TÉCNICAS DE MODELAMENTO NUMÉRICO APLICADO A UM RESERVATÓRIO DE PETRÓLEO

Prof.: Nivaldo

Alunos: Bismarck Gomes Souza Júnior

Fernando Vizeu Santos

FORMULAÇÃO DO PROBLEMA

O presente trabalho pretende modelar um reservatório utilizando uma malha:

$$(x, y) = \{(x, y) \in \mathbb{R}^2 \mid (x, y) \in (0, 1) \times (0, 1)\} \qquad (1)$$

Utilizando técnicas de modelamento numérico, pretende-se determinar as pressões nos blocos da malha através da solução da equação de Poisson abaixo que representa o fluxo de um fluido em regime permanente em um reservatório.

1. Equação de Poisson

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = -q(x, y) \qquad (2)$$

Onde $p = p(x, y)$ representa a pressão nos pontos de coordenadas (x, y) e $q(x, y)$, a vazão, definida por:

$$q(x, y) = \frac{2\pi}{\ln\left(\frac{r_e}{r_w}\right)}\left(p_{bh} - p(x, y)\right) \qquad (3)$$

Tal que p_{bh} é a pressão no poço (injetor ou produtor) e será definido por:

$$p_{bh} = \begin{cases} 1, & (x, y) = (0.2, 0.2) \\ -1, & (x, y) = (0.7, 0.7) \end{cases} \qquad (4)$$

Para o raio do poço (r_w), usar-se-á a definição de Peaceman (...), dada por:

$$r_w = 0.2\Delta x \qquad (5)$$

Como a malha é quadrada de lado 1, o raio externo do reservatório (r_e) será aproximada pela metade da diagonal do quadrado:

$$r_e = \frac{\sqrt{2}}{2} \qquad (6)$$

2. Condições de Contorno

$$\begin{cases} \frac{\partial p}{\partial x} = 0, & y = 0 \; ou \; y = 1 \\ \frac{\partial p}{\partial y} = 0, & x = 0 \; ou \; x = 1 \end{cases} \qquad (7)$$

DISCRETIZAÇÃO DO PROBLEMA

1. Equação de Poisson discretizada

Da expansão de Taylor, temos as seguintes aproximações:

$$\frac{\partial^2 p}{\partial x^2} = \frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{\Delta x^2} \quad e \quad \frac{\partial^2 p}{\partial y^2} = \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta y^2} \qquad (8)$$

Tal que:

$$p_{i,j} = p(x,y) \qquad \begin{matrix} p_{i+1,j} = p(x+\Delta x, y) & p_{i-1,j} = p(x-\Delta x, y) \\ p_{i,j+1} = p(x, y+\Delta \mathrm{y}) & p_{i,j-1} = p(x, y-\Delta y) \end{matrix} \qquad (9)$$

Considerando a malha quadrada regularmente espaçada ($\Delta x = \Delta y = \mathrm{d}$) e substituindo (8) em (2), temos:

$$p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1} - 4p_{i,j} = \frac{2\pi d^2}{\ln\left(\frac{r_e}{r_w}\right)}\left(p_{bh_{i,j}} - p_{i,j}\right) \qquad (10)$$

Assim, obtemos a equação discretizada dada por:

$$p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1} + \left(\frac{2\pi d^2}{\ln\left(\frac{r_e}{r_w}\right)} - 4\right)p_{i,j} = \frac{2\pi d^2}{\ln\left(\frac{r_e}{r_w}\right)}\, p_{bh_{i,j}} \qquad (11)$$

De (1), sabemos que $0 \leq x \leq 1$ e $0 \leq y \leq 1$ e considerando-se que a malha seja 10x10, temos que:

$$d = \Delta x = \Delta y = \frac{1}{10} = 0.1 \qquad (12)$$

Desse modo, substituindo (5), (6) e (12) em (11), temos:

$$p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1} + \left(\frac{0.02\pi}{\ln\left(25\sqrt{2}\right)} - 4\right)p_{i,j} = \frac{0.02\pi}{\ln\left(25\sqrt{2}\right)}\, p_{bh_{i,j}} \qquad (13)$$

2. Condições de Contorno

Da expansão de Taylor e utilizando as mesmas denotações em (9), temos as seguintes aproximações:

$$\frac{\partial p}{\partial x} = \frac{p_{i+1,j} - p_{i-1,j}}{2\Delta x} \quad e \quad \frac{\partial p}{\partial y} = \frac{p_{i,j+1} - p_{i,j-1}}{2\Delta y} \qquad (14)$$

Substituindo (14) em (7), temos que:

$$\begin{cases} p_{1,j} = p_{-1,j}\,, & y = 0 \; ou \; y = 1 \\ p_{i,-1} = p_{i,1}\,, & x = 0 \; ou \; x = 1 \end{cases} \qquad (15)$$

CONSTRUÇÃO DO SISTEMA

Isolando-se um bloco da malha, nota-se que a pressão no mesmo depende das pressões dos seus adjacentes, conforme a Figura 1. Isso pode ser comprovado matematicamente pela equação discretizada (13).

Por isso, por isso, a matriz dos coeficientes será uma matriz penta-diagonal, pois para cada equação há cinco variáveis (o bloco analisado e os seus quatro adjacentes), conforme a Figura 2.

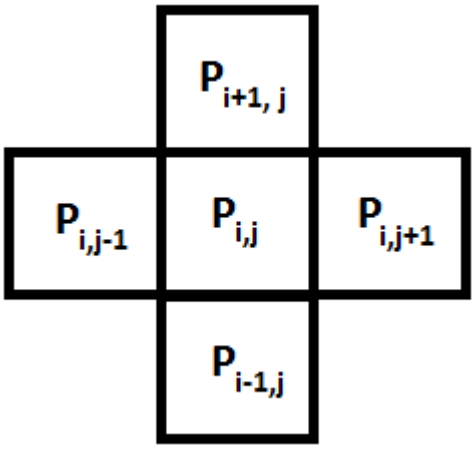


Figura 1 - Configuração dos blocos

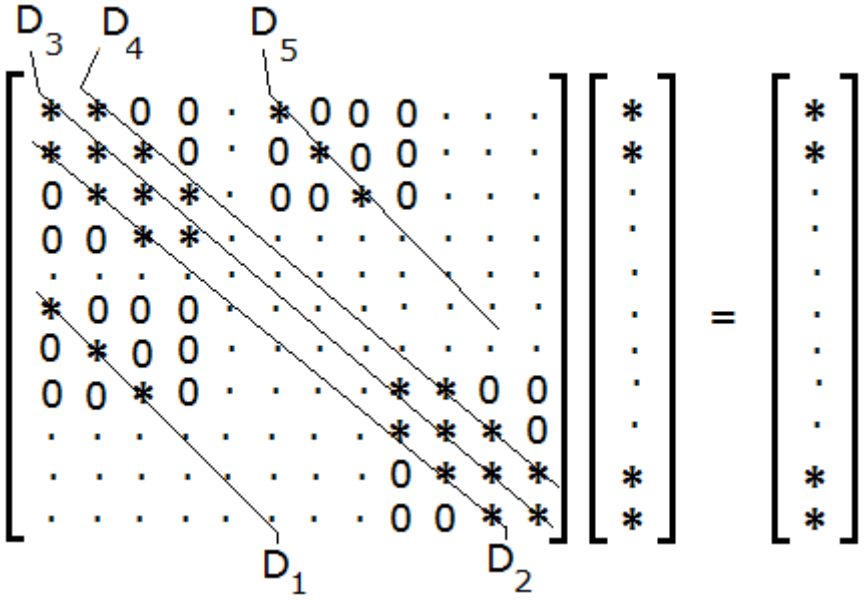


Figura 2 - Sistema Linear

Na fronteira superior (Norte), não existe o bloco $P_{i+1,j}$, por isso iremos considerar a existência de um bloco fictício cuja pressão é igual a pressão no bloco $P_{i-1,j}$. Deste modo, quando $i = 0$, temos:

$$2p_{i-1,j} + p_{i,j+1} + p_{i,j-1} + \left(\frac{0.02\pi}{\ln(25\sqrt{2})} - 4\right)p_{i,j} = \frac{0.02\pi}{\ln(25\sqrt{2})}\, p_{bh_{i,j}} \qquad \text{[NORTE]} \qquad (15.1)$$

