

Álgebra Linear Computacional - Grupo 4

Felipe Pêpe, Gustavo Oliveira, Igor Godinho, Lucas Inocêncio, Matheus Cardoso

Solver, o primeiro problema

Aqui, primeiramente projetamos a matriz para resolver o problema:

$$\frac{\partial}{\partial x} \left(k \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(k \frac{\partial u}{\partial y} \right) = f(x, y), \forall \quad x, y \text{ em } \Omega : (0, 1) \times (0, 1)$$

condições de contorno: $u(x, y) = 0, \forall \quad x, y \text{ em } \partial\Omega$

Iniciando o Problema

```
L <- 1
n <- 10
k <- 7

d <- L / (n - 1)

# Gerando D (Matriz que faz d2u)
solver <- matrix(0, n*n, n*n)

for (i in 1:(n * n)) {
  if (i <= n || i >= (n*n - n) || i %% n <= 1) {
    solver[i, i] <- 1
    next
  }

  solver[i, i - n] <- 1
  solver[i, i - 1] <- 1
  solver[i, i] <- -4
}
```

```

    solver[i, i + 1] <- 1
    solver[i, i + n] <- 1
  }

A <- solve(solver)

```

Perceba que o R já possui uma função para calcular a inversa de uma matriz, a função *solve*. A partir deste momento, já temos a matriz A.

Projetando a função f .

Nesse caso, escolhemos que f seja a função:

$$f(x, y) = -\frac{e^x}{|y| + 1}$$

```

f <- matrix(0, n * n, 1)
X = seq(0, L, d)
for (i in 1:length(X)) {
  for (j in 1:length(X)) {
    f[i + (j - 1) * n] <- - ((exp(X[i]) / (abs(X[j]) + 1)) * ((d ** 2) / k))
  }
}

solution <- A %*% f

```

Ao final, obtemos a matriz solução.

1. Temos a matriz A.
2. Temos o vetor força f .

Fazemos:

$$U = Af$$

Em que U é o vetor solução.

Aplicando as Condições de Contorno

Apenas definimos a matriz cujos valores serão o do espaço (em z). Caso o ponto esteja nas bordas, o valor é zero.

```
x <- seq(0, L, d)
y <- seq(0, L, d)

z <- matrix(0, n, n)
for (i in 1:n) {
  for (j in 1:n) {
    if (i == 1 || j == 1) {
      z[i, j] <- 0
    } else {
      z[i, j] <- solution[i + (j - 1) * n]
    }
  }
}
```

Plotagem do Gráfico

Finalmente, plotamos o gráfico.

```
plotting_mesh <- function(solution, to_pdf=FALSE) {
  fig <- plot_ly(
    type = 'surface',
    contours = list(
      x = list(show = TRUE, start = 1.5, end = 2, size = 0.04, color = 'white'),
      z = list(show = TRUE, start = 0.5, end = 0.8, size = 0.05)),
    x = ~x,
    y = ~y,
    z = ~z)

  fig <- fig %>% layout(
    scene = list(
      xaxis = list(nticks = 20),
      zaxis = list(nticks = 4),
      camera = list(eye = list(x = 0, y = -1, z = 1)),
      aspectratio = list(x = .9, y = .8, z = .8)))

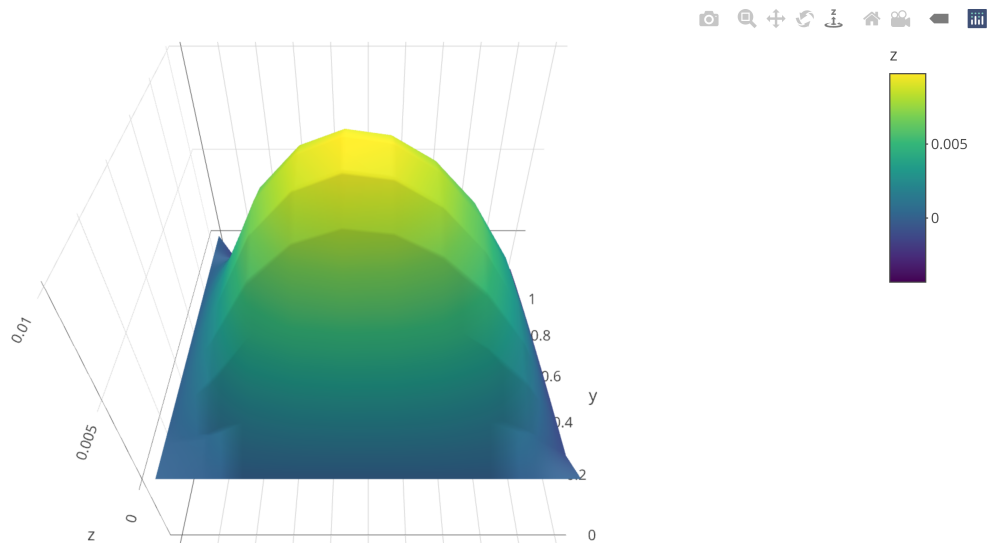
  if (to_pdf == TRUE) {
    htmlwidgets::saveWidget(widget = fig, file = "hc.html")
  }
}
```

```

    webshot(url = "hc.html", file = "hc.png", delay = 1, zoom = 4, vheight = 500)
  } else {
    return(fig)
  }
}

plotting_mesh(solution, to_pdf = TRUE)

