# 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:

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

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

### 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.

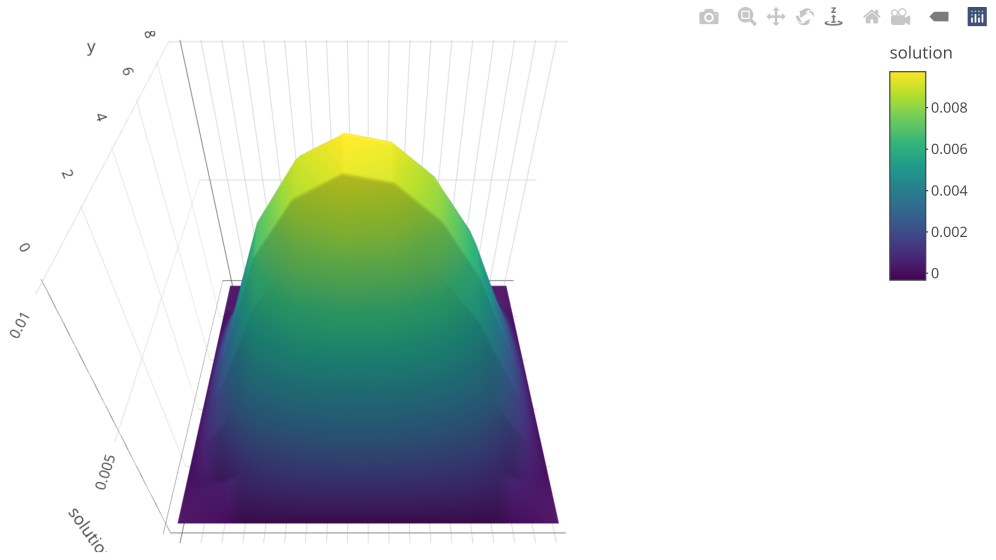
```
x <- seq(0, L, d)
y <- seq(0, L, d)

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

### Plotagem do Gráfico

Finalmente, plotamos o gráfico.

```
ploting_mesh <- function(solution, to_pdf=FALSE) {
  fig <- plot_ly(z = ~solution)
  fig <- fig %>% add_surface()

  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)
  }
}
```

```r
ploting_mesh(z, to_pdf = TRUE)
```




## Grids e Grafos

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

### Código necessário

```r
# 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(order) {
  mtx <- t(matrix(linspace(0, 1, order), order, order))
  mty <- matrix(linspace(1, 0, order), order, order)

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

```r
# Recebe a ordem de uma matriz e devolve
# um espaço de triângulos
get_triangles <- function(order, ramdomize_vertices=FALSE) {
  grid <- generate_gen_matrices(order) # 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, order, order)   # - Para o Grid com os
                                        # vértices numerados

  V <- rep(1:(order*order))
  if (ramdomize_vertices == TRUE) {
    set.seed(245)
    V <- sample(V)
  }

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

  # Fazendo a triangulação
  for (i in (order):(2)) {
    for (j in 1:(order-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[order - i + 1, j],
          vertices[order - i + 2, j],
          vertices[order - i + 2, j+1]))
      triangles_v <- rbind(
        triangles_v,
        c(vertices[order - i + 1, j],
          vertices[order - i + 1, j+1],
          vertices[order - 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$trx      == coordinates in x axis
## triangled_grid$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
```