Analogamente para as outras fronteiras, temos:

$$2p_{i+1,j} + p_{i,j+1} + p_{i,j-1} + \left(\frac{0.02\pi}{\ln(25\sqrt{2})} - 4\right)p_{i,j} = \frac{0.02\pi}{\ln(25\sqrt{2})}\, p_{bh_{i,j}} \qquad \text{[SUL]} \qquad (15.2)$$

$$p_{i+1,j} + p_{i-1,j} + 2p_{i,j-1} + \left(\frac{0.02\pi}{\ln(25\sqrt{2})} - 4\right)p_{i,j} = \frac{0.02\pi}{\ln(25\sqrt{2})}\, p_{bh_{i,j}} \qquad \text{[LESTE]} \qquad (15.3)$$

$$p_{i+1,j} + p_{i-1,j} + 2p_{i,j+1} + \left(\frac{0.02\pi}{\ln(25\sqrt{2})} - 4\right)p_{i,j} = \frac{0.02\pi}{\ln(25\sqrt{2})}\, p_{bh_{i,j}} \qquad \text{[OESTE]} \qquad (15.4)$$

Porém, para as quinas do quadrado, temos:

$$2p_{i-1,j} + 2p_{i,j+1} + \left(\frac{0.02\pi}{\ln(25\sqrt{2})} - 4\right)p_{i,j} = \frac{0.02\pi}{\ln(25\sqrt{2})}\, p_{bh_{i,j}} \qquad \text{[NOROESTE]} \qquad (15.5)$$

$$2p_{i-1,j} + 2p_{i,j-1} + \left(\frac{0.02\pi}{\ln(25\sqrt{2})} - 4\right)p_{i,j} = \frac{0.02\pi}{\ln(25\sqrt{2})}\, p_{bh_{i,j}} \qquad \text{[NORDESTE]} \qquad (15.6)$$

$$2p_{i+1,j} + 2p_{i,j+1} + \left(\frac{0.02\pi}{\ln(25\sqrt{2})} - 4\right)p_{i,j} = \frac{0.02\pi}{\ln(25\sqrt{2})}\, p_{bh_{i,j}} \qquad \text{[SULDOESTE]} \qquad (15.7)$$

$$2p_{i+1,j} + 2p_{i,j-1} + \left(\frac{0.02\pi}{\ln(25\sqrt{2})} - 4\right)p_{i,j} = \frac{0.02\pi}{\ln(25\sqrt{2})}\, p_{bh_{i,j}} \qquad \text{[SULDESTE]} \qquad (15.8)$$

Assim, os valores de D_1, D_2, D_3, D_4 e D_5 da Figura 2, são tais que:

$$D_1 = \left[\underbrace{1\ 1\ \dots\ 1}_{n^2-2n}\ \underbrace{2\ 2\ \dots\ 2}_{n}\right] \qquad (16.1)$$

$$D_2 = \left[\overbrace{2\ \underbrace{1\ 1\ \dots\ 1}_{n-2}\ 0\ \dots\ 2\ \underbrace{1\ 1\ \dots\ 1}_{n-2}\ 0}^{n-1}\ 2\ \underbrace{1\ 1\ \dots\ 1}_{n-2}\ \right] \qquad (16.2)$$

$$D_3 = \left[\underbrace{C\ C\ \dots\ C}_{n^2}\right], \qquad C = \left(\frac{0.02\pi}{\ln(25\sqrt{2})} - 4\right) \qquad (16.3)$$

$$D_4 = \left[\underbrace{1\ 1\ \dots\ 1}_{n-2}\ 2\ \overbrace{0\ \underbrace{1\ 1\ \dots\ 1}_{n-2}\ 2\ \dots\ 0\ \underbrace{1\ 1\ \dots\ 1}_{n-2}\ 2}^{n}\right] \qquad (16.4)$$

$$D_5 = \left[\underbrace{2\ 2\ \dots\ 2}_{n}\ \underbrace{1\ 1\ \dots\ 1}_{n^2-2n}\ \right] \qquad (16.5)$$

Ainda com respeito à Figura 2 e utiliazando-se a matriz $b = [b_i]_{n^2}$ é tal que:

$$b_i = \frac{2\pi d^2}{\ln\left(\frac{r_e}{r_w}\right)}\, p_{bh_{i,j}} \qquad (17)$$

ALGORITMOS DE RESOLUÇÃO DO PROBLEMA

Para a resolução da modelagem do reservatório, desenvolveram-se algoritmos para resolver o sistema linear apresentado anteriormente. A linguagem de programação utilizada foi Python devido a sua versatilidade e sua excelente biblioteca gráfica (matplotlib).

A fim de otimizar o processamento dos dados da malha, os algoritmos foram adaptados para uso específico de matrizes esparças penta-diagonais. Isso permitiu uma execução mais eficiente tanto em cálculos realizados quanto em utilização de memória.

Para solucionar o sistema linear, utilizamos quatro diferentes métodos para solucioná-lo:

- Método de Gauss-Jacobi
- Método de Gauss-Seidel
- Método de SOR (Successive Over-Relaxation)
- Eliminação Gaussian

Esses algoritmos encontram-se no Anexo I e seus dados de pressão encontrados se encontram no Anexo II.

Além disso, foram realizadas simulações com outros tamanhos de malha. Os resultados podem ser encontrados no Anexo III.