```



Grids e Grafos

Vamos gerar uma malha estruturada e montar a matriz de adjacência.

Código necessário

```

# Gerar espaço dado um intervalo
linspace <- function(from, to, nodes) {
  div <- (to - from) / (nodes - 1)
  return(seq(from, to, div))
}

# Gera as metrizes em x e y

```

```

generate_gen_matrices <- function(lines, columns) {
  mtx <- t(matrix(linspace(1, lines, columns), columns, lines))
  mty <- matrix(linspace(lines, 1, lines), lines, columns)

  return(list("x"=mtx, "y"=mty))
}

# Recebe a ordem de uma matriz e devolve
# um espaço de triângulos
get_triangles <- function(lines, columns, randomize_vertices=FALSE) {
  grid <- generate_gen_matrices(lines, columns) # Gera as matrizes
  mx <- grid$x
  my <- grid$y

  triangles_x <- matrix(0, 1, 3)      # - Eixo x das coordenadas
  triangles_y <- matrix(0, 1, 3)      # - Eixo y das coordenadas
  triangles_v <- matrix(0, 1, 3)      # - Triângulos formados
  # pelos vértices

  vertices <- matrix(0, lines, columns) # - Para o Grid com os
  # vértices numerados

  V <- rep(1:(lines*columns))
  if (randomize_vertices == TRUE) {
    set.seed(245)
    V <- sample(V)
  }

  for (i in 1:lines) {
    for (j in 1:columns) {
      vertices[i, j] <- V[(i - 1) * columns + j]
    }
  }

  # Fazendo a triangulação
  for (i in (lines):(2)) {
    for (j in 1:(columns-1)) {
      triangles_x <- rbind(
        triangles_x,
        c(mx[i, j], mx[i-1, j], mx[i-1, j+1]))
      triangles_x <- rbind(
        triangles_x,

```

```

        c(mx[i, j], mx[i, j+1], mx[i-1, j+1]))

triangles_y <- rbind(
  triangles_y,
  c(my[i, j], my[i-1, j], my[i-1, j+1]))
triangles_y <- rbind(
  triangles_y,
  c(my[i, j], my[i, j+1], my[i-1, j+1]))

triangles_v <- rbind(
  triangles_v,
  c(vertices[lines - i + 1, j],
    vertices[lines - i + 2, j],
    vertices[lines - i + 2, j+1]))
triangles_v <- rbind(
  triangles_v,
  c(vertices[lines - i + 1, j],
    vertices[lines - i + 1, j+1],
    vertices[lines - i + 2, j+1]))
}
}

# Removendo a primeira linha da cada
triangles_x <- triangles_x[2:nrow(triangles_x), ]
triangles_y <- triangles_y[2:nrow(triangles_y), ]
triangles_v <- triangles_v[2:nrow(triangles_v), ]

# Retorna toda a "situação"
return(
  list("mx"=mx, "my"=my, "x_axis"=triangles_x,
    "y_axis"=triangles_y, "mv"=triangles_v, "vertices"=vertices))
}

# Receive a list like:
## triangled_grid$mx      == grid in x axis
## triangled_grid$my      == grid in y axis
## triangled_grid$trx     == coordinates in x axis
## triangled_grid$trx     == coordinates in x axis
## triangled_grid$try     == coordinates in y axis
## triangled_grid$mv      == triangles based in order of vertices
## triangled_grid$vertices == grid with vertices
plot_mesh_grid <- function(triangled_grid) {

