0,:] = np.array([2., 0.])
    hess[1,:] = np.array([0., 0])
    return hess
```

3.1.1 Penalidade - Prob. 1

Definição do ponto inicial $x^0 = \{3, 2\}$ no código principal :

```
x = np.array([3., 2.])
```

Controle numérico e parâmetros:

- Máximas iterações na OSR : 1000
- Tolerância OSR: 10^{-6}
- Tolerância Seção Áurea: 10^{-7}
- $\Delta\alpha$: 10^{-2}
- ϵ: 10^{-10}
- Tolerância OCR: 10^{-6}
- $r_p^0 = 1$
- $\beta = 10$

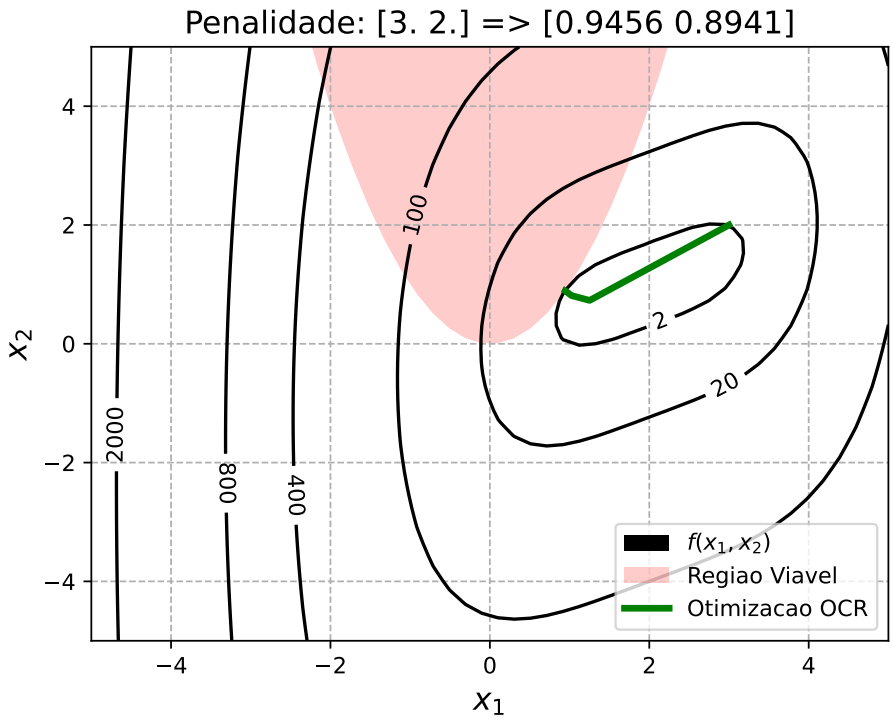

Figura 1: Curvas de nível de $f(x_1, x_2)$, restrições e otimização realizada.

Prob. 1 - Penalidade - Univariante

Iter	P_{min}	r	# Passos	Conv_OCR	Conv_OSR	t(s)
1	[1.25174114 0.7304246]	1e+00	27	3.5e-01	9.6e-07	0.028
2	[1.02501305 0.81147611]	1e+01	1000	2.9e-01	3.7e-06	0.230
3	[0.95576688 0.88122316]	1e+02	1000	5.2e-02	3.5e-06	0.189
4	[0.94659013 0.89267782]	1e+03	1000	5.6e-03	1.6e-03	0.214
5	[0.94526659 0.89319248]	1e+04	1000	5.7e-04	1.5e-02	0.190
6	[0.94513054 0.89323808]	1e+05	1000	5.7e-05	1.5e-02	0.195
7	[0.94511456 0.89323814]	1e+06	1000	5.7e-06	3.3e-02	0.180
8	[0.94511298 0.8932382]	1e+07	1000	6.2e-07	3.0e-01	0.228

Tabela 1: Resultados obtidos para o problema 1, método de penalidade, univariante para $x^0 = \{3, 2\}$

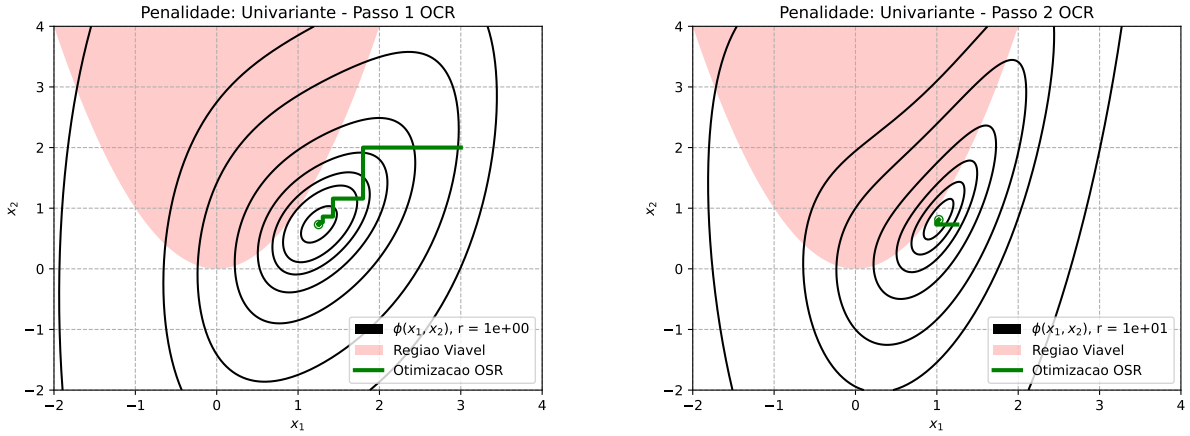

Figura 2: Exemplo com 2 primeiros passos da OCR com método OSR Univariante.