RESULTADOS

MÉTODO DE GAUSS-JACOBI

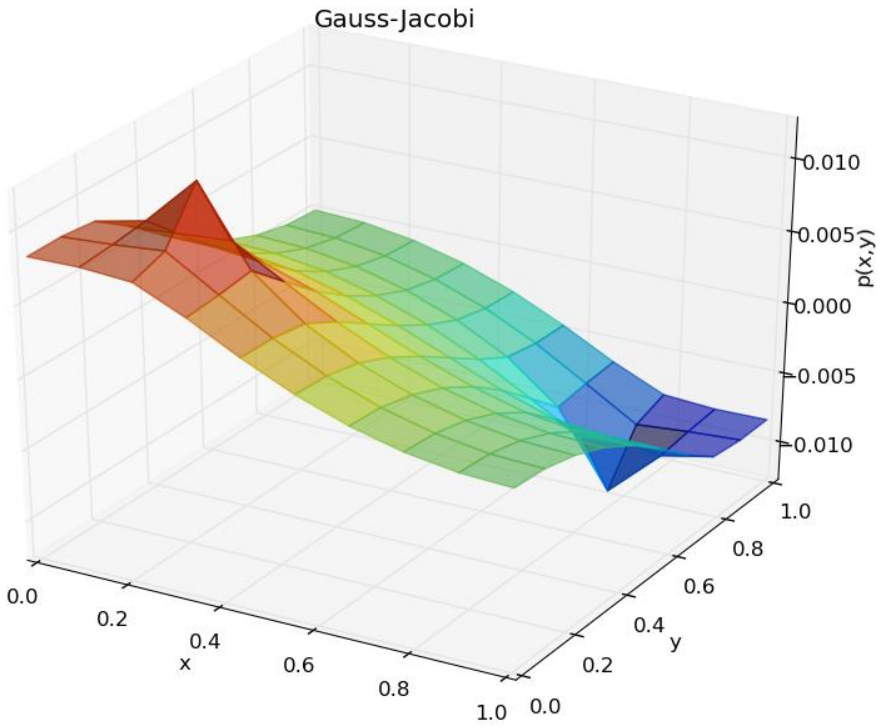


Figura 3 - Modelagem do reservatório pelo método de Gauss-Jacobi

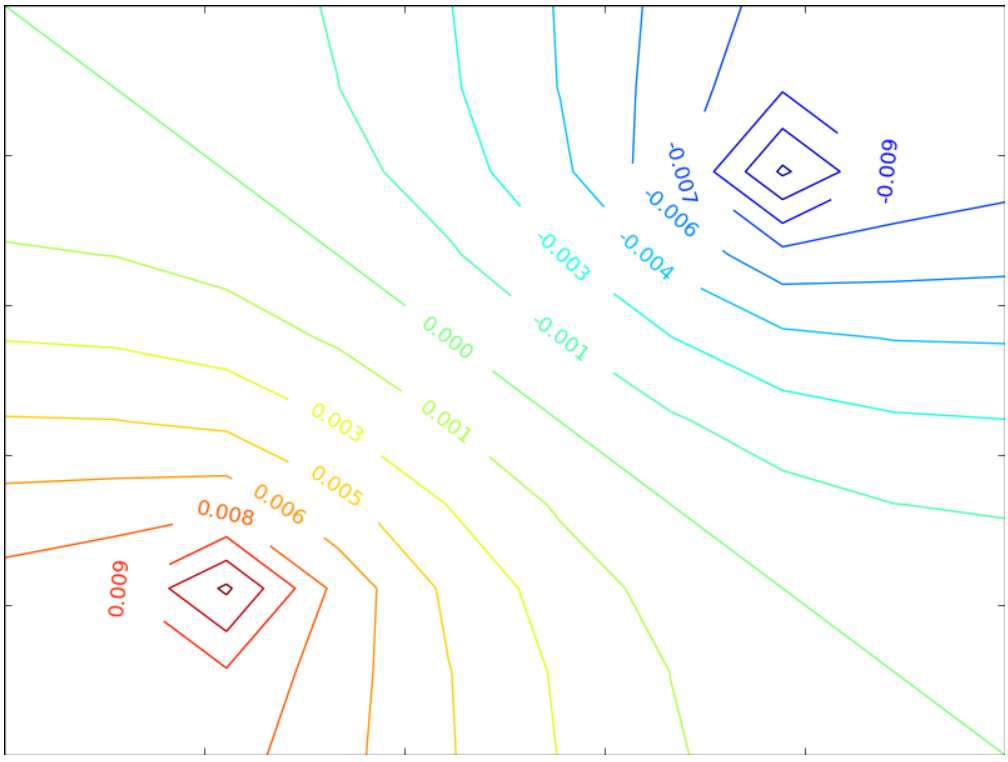


Figura 4 - Curvas de nível das pressões utilizando o método de Gauss-Jacobi

MÉTODO DE GAUSS-SEIDEL

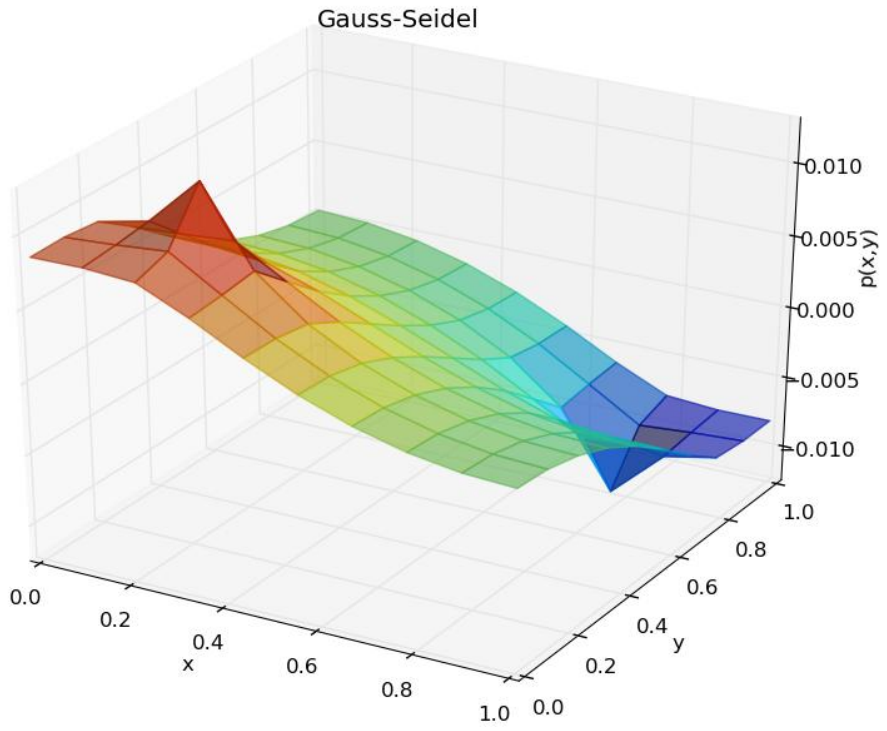


Figura 5 - Modelagem do reservatório pelo método de Gauss-Jacobi

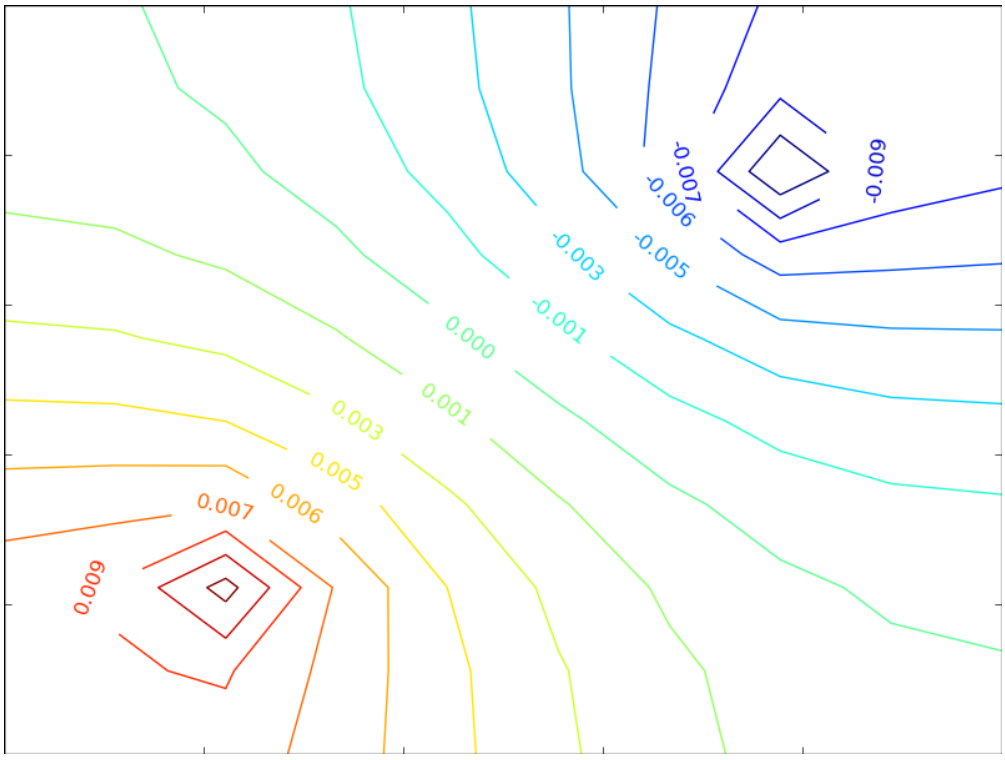


Figura 6 – Curvas de nível das pressões utilizando o método de Gauss-Jacobi

MÉTODO DE SOR (SUCCESSIVE OVER-RELAXATION)

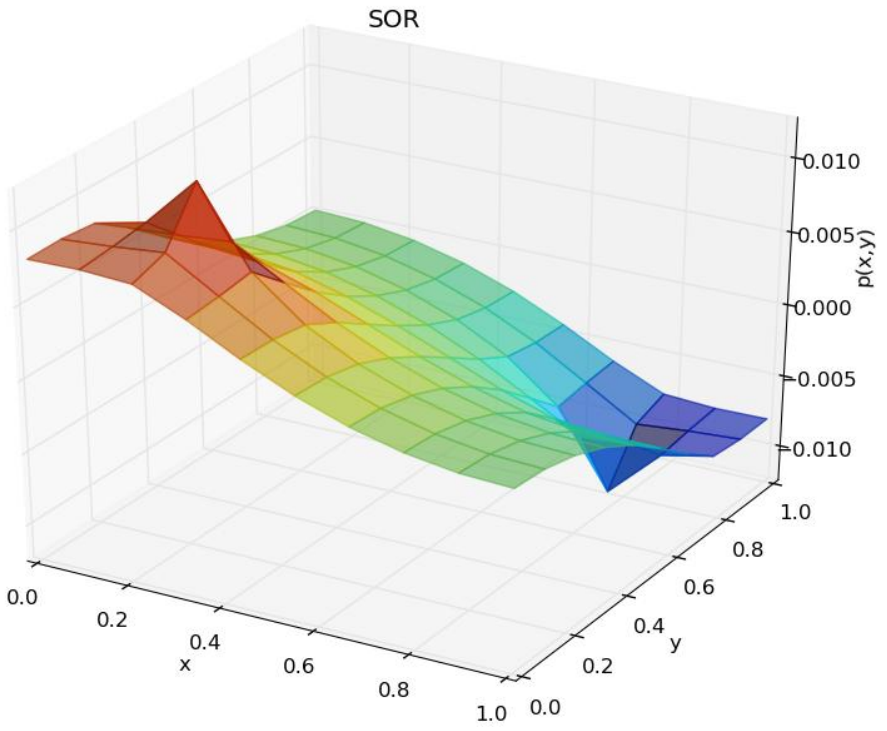


Figura 7 - Modelagem do reservatório pelo método de SOR (Successive Over-Relaxation)