```

```

mx <- triangled_grid$mx
my <- triangled_grid$my
trx <- triangled_grid$x_axis
try <- triangled_grid$y_axis
mv <- triangled_grid$mv
vertices <- triangled_grid$vertices
lines <- nrow(mx)
columns <- ncol(my)

# Plot inicial, apenas o grid
plot(mx, my, xlab="x", ylab="y", main="Space Grid")
q_triangles <- length(mv) / 3

# Plotagem dos triângulos e seus respectivos números
for (i in 1:q_triangles) {
  polygon(trx[i,], try[i, ])
  text(sum(trx[i,]) / 3, sum(try[i, ]) / 3, i, col="blue")
}

# Plotagem dos números dos vértices
for (i in 1:lines) {
  for (j in 1:columns) {
    text(mx[i, j], my[lines - i + 1, j], vertices[i, j], col="red", cex=2.1)
  }
}

# Criação da matriz de adjacência
create_adj_matrix <- function(triangled_grid) {
  lines <- nrow(triangled_grid$mx)
  columns <- ncol(triangled_grid$my)

  adj_matrix <- matrix(0, lines*columns, lines*columns)

  q_triangles <- length(triangled$mv) / 3

  for (triangle in 1:q_triangles) {
    for (i in 1:3) {
      for (j in i:3) {
        v1 <- triangled_grid$mv[triangle, i]
        v2 <- triangled_grid$mv[triangle, j]
        adj_matrix[v1, v2] <- 1
      }
    }
  }
}

```

```

        adj_matrix[v2, v1] <- 1
      }
    }
  }

  return(adj_matrix)
}

# Plotagem da matriz de adjacência
plot_adj_matrix <- function(adj_matrix) {
  heatmap(adj_matrix, Colv = "Rowv", Rowv = NA,
          symm=TRUE, main="Matriz de Adjacência")
}

```

Vendo funcionar

A partir daqui temos todo o necessário para gerar matrizes e fazer a triangulação.

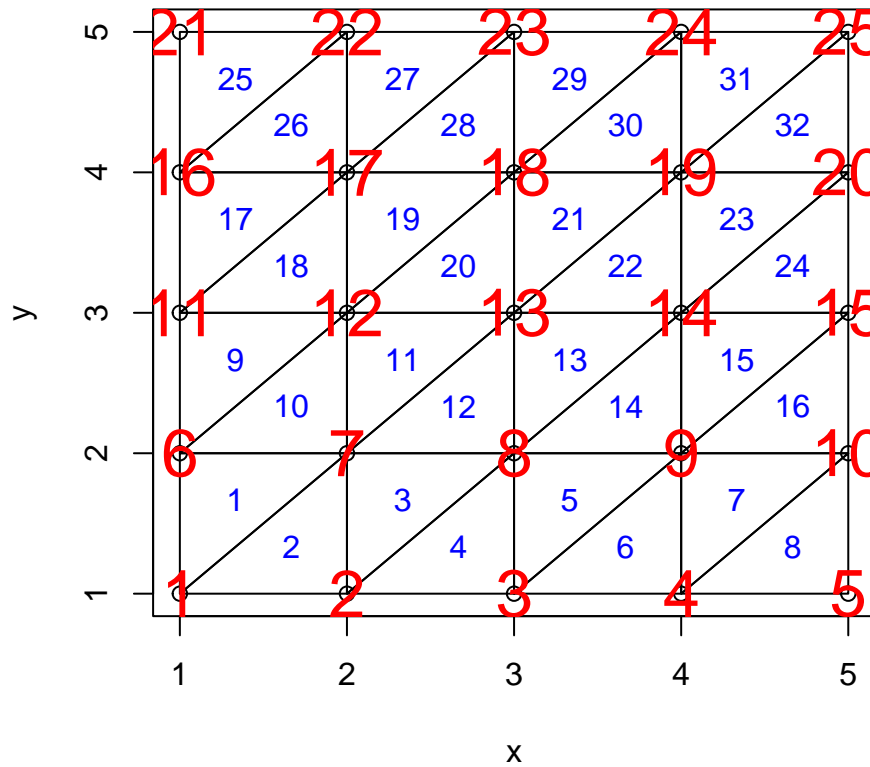
Perceba que a ordem da matriz está em dois lugares. Isso pelo fato de termos a possibilidade de colocarmos um número diferente de linhas e colunas, para realizar mais experimentos interessantes.

```

# Gera uma matriz com vértices ordenados e faz a triangulação
triangled <- get_triangles(5, 5, FALSE)
plot_mesh_grid(triangled)