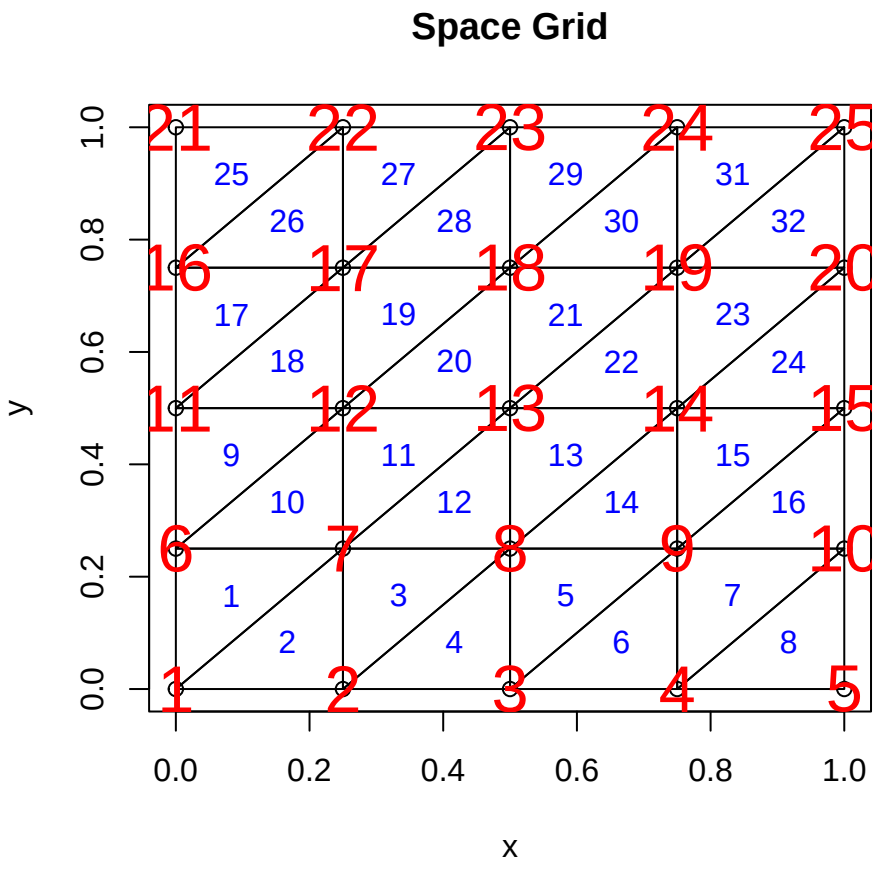
```r
plot_adj_matrix <- function(adj_matrix) {
  heatmap(adj_matrix, Colv = "Rowv", Rowv = NA,
          symm=TRUE, main="Matriz de Adjacência")
}
```

### 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.

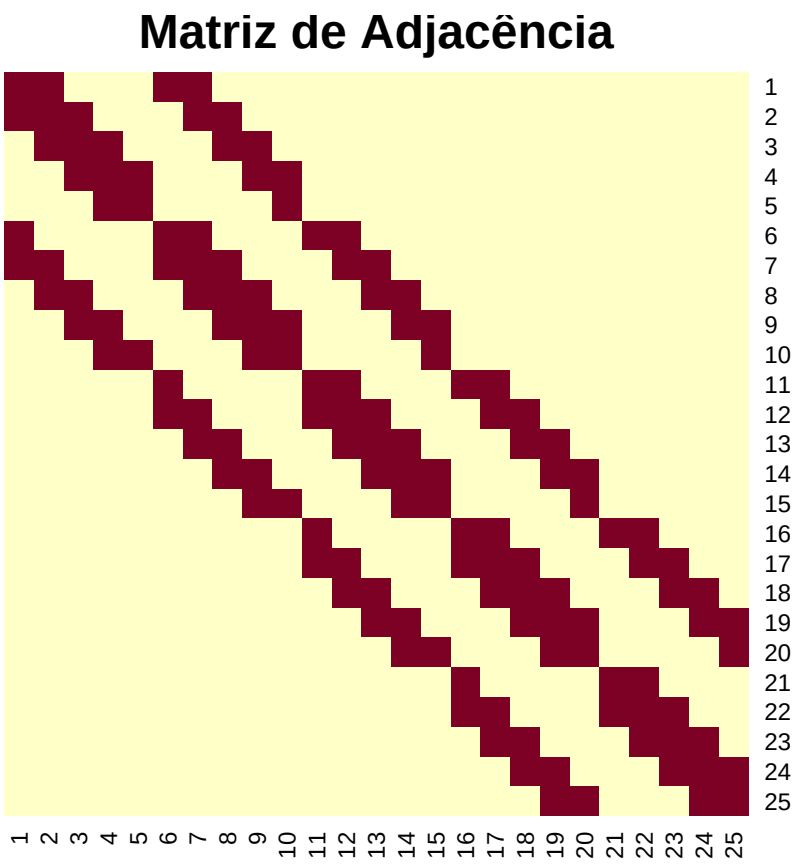
```r
# Gera uma matriz com vértices ordenados e faz a triangulação
triangled <- get_triangles(5, FALSE)
plot_mesh_grid(triangled)
```