Prob. 1 - Penalidade - Powell

Iter	P_{min}	r	# Passos	Conv_OCR	Conv_OSR	t(s)
1	[1.25174105 0.73042442]	1e+00	6	3.5e-01	2.2e-07	0.027
2	[1.02501313 0.81147619]	1e+01	7	2.9e-01	9.1e-07	0.005
3	[0.95576689 0.88122318]	1e+02	53	5.2e-02	5.1e-07	0.023
4	[0.94663397 0.89276033]	1e+03	38	5.6e-03	4.8e-07	0.013
5	[0.94568845 0.89398972]	1e+04	30	5.7e-04	2.8e-07	0.009
6	[0.94559354 0.89411344]	1e+05	78	5.7e-05	3.0e-08	0.019
7	[0.94558401 0.89412573]	1e+06	1000	5.8e-06	5.2e-02	0.264
8	[0.94558315 0.89412714]	1e+07	1000	5.9e-07	1.4e-01	0.217

Tabela 2: Resultados obtidos para o problema 1, método de penalidade, powell para $x^0 = \{3, 2\}$

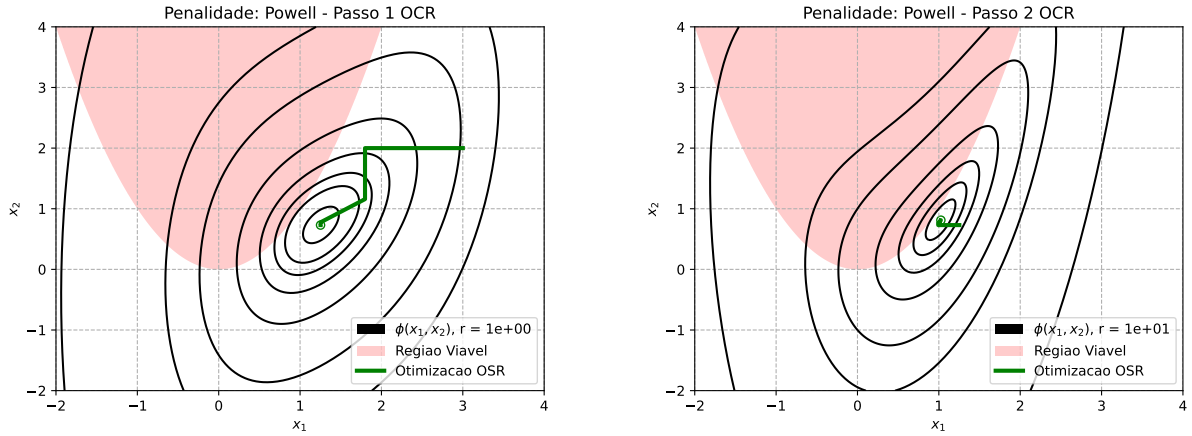

Figura 3: Exemplo com 2 primeiros passos da OCR com método OSR Powell.

Prob. 1 - Penalidade - Steepest Descent

Iter	P_{min}	r	# Passos	Conv_OCR	Conv_OSR	t(s)
1	[1.2517411 0.73042453]	1e+00	25	3.5e-01	6.2e-07	0.025
2	[1.02501316 0.81147628]	1e+01	1000	2.9e-01	1.6e-06	0.254
3	[0.9557669 0.88122324]	1e+02	1000	5.2e-02	1.2e-05	0.248
4	[0.94663395 0.89276024]	1e+03	1000	5.6e-03	1.1e-04	0.210
5	[0.94568838 0.89398961]	1e+04	1000	5.7e-04	1.3e-04	0.250
6	[0.94556416 0.89405783]	1e+05	1000	5.7e-05	1.1e-02	0.203
7	[0.94555192 0.89406498]	1e+06	1000	6.0e-06	1.8e-01	0.239
8	[0.94555068 0.89406568]	1e+07	1000	8.9e-07	1.8e+00	0.212

Tabela 3: Resultados obtidos para o problema 1, método de penalidade, steepest descent para $x^0 = \{3, 2\}$

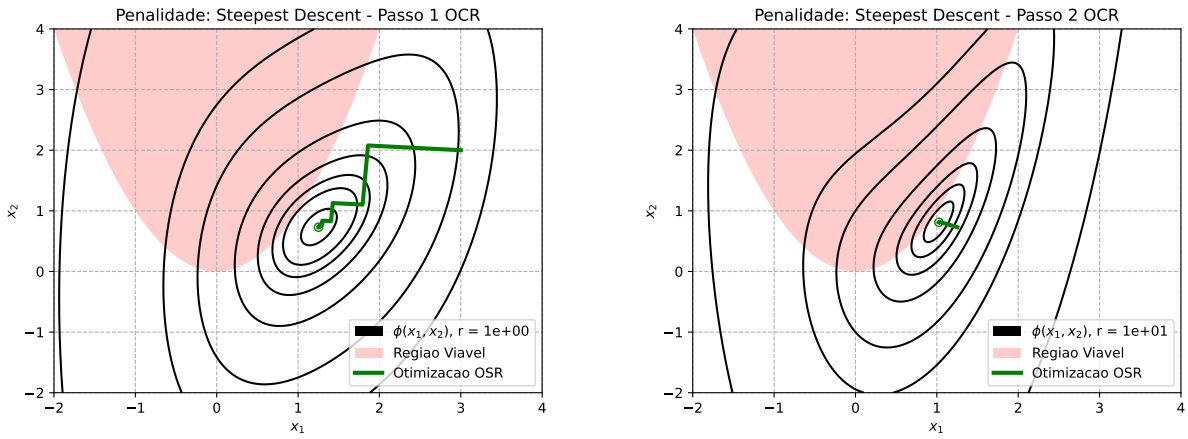

Figura 4: Exemplo com 2 primeiros passos da OCR com método OSR Steepest Descent.