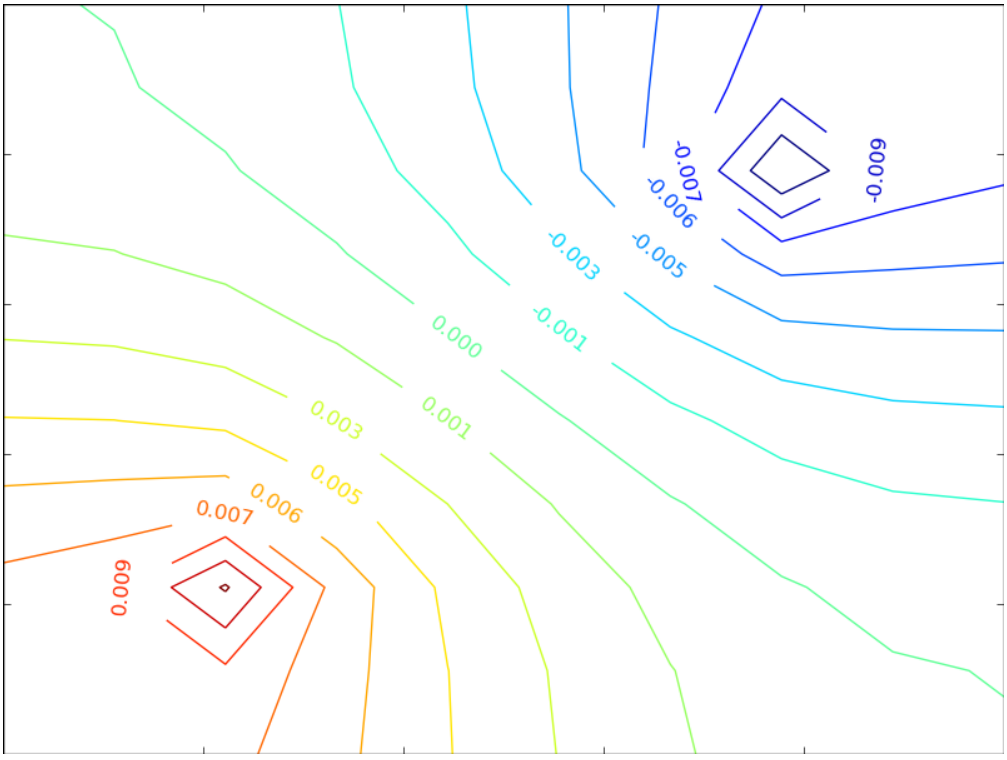


Figura 8 – Curvas de nível das pressões utilizando o método de SOR (Successive Over-Relaxation)

ELIMINAÇÃO GAUSSIANA

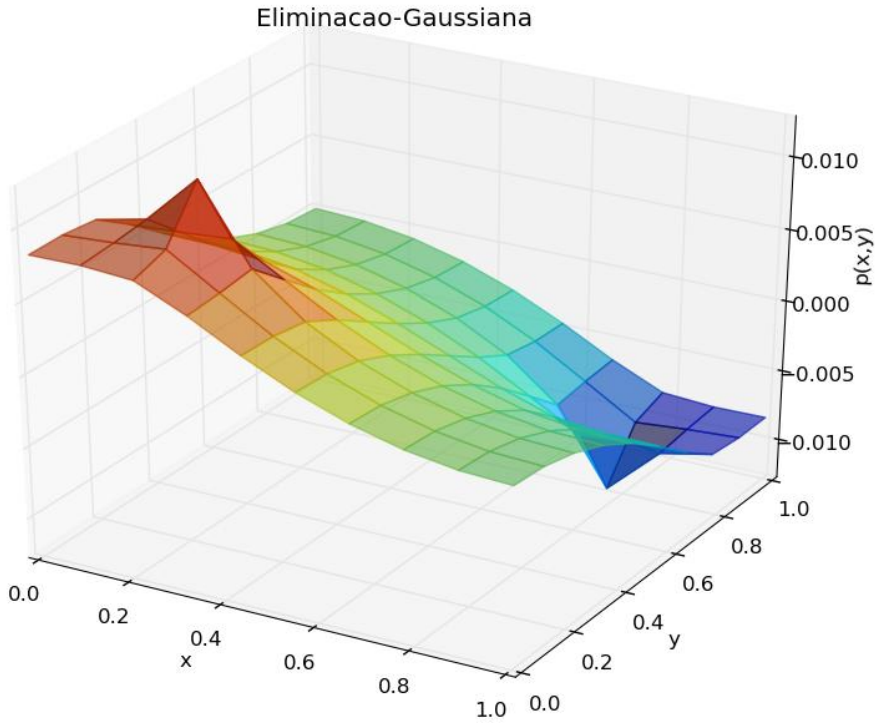


Figura 9 - Modelagem do reservatório utilizando a técnica da Eliminação Gaussiana

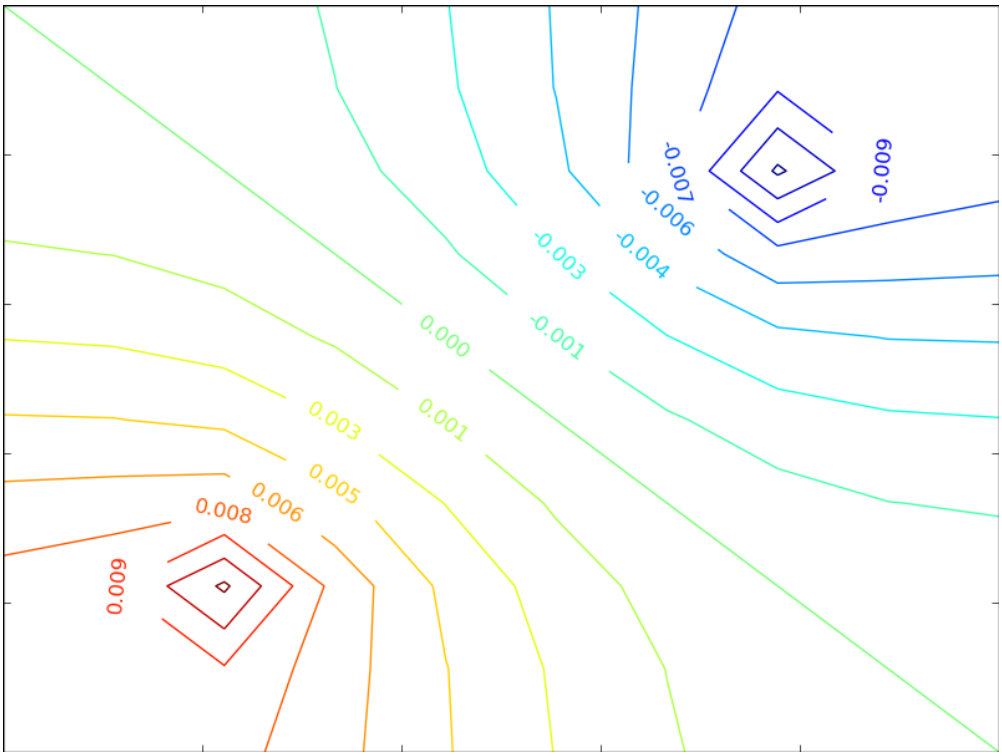


Figura 10 - Curvas de nível das pressões utilizando o método da Eliminação Gaussiana