```


Space Grid

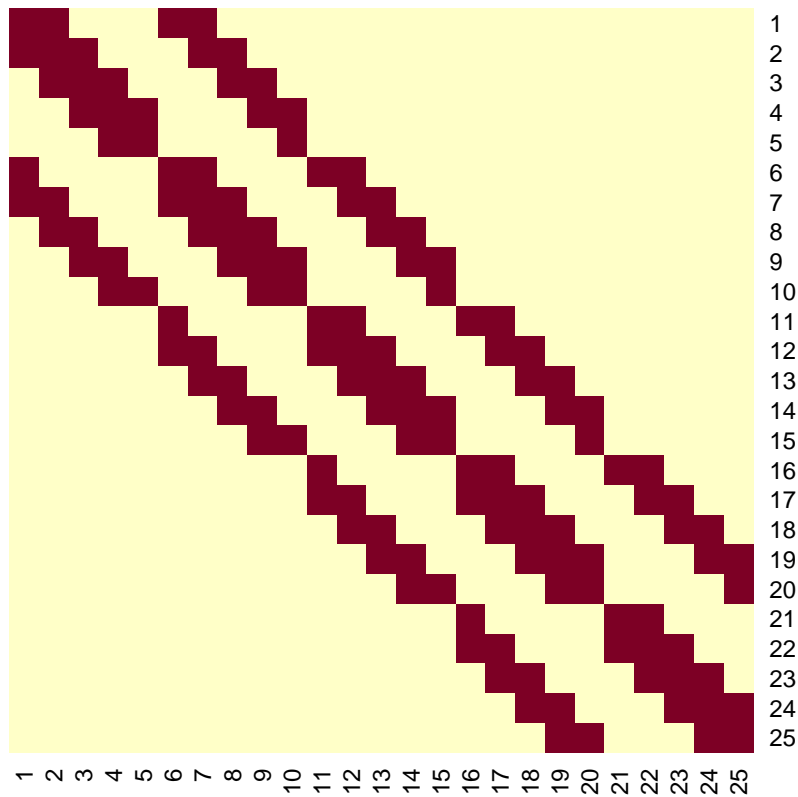


Agora, vamos à matriz de adjacências, que foi gerada através de um gráfico de heatmap, como é visto no código principal.

Isso é pelo fato da matriz de adjacência ser composta por zeros e uns, o que facilita nesse caso.

```
adj <- create_adj_matrix(triangled)
plot_adj_matrix(adj)
```

Matriz de Adjacência



Essa é a matriz gerada.