Prob. 1 - Penalidade - Newton-Raphson

Iter	P_{min}	**r**	**# Passos**	**Conv_OCR**	**Conv_OSR**	**t(s)**
1	[1.25174109 0.73042445]	1e+00	3	3.5e-01	2.7e-07	0.035
2	[1.02501318 0.81147627]	1e+01	4	2.9e-01	8.9e-08	0.009
3	[0.95576692 0.88122322]	1e+02	4	5.2e-02	1.0e-07	0.008
4	[0.94663397 0.89276032]	1e+03	1000	5.6e-03	4.6e-06	0.668
5	[0.94568841 0.89398971]	1e+04	1000	5.7e-04	1.3e-03	0.422
6	[0.94559354 0.89411344]	1e+05	1000	5.7e-05	5.1e-04	0.502
7	[0.94558405 0.89412582]	1e+06	1000	5.7e-06	2.0e-04	0.526
8	[0.9455831 0.89412707]	1e+07	1000	5.7e-07	4.2e-03	0.401

Tabela 4: Resultados obtidos para o problema 1, método de penalidade, Newton-Raphson para $x^0 = \{3, 2\}$

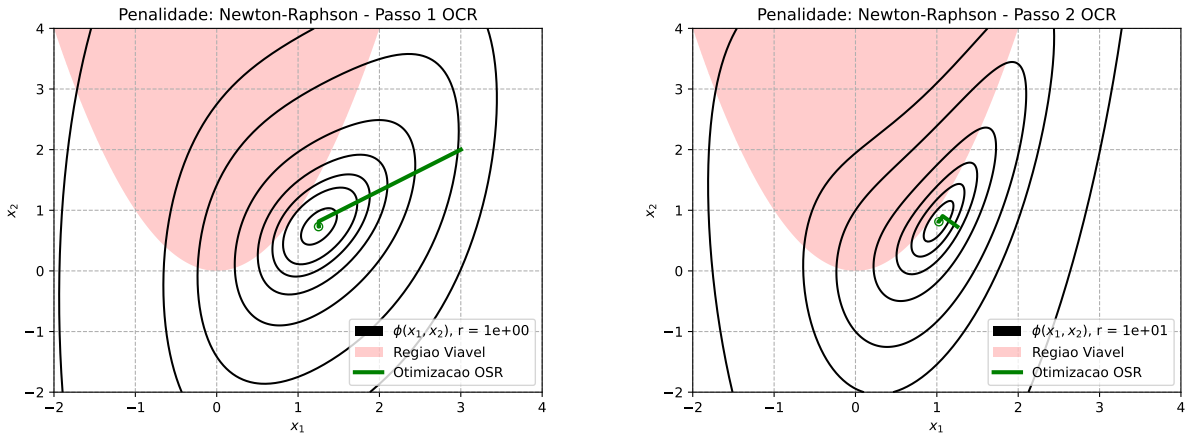

Figura 5: Exemplo com 2 primeiros passos da OCR com método OSR Newton-Raphson.

Prob. 1 - Penalidade - Fletcher-Reeves

Iter	P_{min}	r	# Passos	Conv_OCR	Conv_OSR	t(s)
1	[1.24978283 0.72770155]	1e+00	1000	3.5e-01	1.9e-02	0.338
2	[1.0247375 0.81081343]	1e+01	1000	2.9e-01	5.2e-03	0.305
3	[0.95598843 0.88169493]	1e+02	1000	5.2e-02	1.0e-02	0.253
4	[0.94673281 0.89294715]	1e+03	1000	5.6e-03	2.2e-03	0.292
5	[0.94569677 0.89400545]	1e+04	1000	5.7e-04	2.9e-04	0.259
6	[0.94557152 0.89407169]	1e+05	1000	5.7e-05	2.0e-02	0.250
7	[0.94555891 0.89407834]	1e+06	1000	5.5e-06	1.4e-01	0.235
8	[0.94555764 0.89407898]	1e+07	1000	3.7e-07	1.4e+00	0.244