ANEXO I – ALGORITMOS DESENVOLVIDOS

Arquivo: Reservatorio.py

```
import SistemaLinear as SL
from matplotlib import cm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from numpy import array, ones, zeros, pi, log, meshgrid, linspace

def Reservatorio(n=10, p0=None, inj=None, prod=None, erro=1E-5, fun =
SL.SSLI5D_GS):
  dx = 1./n
  r_e = 2**(-0.5)
  r_w = 0.2*dx
  b = zeros(n**2)
  if inj is None: inj = (n/5, n/5, 1)
  if prod is None: prod = (7*n/10, 7*n/10, -1)
  b[inj[0]*n + inj[1]] = -inj[2]*2.*dx**2*pi/log(r_e/r_w)
  b[prod[0]*n + prod[1]] = -prod[2]*2.*dx**2*pi/log(r_e/r_w)

  D3 = ones(n**2)*( -4. - 2.*dx**2*pi/log(r_e/r_w) )
  D1 = ones(n**2-n)
  D5 = ones(n**2-n)
  D2 = ones(n**2-1)
  D4 = ones(n**2-1)
  D4[0] += 1.
  D2[-1] += 1.

  for i in range(n):
    D1[-(i+1)] += 1.
    D5[i] += 1.
  for i in range(n, n**2, n):
    D4[i] += 1.
    D2[-i] -= 1.
    D4[i-1] -= 1.
    D2[-(i+1)] += 1.
  if fun == SL.SSLD5D:
    p = array(fun(D1, D2, D3, D4, D5, b))
  else:
    p = array(fun(D1, D2, D3, D4, D5, b, x_=p0, erro=erro))
  p.resize(n,n)
  return p

if __name__ == "__main__":
  x, y = meshgrid(linspace(0,1,10), linspace(0,1, 10))
  p_EG = Reservatorio(fun=SL.SSLI5D_GJ)
  p_GJ = Reservatorio(fun=SL.SSLI5D_GJ)
  p_GS = Reservatorio(fun=SL.SSLI5D_GS)
  p_SOR = Reservatorio(fun=SL.SSLI5D_SOR)

  for p, name in zip((p_EG, p_GJ, p_GS, p_SOR),('Eliminacao-
  Gaussiana.txt', 'Gauss-Jacobi.txt', 'Gauss-Seidel.txt','SOR.txt')):
    f = file(name, 'w')
    f.write('Metodo de '+name[:-4]+'\n\n')
    f.write(' i\j'+' '*8 + (' '*12).join([str(a) for a in range(10)]))
    for i in range(p.shape[0]):
      f.write('\n%2i ' % (i))
    for j in range(p.shape[1]):
      f.write('%13.7f' % (p[i][j]))
    f.close()
```

```
  fig_3d = [plt.figure(i) for i in range(4)]
  for fig_, p, name in zip(fig_3d ,(p_EG, p_GJ, p_GS, p_SOR),
   ('Eliminacao-Gaussiana', 'Gauss-Jacobi', 'Gauss-Seidel', 'SOR')):
    ax = fig_.add_axes((0,0,1,1),projection='3d')
    N = (p-p.min())/(p.max()-p.min()) # Normalização da pressão (p)
    ax.plot_surface(x,y,p, rstride=1, cstride=1, alpha=0.6,
     facecolors=cm.jet(N))
    ax.set_title(name)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('p(x,y)')
    fig_.savefig(name+'.png')

  fig_2d = [plt.figure(i) for i in range(4)]
  for fig_, p, name in zip(fig_2d ,(p_EG, p_GJ, p_GS, p_SOR),
   ('Eliminacao-Gaussiana', 'Gauss-Jacobi', 'Gauss-Seidel', 'SOR')):
    ax = fig_.add_axes((0,0,1,1))
    cset = ax.contour(x,y, p, 20)
    ax.set_title(name)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.clabel(cset, inline=1, fotsize=10)
    fig_.savefig(name+'-2d.png')
```

Arquivo: SistemaLinear.py

```
from numpy import array, append
from numpy import zeros, empty

def M5D2M(D1, D2, D3, D4, D5):
  N = D3.size
  n = int(N**0.5)
  M = zeros((N,N))
  M[0][0], M[0][1], M[0][n] = D3[0], D4[0], D5[0]
  for i in range(1, n):
    M[i][i-1], M[i][i], M[i][i+1], M[i][i+n] = D2[i-1], D3[i], D4[i],
     D5[i]
  for i in range(n, N-n):
    M[i][i-n], M[i][i-1], M[i][i], M[i][i+1], M[i][i+n] = D1[i-n],
     D2[i-1], D3[i], D4[i], D5[i]
  for i in range(N-n, N-1):
    M[i][i-n], M[i][i-1], M[i][i], M[i][i+1] = D1[i-n], D2[i-1],
     D3[i], D4[i]
  M[N-1][N-n], M[N-1][N-2], M[N-1][N-1] = D1[N-n-1], D2[N-2], D3[N-1]
  return M

def SSLD(A, b):
  try:
    b.resize(b.size, 1)
    C = append(A, b, 1)
  except MemoryError:
    print 'Memória insuficiente!'
    return empty(A.shape)
  i = 0
  j = 0
```

```
  while i < C.shape[0] and j < C.shape[1]:
    maxi = i
    for k in range(i+1, C.shape[0]):
      if abs(C[k][j]) > abs(C[maxi,j]): maxi = k
    if C[maxi, j]:
      C[[i,maxi]] = C[[maxi,i]]
      C[i] /= C[i][j]
      for u in range(i+1, C.shape[0]): C[u] -= C[i]*C[u,j]
      i += 1
    j += 1
  print j
  x = array([])
  for l in range(C.shape[0]):
    x_ = C[-1-l][-1]
    for m in range(l): x_ -= C[-1-l][-2-m]*x[m]
  x = append(x, x_)
  return x[::-1]

def SSLD5D(D1, D2, D3, D4, D5, b):
  return SSLD(M5D2M(D1, D2, D3, D4, D5), b)

def SSLI5D_SOR(D1, D2, D3, D4, D5, b, w=0.5, x_=None, erro=1.0E-6):
  error = True
  N = b.size
  n = int(N**0.5)
  if x_ is None: x_ = array([0.0 for i in b])
  while(error > erro):
    x = array([(1-w)*x_[0] + w*(b[0] - D4[0]*x_[1] -
     D5[0]*x_[n])/D3[0]])
    for i in range(1, n):
      x = append(x, (1-w)*x_[i] + w*(b[i]- D2[i-1]*x[i-1] -
       D4[i]*x_[i+1] - D5[i]*x_[i+n])/D3[i])
    for i in range(n, N-n):
      x = append(x, (1-w)*x_[i] + w*(b[i] - D1[i-n]*x[i-n] - D2[i-
       1]*x[i-1] - D4[i]*x_[i+1] - D5[i]*x_[i+n])/D3[i])
    for i in range(N-n, N-1):
      x = append(x, (1-w)*x_[i] + w*(b[i] - D1[i-n]*x[i-n] - D2[i-
       1]*x[i-1] - D4[i]*x_[i+1])/D3[i])
    x = append(x, (1-w)*x_[N-1] + w*(b[N-1]- D1[N-n-1]*x[N-n-1] -
     D2[N-2]*x[N-2])/D3[N-1])
    error = abs(x-x_).max()
    x_ = x
    j+=1
  print 'SOR:', j
  return x

def SSLI5D_GJ(D1, D2, D3, D4, D5, b, x_=None, erro=1.0E-6):
  error = True
  N = b.size
  n = int(N**0.5)
  if x_ is None: x_ = array([0.0 for i in b])

  while(error > erro):
    x = array([(b[0] - D4[0]*x_[1] - D5[0]*x_[n])/D3[0]])
    for i in range(1, n):
      x = append(x, (b[i]- D2[i-1]*x_[i-1] - D4[i]*x_[i+1] -
       D5[i]*x_[i+n])/D3[i])
    for i in range(n, N-n):
      x = append(x, (b[i] - D1[i-n]*x_[i-n] - D2[i-1]*x_[i-1] -
       D4[i]*x_[i+1] - D5[i]*x_[i+n])/D3[i])
```

```
    for i in range(N-n, N-1):
      x = append(x, (b[i] - D1[i-n]*x_[i-n] - D2[i-1]*x_[i-1] -
       D4[i]*x_[i+1])/D3[i])
    x = append(x, (b[N-1]- D1[N-n-1]*x_[N-n-1] - D2[N-2]*x_[N-
     2])/D3[N-1])
    error = abs(x-x_).max()
    x_ = x
  return x

def SSLI5D_GS(D1, D2, D3, D4, D5, b, x_=None, erro=1.0E-6):
  error = True
  N = b.size
  n = int(N**0.5)
  if x_ is None: x_ = array([0.0 for i in b])

  while(error > erro):
    x = array([(b[0] - D4[0]*x_[1] - D5[0]*x_[n])/D3[0]])
    for i in range(1, n):
      x = append(x, (b[i]- D2[i-1]*x[i-1] - D4[i]*x_[i+1] -
       D5[i]*x_[i+n])/D3[i])
    for i in range(n, N-n):
      x = append(x, (b[i] - D1[i-n]*x[i-n] - D2[i-1]*x[i-1] -
       D4[i]*x_[i+1] - D5[i]*x_[i+n])/D3[i])
    for i in range(N-n, N-1):
      x = append(x, (b[i] - D1[i-n]*x[i-n] - D2[i-1]*x[i-1] -
       D4[i]*x_[i+1])/D3[i])
    x = append(x, (b[N-1]- D1[N-n-1]*x[N-n-1] - D2[N-2]*x[N-2])/D3[N-
     1])
    error = abs(x-x_).max()
    x_ = x
  return x
```