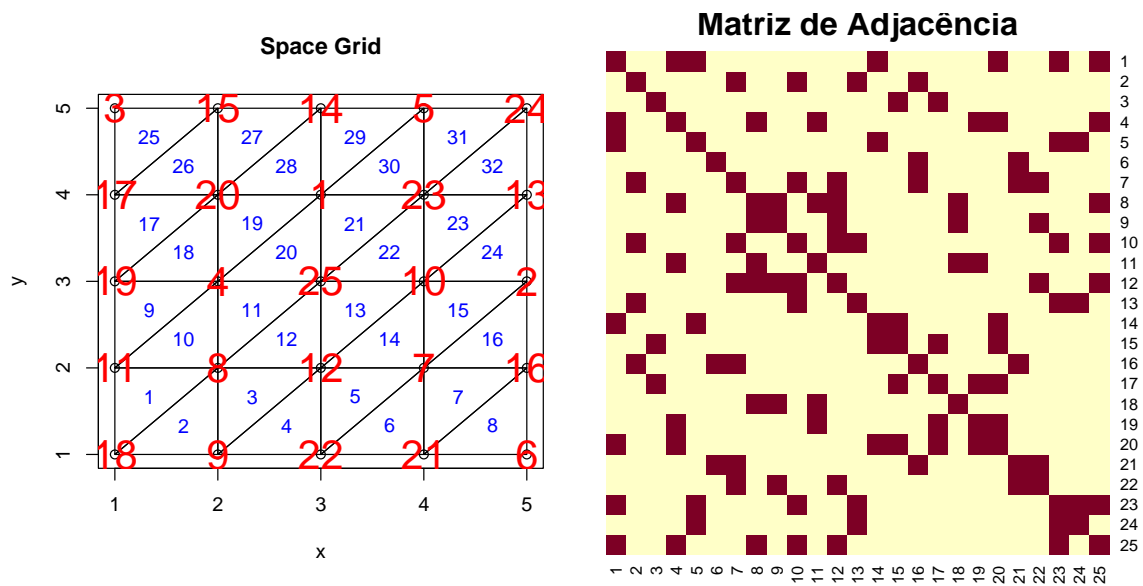
Trocando a ordem dos vértices

Neste momento, vamos experimentar trocar a ordem dos vértices para verificar como ficará a matriz de adjacências.

```
triangled <- get_triangles(5, 5, TRUE)
adj <- create_adj_matrix(triangled)
```

```
plot_mesh_grid(triangled)
plot_adj_matrix(adj)
```

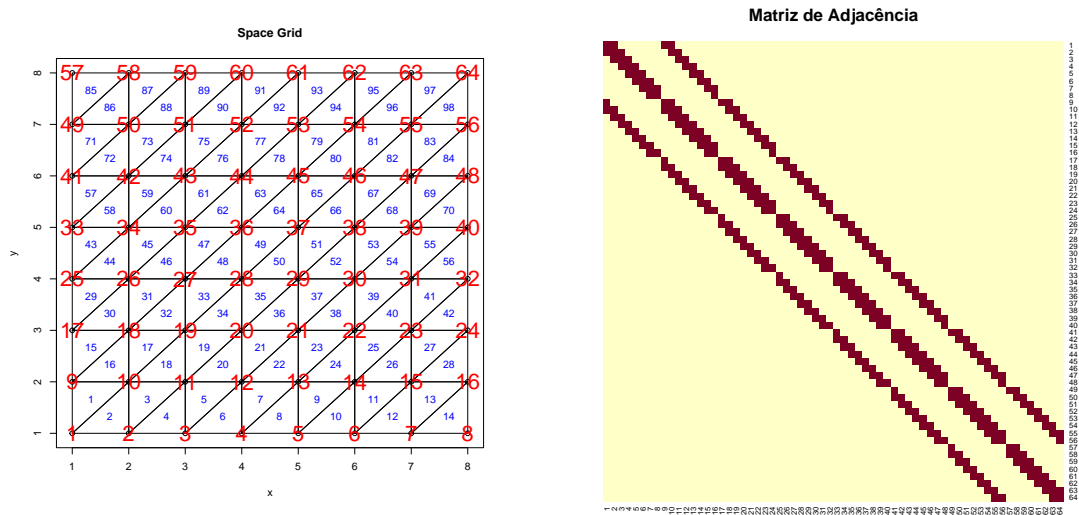
De fato, a matriz é gerada corretamente!



Matrizes Maiores e não Quadradas

Vamos elevar levemente o nível para matrizes maiores:

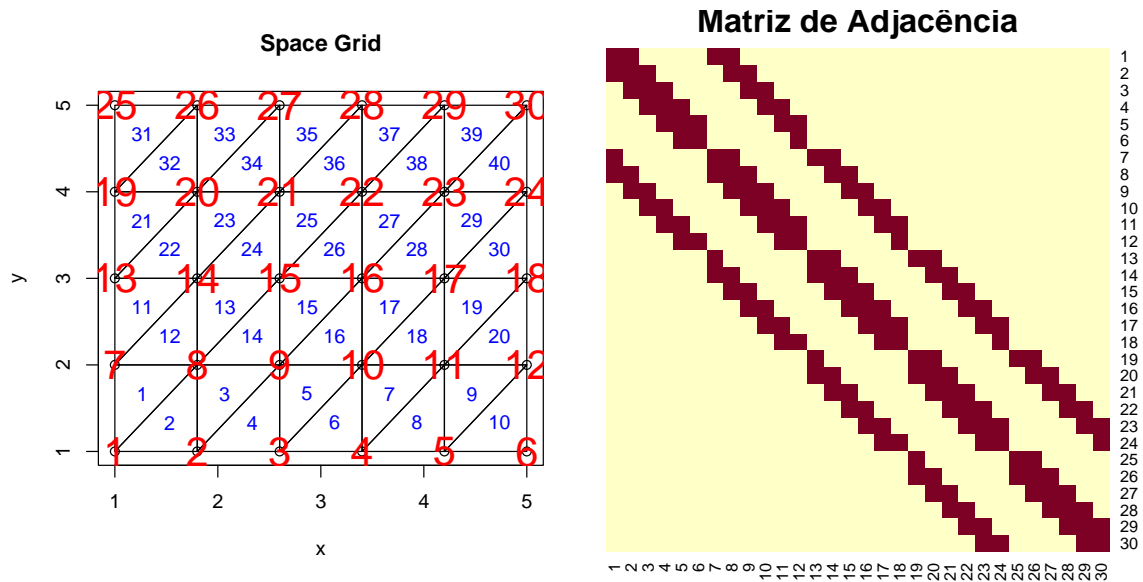
```
triangled <- get_triangles(8, 8, FALSE)
adj <- create_adj_matrix(triangled)
plot_mesh_grid(triangled)
plot_adj_matrix(adj)
```