Tabela 5: Resultados obtidos para o problema 1, método de penalidade, Flecther-Reeves para $x^0 = \{3, 2\}$

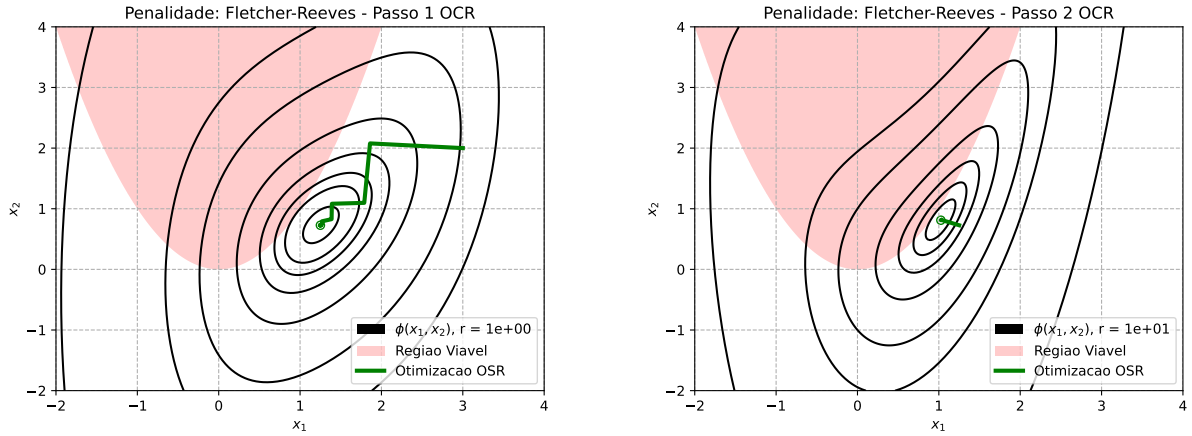

Figura 6: Exemplo com 2 primeiros passos da OCR com método OSR Fletcher-Reeves

Prob. 1 - Penalidade - BFGS

Iter	P_{min}	r	# Passos	Conv_OCR	Conv_OSR	t(s)
1	[1.25174106 0.73042443]	1e+00	6	3.5e-01	4.5e-08	0.021
2	[1.02501318 0.81147628]	1e+01	4	2.9e-01	5.3e-08	0.003
3	[0.95576696 0.88122325]	1e+02	1000	5.2e-02	8.7e-06	0.278
4	[0.94663396 0.89276031]	1e+03	4	5.6e-03	6.6e-07	0.001
5	[0.94568843 0.8939897]	1e+04	1000	5.7e-04	6.2e-06	0.244
6	[0.94559353 0.89411343]	1e+05	1000	5.7e-05	3.1e-03	0.259
7	[0.94558406 0.89412585]	1e+06	1000	5.7e-06	1.0e-03	0.276
8	[0.94558308 0.89412702]	1e+07	1000	5.7e-07	7.1e-03	0.271

Tabela 6: Resultados obtidos para o problema 1, método de penalidade, BFGS para $x^0 = \{3, 2\}$

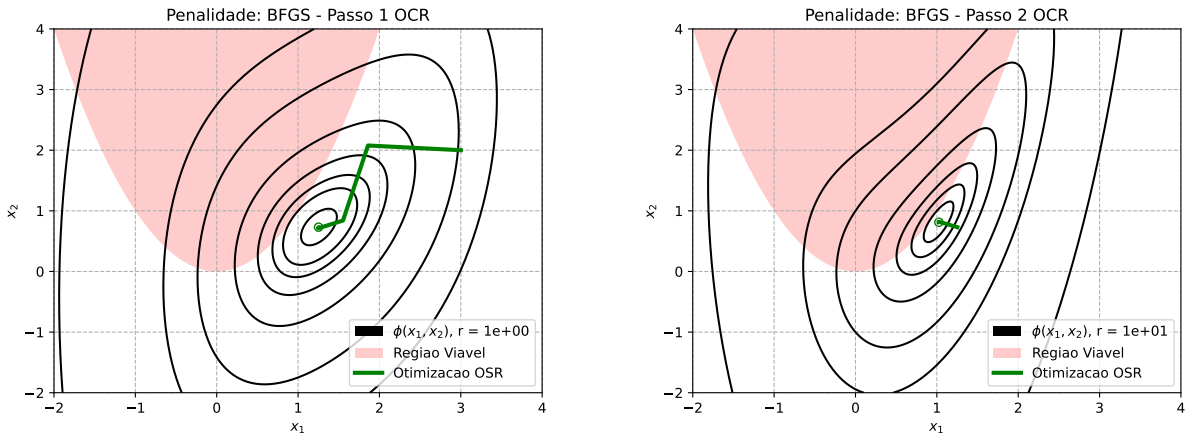

Figura 7: Exemplo com 2 primeiros passos da OCR com método OSR BFGS.

3.1.2 Barreira - Prob. 1

Definição do ponto inicial $x^0 = \{0, 1\}$ no código principal :

```
x = np.array([0., 1.])
```

Controle numérico e parâmetros:

- Máximas iterações na OSR : 1000
- Tolerância OSR: 10^{-6}
- Tolerância Seção Áurea: 10^{-7}
- $\Delta\alpha$: 10^{-2}
- ϵ: 10^{-10}
- Tolerância OCR: 10^{-6}
- $r_b^0 = 10$
- $\beta = 0.1$

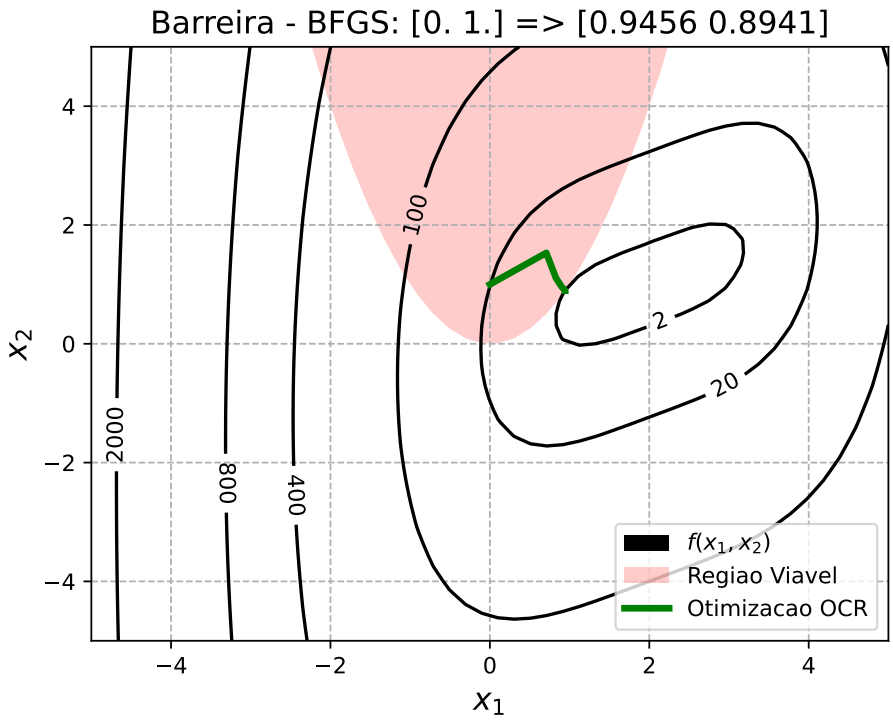

Figura 8: Curvas de nível de $f(x_1, x_2)$, restrições e otimização realizada.