ANEXO II – DADOS DE PRESSÃO

Metodo de Gauss-Jacobi

i\j	0	1	2	3	4
0	0.0082798	0.0083262	0.0080854	0.0064845	0.0046052
1	0.0083262	0.0085433	0.0088543	0.0066807	0.0045451
2	0.0080854	0.0088543	0.0122639	0.0069856	0.0041738
3	0.0064845	0.0066807	0.0069856	0.0049470	0.0029620
4	0.0046052	0.0045451	0.0041738	0.0029620	0.0014728
5	0.0029509	0.0028008	0.0022946	0.0013068	-0.0000000
6	0.0016486	0.0014772	0.0009375	-0.0000000	-0.0013068
7	0.0007282	0.0005481	-0.0000000	-0.0009375	-0.0022946
8	0.0001810	-0.0000000	-0.0005481	-0.0014772	-0.0028008
9	-0.0000000	-0.0001810	-0.0007282	-0.0016486	-0.0029509

i\j	5	6	7	8	9
0	0.0029509	0.0016486	0.0007282	0.0001810	-0.0000000
1	0.0028008	0.0014772	0.0005481	-0.0000000	-0.0001810
2	0.0022946	0.0009375	-0.0000000	-0.0005481	-0.0007282
3	0.0013068	-0.0000000	-0.0009375	-0.0014772	-0.0016486
4	-0.0000000	-0.0013068	-0.0022946	-0.0028008	-0.0029509
5	-0.0014728	-0.0029620	-0.0041738	-0.0045451	-0.0046052
6	-0.0029620	-0.0049470	-0.0069856	-0.0066807	-0.0064845
7	-0.0041738	-0.0069856	-0.0122639	-0.0088543	-0.0080854
8	-0.0045451	-0.0066807	-0.0088543	-0.0085433	-0.0083262
9	-0.0046052	-0.0064845	-0.0080854	-0.0083262	-0.0082798

Metodo de Gauss-Seidel

i\j	0	1	2	3	4
0	0.0085777	0.0086246	0.0083945	0.0067949	0.0049233
1	0.0086246	0.0088513	0.0091630	0.0069979	0.0048621
2	0.0083945	0.0091630	0.0125807	0.0073021	0.0044947
3	0.0067949	0.0069979	0.0073021	0.0052681	0.0032812
4	0.0049233	0.0048621	0.0044947	0.0032812	0.0017918
5	0.0032687	0.0031214	0.0026135	0.0016256	0.0003154
6	0.0019687	0.0017958	0.0012558	0.0003153	-0.0009964
7	0.0010464	0.0008657	0.0003152	-0.0006265	-0.0019882
8	0.0004977	0.0003151	-0.0002362	-0.0011694	-0.0025012
9	0.0003150	0.0001321	-0.0004183	-0.0013451	-0.0026531

i\j	5	6	7	8	9
0	0.0032687	0.0019687	0.0010464	0.0004977	0.0003150
1	0.0031214	0.0017958	0.0008657	0.0003151	0.0001321
2	0.0026135	0.0012558	0.0003152	-0.0002362	-0.0004183
3	0.0016256	0.0003153	-0.0006265	-0.0011694	-0.0013451
4	0.0003154	-0.0009964	-0.0019882	-0.0025012	-0.0026531
5	-0.0011626	-0.0026565	-0.0038766	-0.0042514	-0.0043190
6	-0.0026565	-0.0046508	-0.0066940	-0.0063995	-0.0062048
7	-0.0038766	-0.0066940	-0.0119844	-0.0085783	-0.0078195
8	-0.0042514	-0.0063995	-0.0085783	-0.0082799	-0.0080641
9	-0.0043190	-0.0062048	-0.0078195	-0.0080641	-0.0080287

Metodo de SOR

i\j	0	1	2	3	4
0	0.0081099	0.0081649	0.0079579	0.0063940	0.0045663
1	0.0081649	0.0083999	0.0087350	0.0066058	0.0045140
2	0.0079579	0.0087350	0.0121764	0.0069339	0.0041708
3	0.0063940	0.0066058	0.0069339	0.0049362	0.0029938
4	0.0045663	0.0045140	0.0041708	0.0029938	0.0015491
5	0.0029586	0.0028204	0.0023368	0.0013856	0.0001202
6	0.0017029	0.0015393	0.0010237	0.0001200	-0.0011468
7	0.0008171	0.0006457	0.0001196	-0.0007852	-0.0021020
8	0.0002927	0.0001193	-0.0004075	-0.0013038	-0.0025907
9	0.0001192	-0.0000545	-0.0005804	-0.0014703	-0.0027335

i\j	5	6	7	8	9
0	0.0029586	0.0017029	0.0008171	0.0002927	0.0001192
1	0.0028204	0.0015393	0.0006457	0.0001193	-0.0000545
2	0.0023368	0.0010237	0.0001196	-0.0004075	-0.0005804
3	0.0013856	0.0001200	-0.0007852	-0.0013038	-0.0014703
4	0.0001202	-0.0011468	-0.0021020	-0.0025907	-0.0027335
5	-0.0013101	-0.0027592	-0.0039428	-0.0042934	-0.0043521
6	-0.0027592	-0.0047089	-0.0067156	-0.0063970	-0.0061935
7	-0.0039428	-0.0067156	-0.0119695	-0.0085396	-0.0077722
8	-0.0042934	-0.0063970	-0.0085396	-0.0082175	-0.0079932
9	-0.0043521	-0.0061935	-0.0077722	-0.0079932	-0.0079494

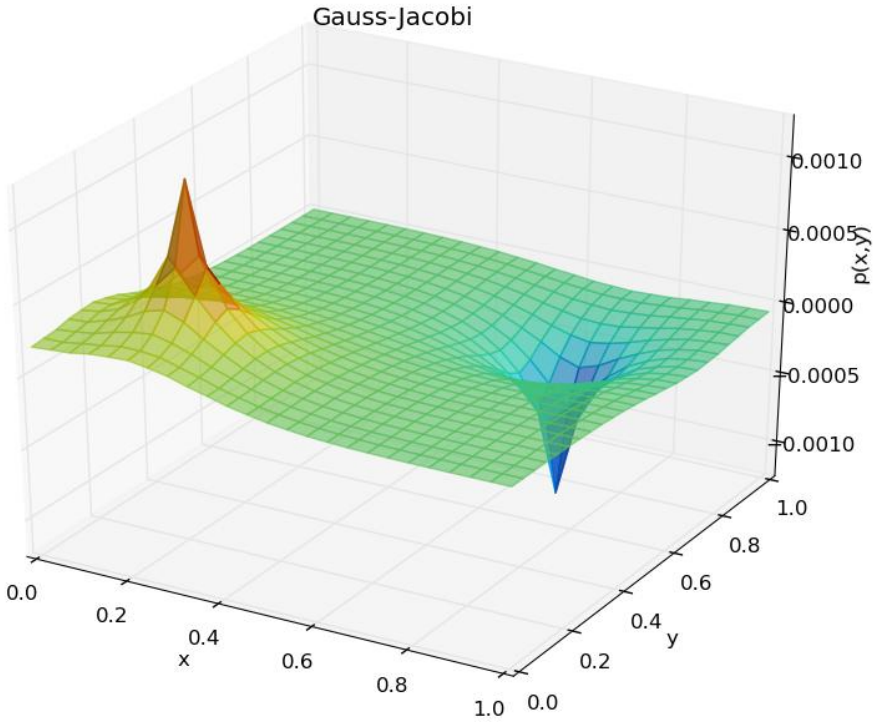
Metodo de Eliminacao-Gaussiana

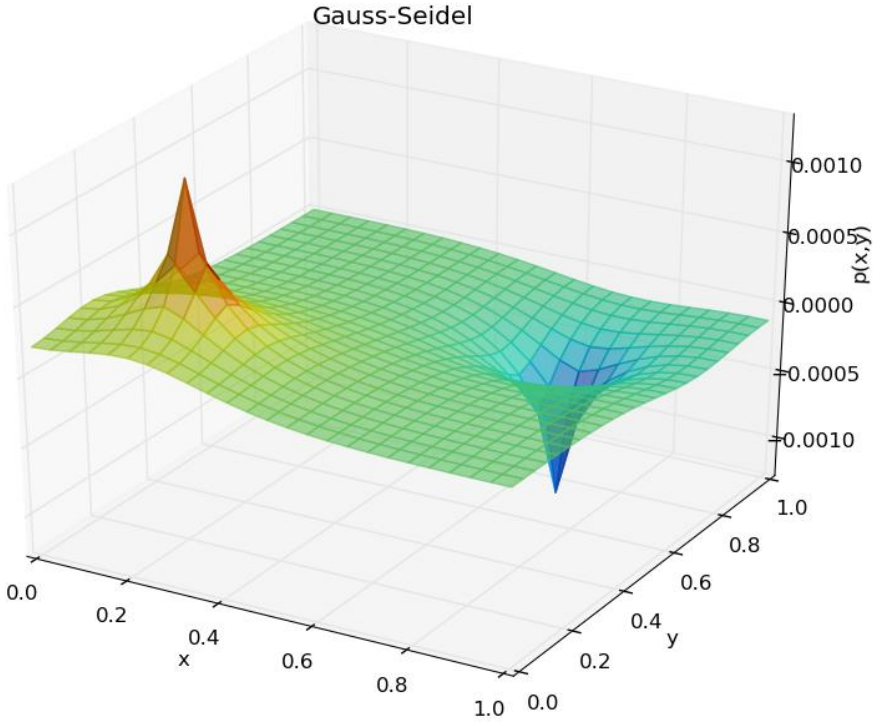
i\j	0	1	2	3	4
0	0.0082798	0.0083262	0.0080854	0.0064845	0.0046052
1	0.0083262	0.0085433	0.0088543	0.0066807	0.0045451
2	0.0080854	0.0088543	0.0122639	0.0069856	0.0041738
3	0.0064845	0.0066807	0.0069856	0.0049470	0.0029620
4	0.0046052	0.0045451	0.0041738	0.0029620	0.0014728
5	0.0029509	0.0028008	0.0022946	0.0013068	-0.0000000
6	0.0016486	0.0014772	0.0009375	-0.0000000	-0.0013068
7	0.0007282	0.0005481	-0.0000000	-0.0009375	-0.0022946
8	0.0001810	-0.0000000	-0.0005481	-0.0014772	-0.0028008
9	-0.0000000	-0.0001810	-0.0007282	-0.0016486	-0.0029509