Ok, funciona! Mas e para tamanhos de linha e coluna distintos?

Vamos testar para uma matriz com 5 linhas e 6 colunas.

```
triangled <- get_triangles(5, 6, FALSE)
adj <- create_adj_matrix(triangled)
plot_mesh_grid(triangled)
plot_adj_matrix(adj)
```



E agora, modificando a ordem.

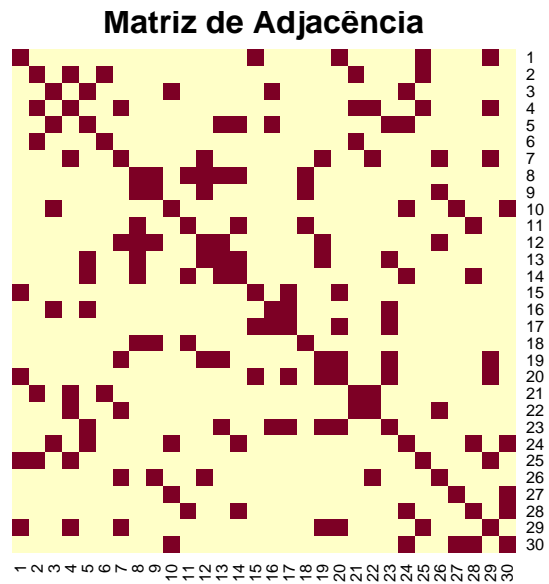
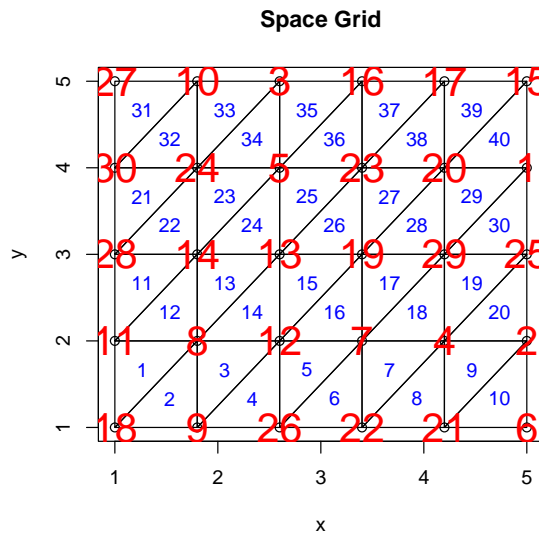
```
triangled <- get_triangles(5, 6, TRUE)
adj <- create_adj_matrix(triangled)
plot_mesh_grid(triangled)
plot_adj_matrix(adj)
```

Figuras mais desordenadas (Delaunay)

```
library("deldir")
```

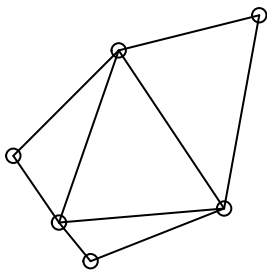
Veja a importação da biblioteca “deldir” acima, utilizada para triangulizar uma figura.