Prob. 1 - Barreira - Univariante

Iter	P_{min}	r	$\Delta\alpha$	# Passos	Conv_OCR	Conv_OSR	t(s)
1	[0.70794442 1.53149922]	1e+01	1e-02	208	9.7e+00	9.3e-07	0.051
2	[0.82820088 1.10979838]	1e+00	1e-02	76	2.4e+00	4.2e-07	0.020
3	[0.89886443 0.96384102]	1e-01	1e-02	144	6.4e-01	8.4e-07	0.040
4	[0.92935334 0.91630707]	1e-02	1e-02	1000	1.9e-01	5.4e-06	0.357
5	[0.94027885 0.90115413]	1e-03	1e-02	1000	5.9e-02	1.2e-05	0.181
6	[0.94393558 0.89644099]	1e-04	1e-03	1000	1.8e-02	1.7e-03	0.137
7	[0.94528564 0.89528469]	1e-05	1e-03	1000	5.8e-03	8.6e-03	0.135
8	[0.94574387 0.89497567]	1e-06	1e-04	1000	1.8e-03	1.2e-02	0.115
9	[0.94589637 0.89489206]	1e-07	1e-04	1000	5.8e-04	1.3e-02	0.102
10	[0.94593827 0.89485365]	1e-08	1e-05	1000	1.8e-04	1.5e-02	0.098
11	[0.94595766 0.89485312]	1e-09	1e-05	1000	5.8e-05	8.6e-03	0.086
12	[0.94596395 0.89485321]	1e-10	1e-06	1000	1.8e-05	7.2e-02	0.065
13	[0.94596577 0.89485297]	1e-11	1e-06	1000	5.7e-06	1.6e-01	0.082
14	[0.94596639 0.89485294]	1e-12	1e-07	1000	1.9e-06	3.3e-01	0.063
15	[0.94596659 0.89485294]	1e-13	1e-07	1000	6.5e-07	1.8e+00	0.064

Tabela 7: Resultados obtidos para o problema 1, método de barreira, univariante para $x^0 = \{0, 1\}$

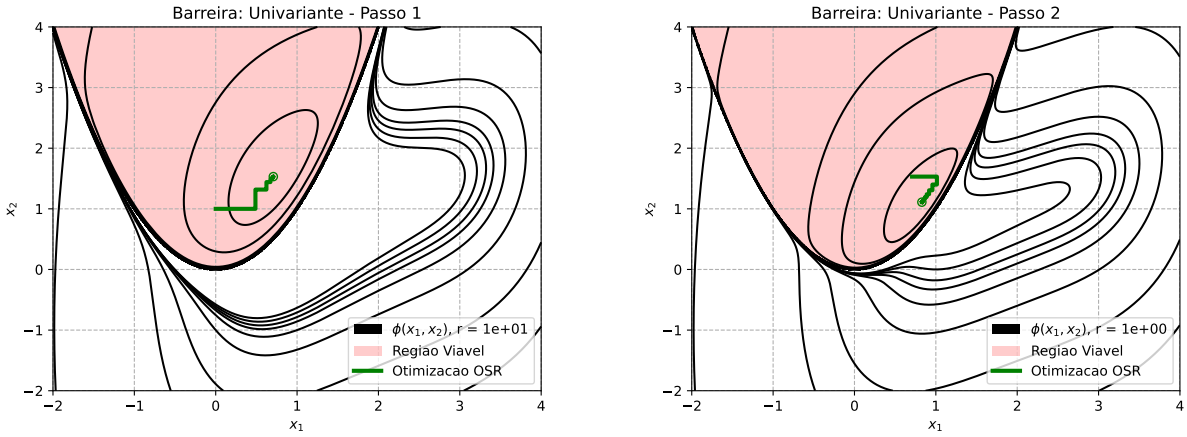

Figura 9: Exemplo com 2 primeiros passos da OCR com método OSR Univariante.

Prob. 1 - Barreira - Powell

Iter	P_{min}	r	$\Delta\alpha$	# Passos	Conv_OCR	Conv_OSR	t(s)
1	[0.70794439 1.53149919]	1e+01	1e-02	8	9.7e+00	5.6e-07	0.010
2	[0.82820086 1.10979833]	1e+00	1e-02	25	2.4e+00	8.4e-07	0.009
3	[0.89886443 0.963841]	1e-01	1e-02	20	6.4e-01	2.8e-07	0.006
4	[0.92935322 0.91630682]	1e-02	1e-02	9	1.9e-01	9.3e-07	0.002
5	[0.94027827 0.90115302]	1e-03	1e-02	12	5.9e-02	8.0e-07	0.003
6	[0.9438872 0.8963501]	1e-04	1e-03	44	1.8e-02	4.0e-07	0.013
7	[0.9450449 0.89483025]	1e-05	1e-03	1000	5.8e-03	5.1e-04	0.367
8	[0.94541263 0.89434951]	1e-06	1e-04	54	1.8e-03	5.1e-07	0.007
9	[0.94552911 0.89419752]	1e-07	1e-04	24	5.8e-04	4.1e-07	0.005