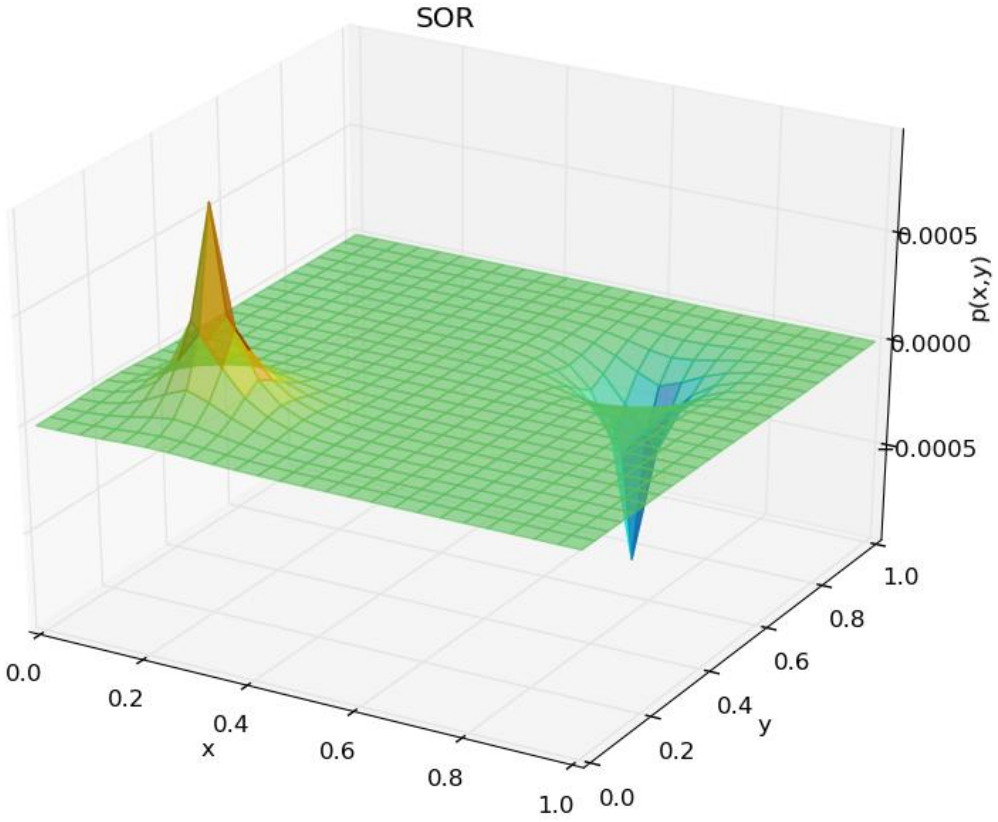
i\j	5	6	7	8	9
0	0.0029509	0.0016486	0.0007282	0.0001810	-0.0000000
1	0.0028008	0.0014772	0.0005481	-0.0000000	-0.0001810
2	0.0022946	0.0009375	-0.0000000	-0.0005481	-0.0007282
3	0.0013068	-0.0000000	-0.0009375	-0.0014772	-0.0016486
4	-0.0000000	-0.0013068	-0.0022946	-0.0028008	-0.0029509
5	-0.0014728	-0.0029620	-0.0041738	-0.0045451	-0.0046052
6	-0.0029620	-0.0049470	-0.0069856	-0.0066807	-0.0064845
7	-0.0041738	-0.0069856	-0.0122639	-0.0088543	-0.0080854
8	-0.0045451	-0.0066807	-0.0088543	-0.0085433	-0.0083262
9	-0.0046052	-0.0064845	-0.0080854	-0.0083262	-0.0082798