Abaixo, as coordenadas de x e y diretamente escritas no código.



```
x <- c(2.3,3.2,7.0,1.0,4.0,8.0)
y <- c(3.1,2.0,3.5,5.0,8.0,9.0)

dxy2 <- deldir(x,y,plot=TRUE,wl="tr")
```



A figura acima é o plot gerado pela biblioteca “deldir”. Porém, é possível fazer uma plotagem melhor:

A partir daqui, temos alguns trabalhos:

1. Plotar os vértices
2. Obter o índice de cada vértice

```
x1 <- dxy2[["delsgs"]][["x1"]]
x2 <- dxy2[["delsgs"]][["x2"]]
y1 <- dxy2[["delsgs"]][["y1"]]
y2 <- dxy2[["delsgs"]][["y2"]]
```

```

ind1 <- dxy2[["delsgs"]][["ind1"]]
ind2 <- dxy2[["delsgs"]][["ind2"]]

len <- length(x1)

# Plotando os vértices
plot(x, y)

# Plotando as linhas
for (i in 1:len) {
  polygon(c(x1[i], x2[i]), c(y1[i], y2[i]), border = "red")
}

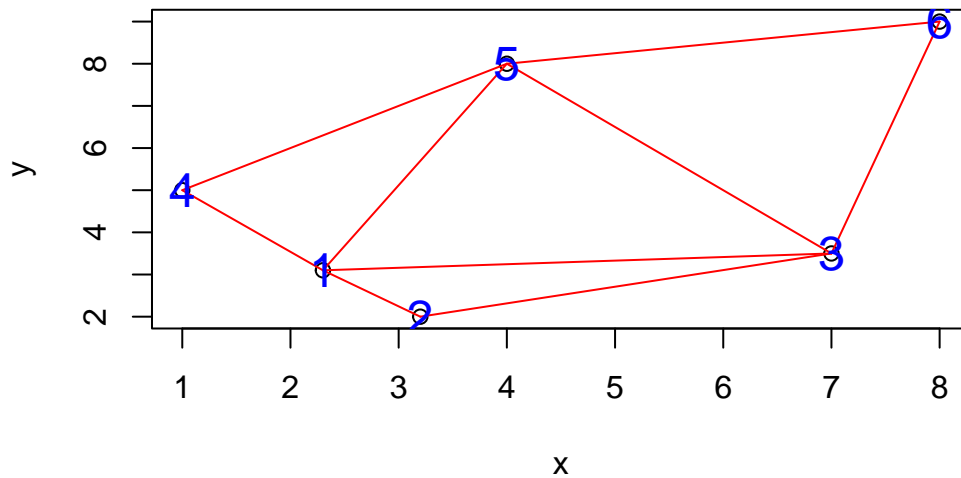
# Matriz que terá os índices
indexes <- matrix(0, length(x), 2)

# Lendo da primeira lista
for (i in 1:len) {
  indexes[ind1[i], 1] <- x1[i]
  indexes[ind1[i], 2] <- y1[i]
}

# Lendo da segunda lista
for (i in 1:len) {
  indexes[ind2[i], 1] <- x2[i]
  indexes[ind2[i], 2] <- y2[i]
}

# Plotando os índices
for (i in 1:length(x)) {
  text(indexes[i, 1], indexes[i, 2], i, col="blue", cex=1.5)
}

```



Gerando a Matriz de Adjacências

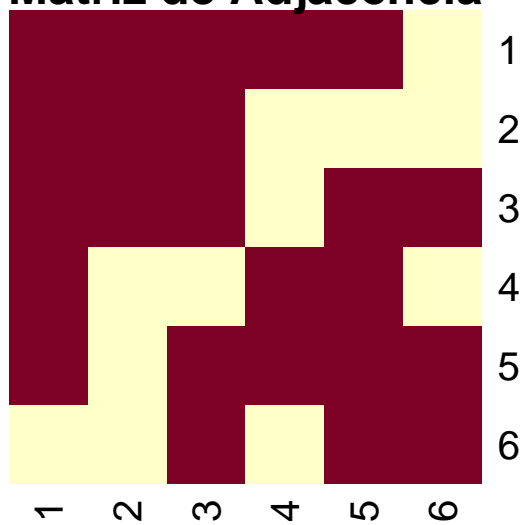
```
adj <- matrix(0, length(x), length(y))

for (i in 1:len) {
  adj[ind1[i], ind2[i]] <- 1
  adj[ind2[i], ind1[i]] <- 1

  adj[ind1[i], ind1[i]] <- 1
  adj[ind2[i], ind2[i]] <- 1
}

heatmap(adj, Colv = "Rowv", Rowv = NA,
        symm=TRUE, main="Matriz de Adjacência")
```

Matriz de Adjacência



Finalmente, a matriz de adjacência montada.