10	[0.94556594 0.89414941]	1e-08	1e-05	393	1.8e-04	4.0e-07	0.065
11	[0.94557753 0.89413409]	1e-09	1e-05	1000	5.8e-05	4.5e-03	0.154
12	[0.94558129 0.89412942]	1e-10	1e-06	30	1.8e-05	4.7e-07	0.002
13	[0.94558246 0.89412794]	1e-11	1e-06	1000	5.7e-06	1.9e-01	0.140
14	[0.94558283 0.89412746]	1e-12	1e-07	1000	1.8e-06	4.0e-01	0.111
15	[0.94558288 0.89412715]	1e-13	1e-07	1000	5.7e-07	3.3e-01	0.119

Tabela 8: Resultados obtidos para o problema 1, método de barreira, Powell para $x^0 = \{0, 1\}$

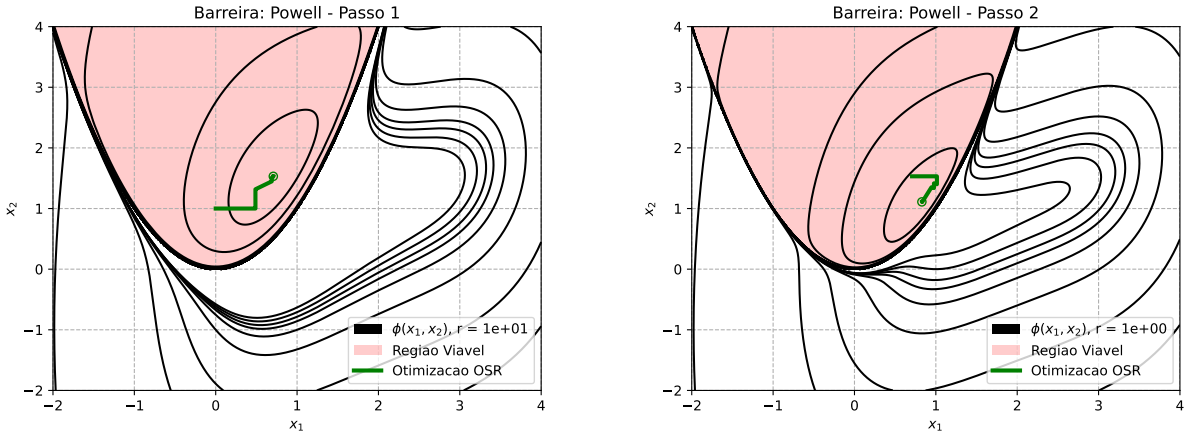

Figura 10: Exemplo com 2 primeiros passos da OCR com método OSR Powell.

Prob. 1 - Barreira - Steepest Descent

Iter	P_{min}	r	$\Delta\alpha$	# Passos	Conv_OCR	Conv_OSR	t(s)
1	[0.70794438 1.53149917]	1e+01	1e-02	52	9.7e+00	4.3e-07	0.033
2	[0.8282009 1.10979841]	1e+00	1e-02	112	2.4e+00	3.5e-07	0.035
3	[0.89886442 0.96384104]	1e-01	1e-02	1000	6.4e-01	6.9e-06	0.219
4	[0.92935325 0.91630683]	1e-02	1e-02	1000	1.9e-01	1.5e-05	0.325
5	[0.94027832 0.90115304]	1e-03	1e-02	1000	5.9e-02	6.6e-05	0.300
6	[0.94388751 0.89635066]	1e-04	1e-03	1000	1.8e-02	9.5e-05	0.172
7	[0.94504565 0.8948317]	1e-05	1e-03	1000	5.8e-03	3.0e-04	0.145
8	[0.94549366 0.89450285]	1e-06	1e-04	1000	1.8e-03	2.3e-03	0.160
9	[0.94563767 0.89440286]	1e-07	1e-04	1000	5.8e-04	3.3e-03	0.193
10	[0.94568309 0.89437093]	1e-08	1e-05	1000	1.8e-04	1.2e-02	0.374
11	[0.94569739 0.89436074]	1e-09	1e-05	1000	5.8e-05	3.6e-02	0.307
12	[0.94570181 0.89435729]	1e-10	1e-06	1000	1.9e-05	1.7e-01	0.150
13	[0.94570312 0.89435619]	1e-11	1e-06	1000	5.6e-06	6.0e-01	0.098
14	[0.94570357 0.89435581]	1e-12	1e-07	1000	1.8e-06	3.1e-01	0.062
15	[0.94570374 0.89435569]	1e-13	1e-07	1000	7.8e-07	5.6e+00	0.063

Tabela 9: Resultados obtidos para o problema 1, método de barreira, Steepest Descent para $x^0 = \{0, 1\}$

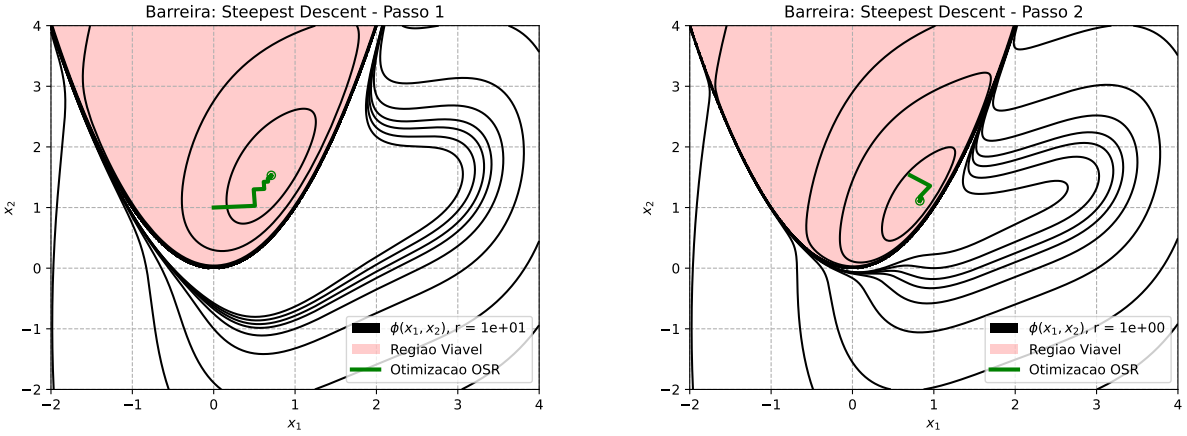

Figura 11: Exemplo com 2 primeiros passos da OCR com método OSR Steepest Descent.