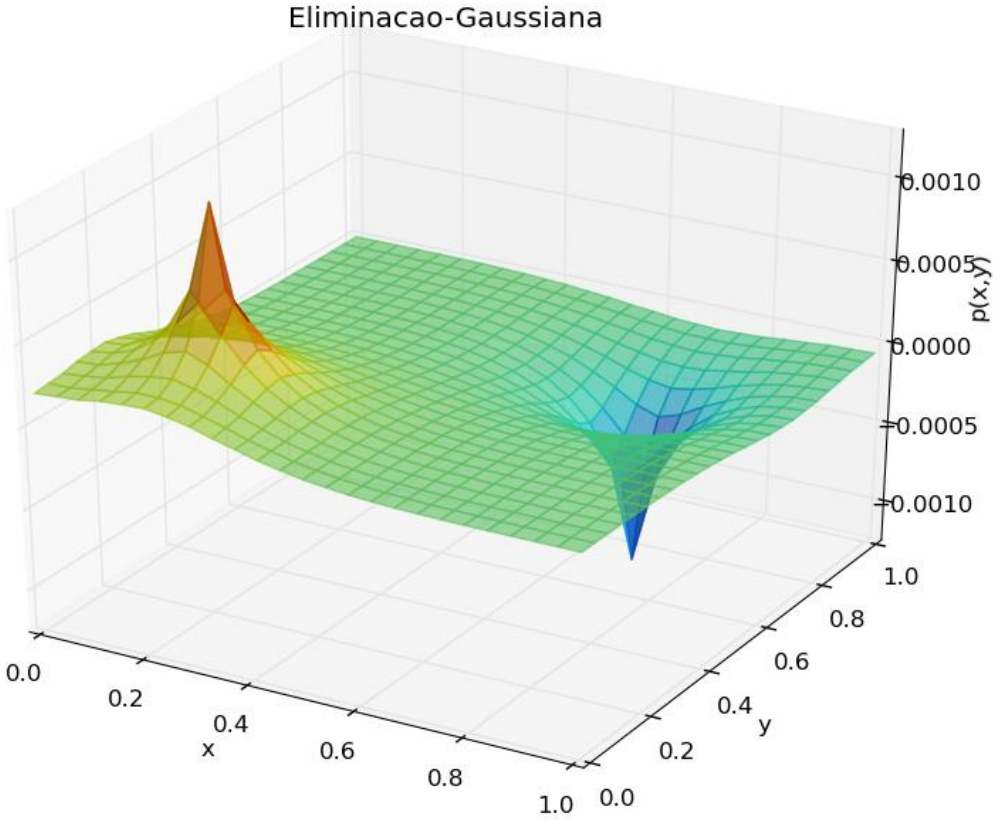
ANEXO III – OUTROS VALORES DE MALHA

N = 25

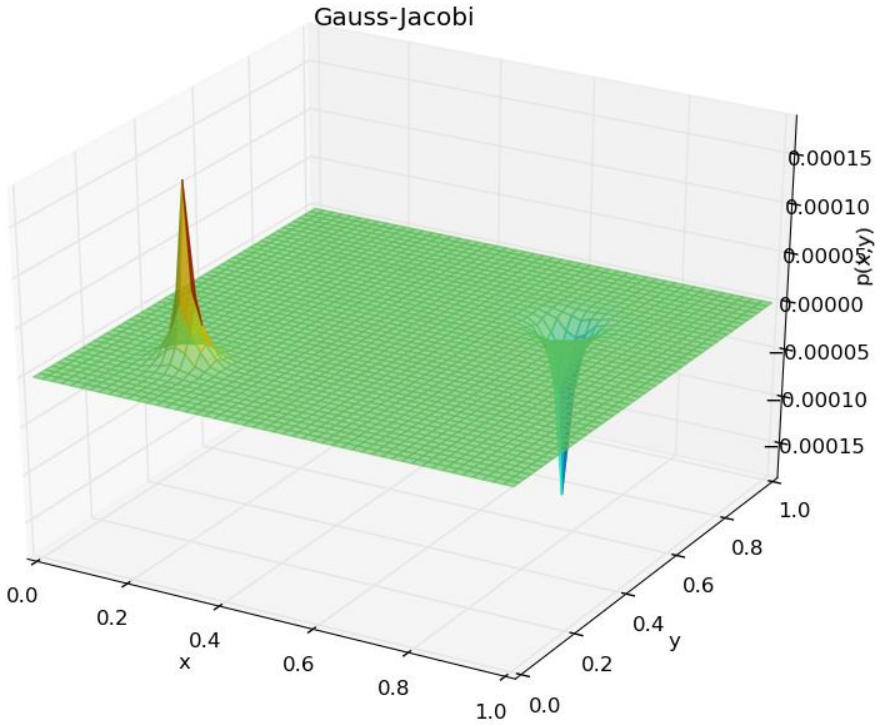


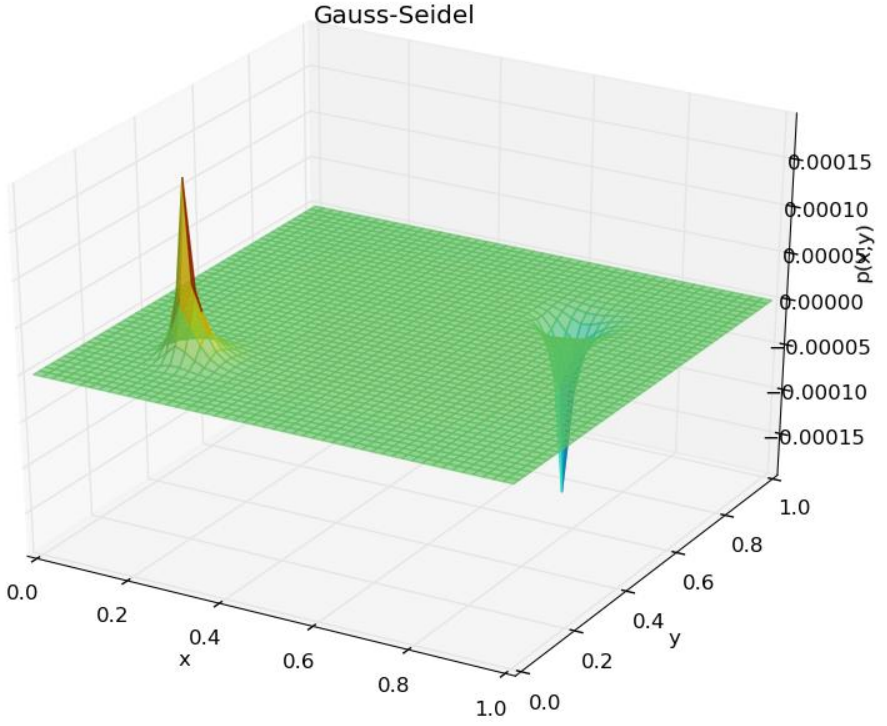


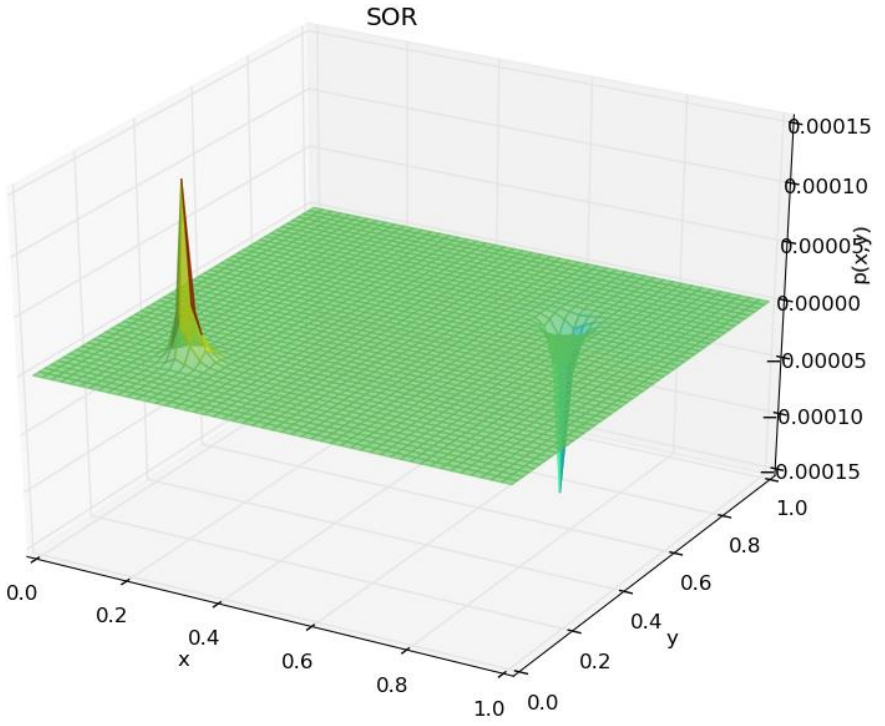

SOR
0.0005
0.0000
0.0005
p(x,y)
0.0
0.2
0.4
0.6
0.8
1.0
x
0.0
0.2
0.4
0.6
0.8
1.0
y

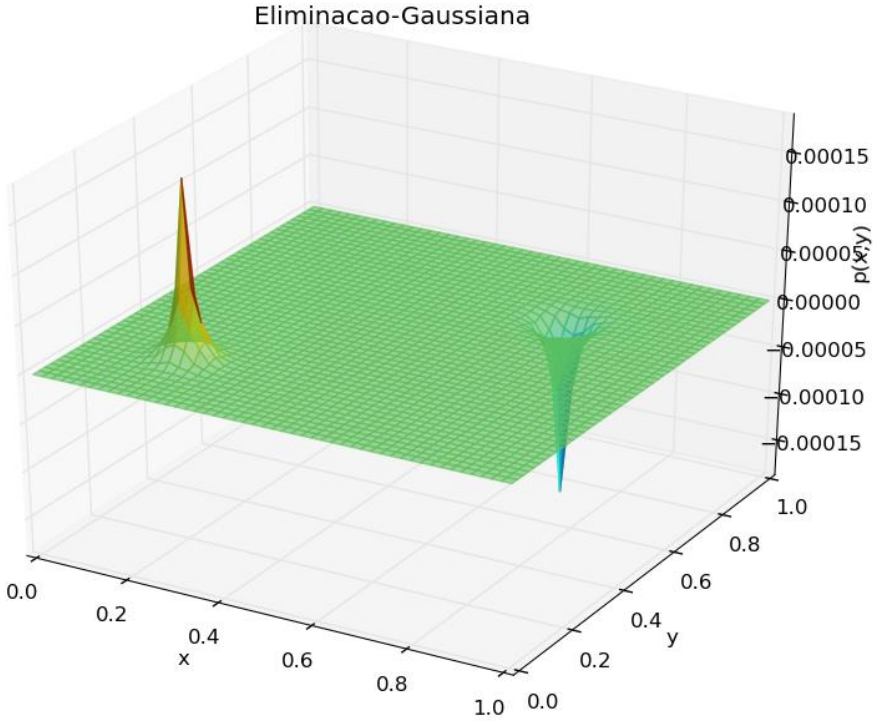

Eliminacao-Gaussiana
0.0010
0.0005
0.0000
0.0005
0.0010
p(x,y)
0.0
0.2
0.4
0.6
0.8
1.0
x
0.0
0.2
0.4
0.6
0.8
1.0
y

N=50


Gauss-Jacobi
0.00015
0.00010
0.00005
0.00000
−0.00005
−0.00010
−0.00015
p(x,y)
0.0
0.2
0.4
0.6
0.8
1.0
x
y


Gauss-Seidel
0.00015
0.00010
0.00005
0.00000
−0.00005
−0.00010
−0.00015
p(x,y)
0.0
0.2
0.4
0.6
0.8
1.0
x
y


SOR
0.00015
0.00010
0.00005
0.00000
0.00005
0.00010
0.00015
p(x,y)
0.0
0.2
0.4
0.6
0.8
1.0
x
y


Eliminacao-Gaussiana
0.00015
0.00010
0.00005
0.00000
0.00005
0.00010
0.00015
p(x,y)
0.0
0.2
0.4
0.6
0.8
1.0
x
y