Prob. 1 - Barreira - Newton-Raphson

Iter	P_{min}	r	$\Delta\alpha$	# Passos	Conv_OCR	Conv_OSR	t(s)
1	[0.7079444 1.5314992]	1e+01	1e-02	4	9.7e+00	1.2e-07	0.014
2	[0.8282009 1.10979841]	1e+00	1e-02	6	2.4e+00	1.0e-07	0.011
3	[0.89886444 0.96384105]	1e-01	1e-02	1000	6.4e-01	3.0e-06	0.395
4	[0.9293532 0.91630686]	1e-02	1e-02	1000	1.9e-01	2.3e-05	0.344
5	[0.94027832 0.90115304]	1e-03	1e-02	1000	5.9e-02	6.9e-05	0.389
6	[0.94388719 0.89635012]	1e-04	1e-03	1000	1.8e-02	1.4e-04	0.295
7	[0.9450449 0.89483025]	1e-05	1e-03	1000	5.8e-03	4.7e-04	0.327
8	[0.94541264 0.89434953]	1e-06	1e-03	23	1.8e-03	5.0e-07	0.004
9	[0.9455291 0.8941975]	1e-07	1e-04	1000	5.8e-04	5.7e-05	0.319
10	[0.94556593 0.89414939]	1e-08	1e-04	1000	1.8e-04	1.8e-04	0.330
11	[0.94557761 0.89413424]	1e-09	1e-05	1000	5.8e-05	2.1e-05	0.241
12	[0.94558128 0.8941294]	1e-10	1e-05	1000	1.8e-05	3.8e-04	0.276
13	[0.94558246 0.89412791]	1e-11	1e-06	1000	5.8e-06	1.8e-03	0.192
14	[0.94558281 0.8941274]	1e-12	1e-06	1000	1.8e-06	1.1e-02	0.173
15	[0.94558294 0.89412727]	1e-13	1e-07	1000	5.8e-07	1.5e-03	0.149

Tabela 10: Resultados obtidos para o problema 1, método de barreira, Newton-Raphson para $x^0 = \{0, 1\}$

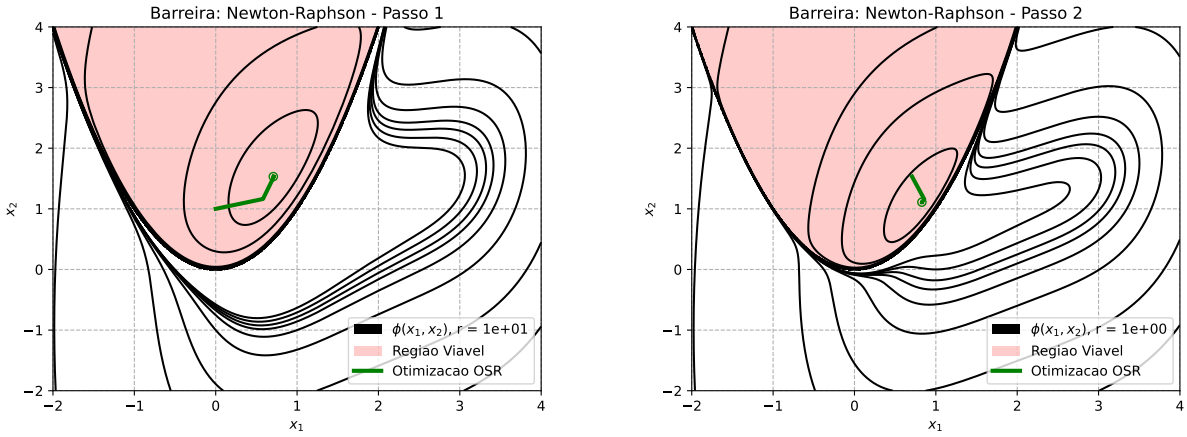

Figura 12: Exemplo com 2 primeiros passos da OCR com método OSR Newton-Raphson.

Prob. 1 - Barreira - Fletcher-Reeves

Iter	P_{min}	r	$\Delta\alpha$	# Passos	Conv_OCR	Conv_OSR	t(s)
1	[0.7083677 1.53228399]	1e+01	1e-02	1000	9.7e+00	1.2e-02	0.272
2	[0.82771558 1.10882661]	1e+00	1e-02	1000	2.4e+00	1.1e-02	0.255
3	[0.89923489 0.96456733]	1e-01	1e-02	1000	6.4e-01	7.5e-03	0.164
4	[0.92959411 0.9167657]	1e-02	1e-02	1000	1.9e-01	4.3e-03	0.174
5	[0.94042115 0.90142408]	1e-03	1e-02	1000	5.9e-02	2.6e-03	0.192
6	[0.94385431 0.89628806]	1e-04	1e-03	1000	1.8e-02	6.6e-04	0.157
7	[0.94502008 0.89478343]	1e-05	1e-03	1000	5.8e-03	6.4e-04	0.199
8	[0.94541561 0.8943552]	1e-06	1e-04	1000	1.8e-03	1.5e-03	0.132
9	[0.94556948 0.89427382]	1e-07	1e-04	1000	5.8e-04	4.4e-03	0.171
10	[0.94561808 0.89424809]	1e-08	1e-05	1000	1.8e-04	1.9e-02	0.123
11	[0.94563354 0.89423997]	1e-09	1e-05	1000	5.8e-05	4.8e-02	0.137
12	[0.94563835 0.89423743]	1e-10	1e-06	1000	1.8e-05	2.3e-01	0.073
13	[0.94563992 0.89423665]	1e-11	1e-06	1000	5.6e-06	5.8e-01	0.127
14	[0.94564043 0.89423638]	1e-12	1e-07	1000	1.8e-06	2.2e-01	0.085
15	[0.94564061 0.89423629]	1e-13	1e-07	1000	8.0e-07	6.4e+00	0.101

Tabela 11: Resultados obtidos para o problema 1, método de barreira, Fletcher-Reeves para $x^0 = \{0, 1\}$

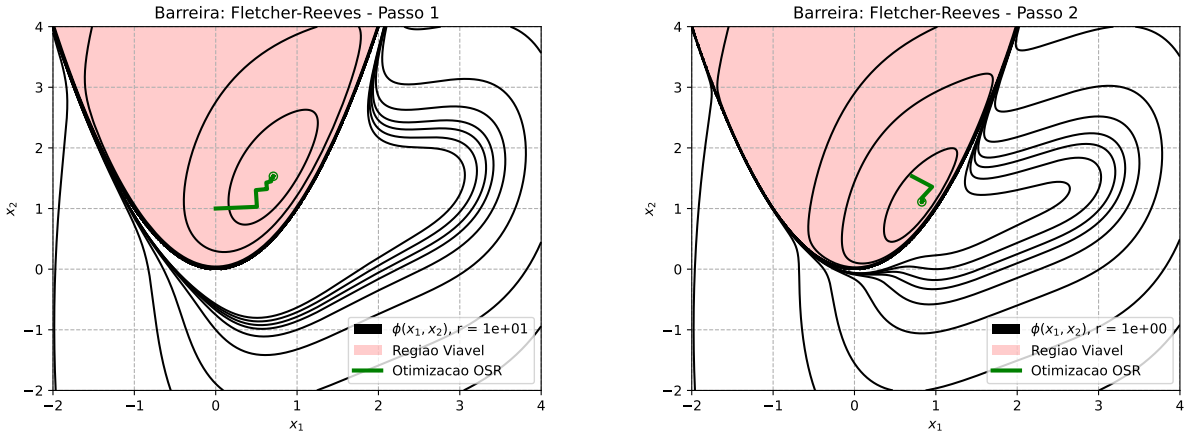

Figura 13: Exemplo com 2 primeiros passos da OCR com método OSR Fletcher-Reeves

Prob. 1 - Barreira - BFGS

Iter	P_{min}	r	$\Delta\alpha$	# Passos	Conv_OCR	Conv_OSR	t(s)
1	[0.7079444 1.5314992]	1e+01	1e-02	11	9.7e+00	1.4e-07	0.017
2	[0.82820084 1.10979841]	1e+00	1e-02	1000	2.4e+00	7.2e-06	0.368
3	[0.89886439 0.96384104]	1e-01	1e-02	1000	6.4e-01	1.3e-05	0.346
4	[0.92935323 0.91630685]	1e-02	1e-02	4	1.9e-01	7.5e-07	0.003
5	[0.94027829 0.90115306]	1e-03	1e-02	4	5.9e-02	3.8e-07	0.001
6	[0.94388719 0.89635009]	1e-04	1e-02	1000	1.8e-02	7.6e-06	0.301
7	[0.94504488 0.89483027]	1e-05	1e-03	1000	5.8e-03	1.4e-05	0.358
8	[0.94541264 0.89434954]	1e-06	1e-04	1000	1.8e-03	3.6e-04	0.152
9	[0.94552912 0.89419753]	1e-07	1e-04	1000	5.8e-04	3.1e-04	0.290
10	[0.94556597 0.89414946]	1e-08	1e-04	1000	1.8e-04	1.5e-03	0.195
11	[0.94557758 0.89413418]	1e-09	1e-05	1000	5.8e-05	3.1e-04	0.194
12	[0.94558126 0.89412937]	1e-10	1e-05	1000	1.8e-05	3.6e-04	0.211
13	[0.94558245 0.89412789]	1e-11	1e-06	1000	5.8e-06	3.7e-04	0.206
14	[0.94558281 0.8941274]	1e-12	1e-06	1000	1.8e-06	6.4e-04	0.190
15	[0.94558295 0.89412729]	1e-13	1e-07	1000	5.8e-07	2.3e-03	0.151

Tabela 12: Resultados obtidos para o problema 1, método de barreira, BFGS para $x^0 = \{0, 1\}$

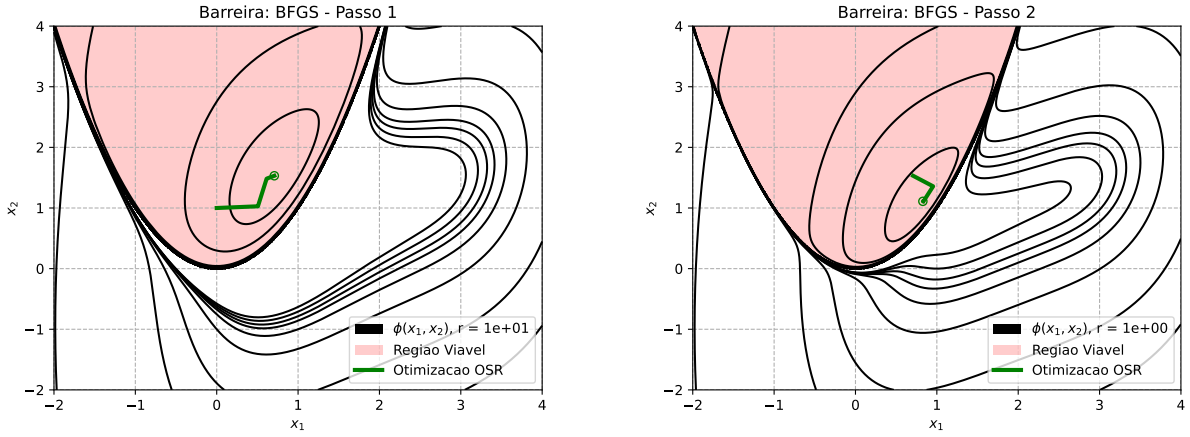

Figura 14: Exemplo com 2 primeiros passos da OCR com método OSR BFGS.