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)
```



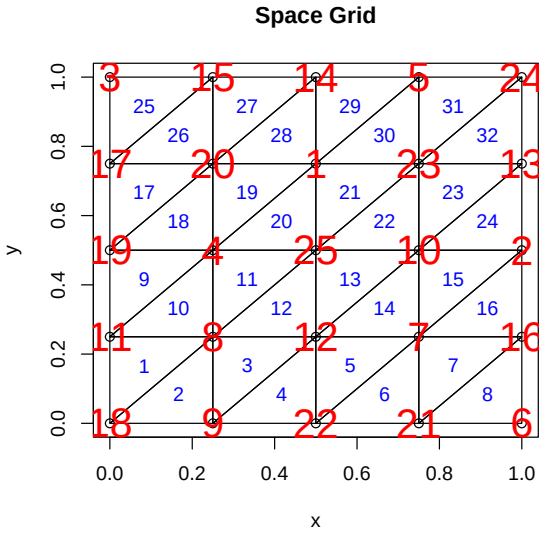

Essa é a matriz gerada.

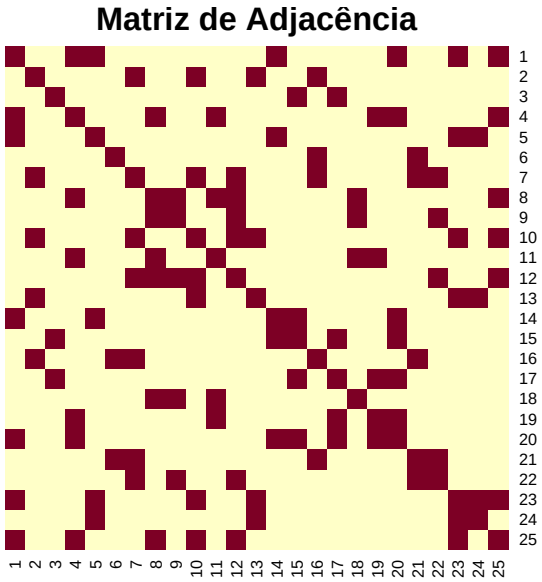
### Trocando a ordem dos vértices

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

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

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




De fato, a matriz é gerada corretamente!

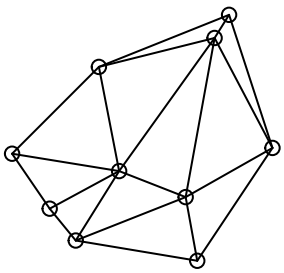
## Figuras mais desordenadas (Delaunay)

```
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, 10.0, 4.7, 8.5, 7.4)
y <- c(3.1,2.0,3.5,5.0,8.0,9.0, 5.2, 4.4, 9.8, 1.3)

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



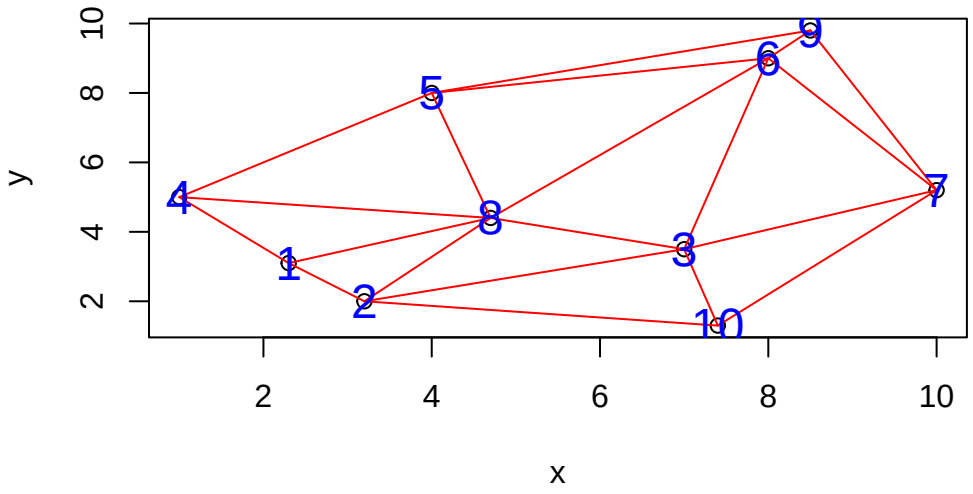
A figura acima é o plot gerado pela biblioteca “deldir”. Porém, é possível fazer uma plotagem melhor:

A partir daqui, temos alguns trabalhos:

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

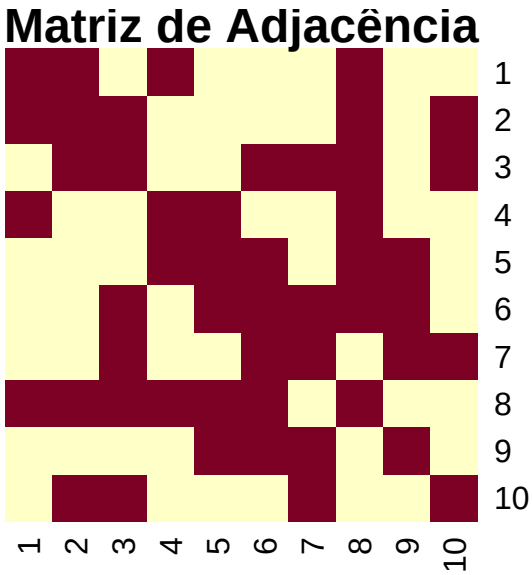
```
x1 <- dxy2[["delsgs"]][["x1"]]
x2 <- dxy2[["delsgs"]][["x2"]]
y1 <- dxy2[["delsgs"]][["y1"]]
y2 <- dxy2[["delsgs"]][["y2"]]

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]
}
```

```r
# 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

```r
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")
```

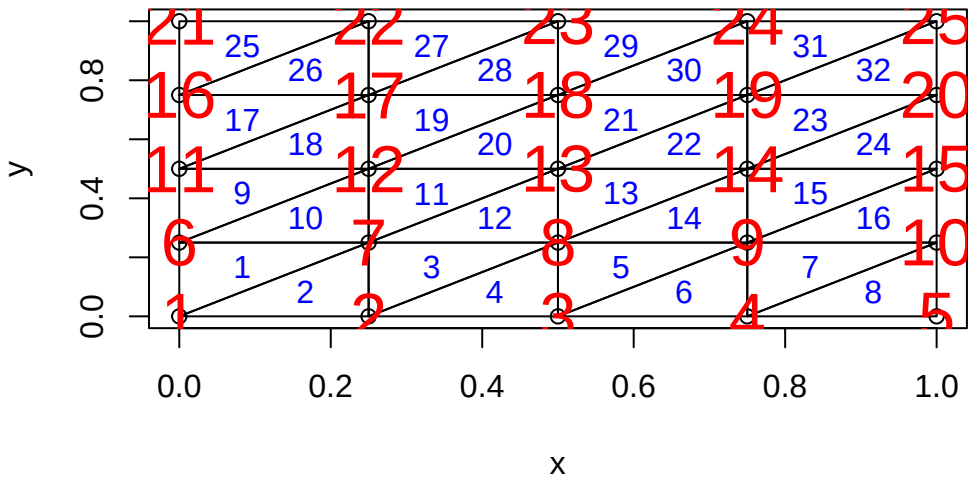



Finalmente, a matriz de adjacência montada.

## Método dos Elementos Finitos

Vamos começar de forma bem simples, iniciando com a malha quadrada de lado 1. Ali, as áreas dos triângulos são sempre as mesmas. Calculemos:

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




Perceba que para qualquer figira gerada o tamanho dos catetos sempre será igual a 1, a área se dá por:

$$A = \frac{1}{2} \times \text{base} \times \text{altura} A = \frac{1}{2} \times 1 \times 1 = \frac{1}{2}$$

Vamos gerar, também, uma função que receba um triângulo e monte uma matriz, que será a matriz B.

```
B_local <- function(coord_X, coord_Y, A) {
  ## Aqui, consideramos que os vértices do
  ## triangulo estão em sentido anti-horário

  B <- matrix(0, 2, 3)
  B[1, 1] <- coord_Y[3] - coord_Y[2]
  B[1, 2] <- coord_Y[2] - coord_Y[1]
  B[1, 3] <- coord_Y[1] - coord_Y[3]

  B[2, 1] <- coord_X[2] - coord_X[3]
  B[2, 2] <- coord_X[1] - coord_X[2]
  B[2, 3] <- coord_X[3] - coord_X[1]

  B <- B / (2 * A)

  return(B)
}
```

Agora, uma função que calcule o K_local em um triângulo.

```
K_local <- function(coord_X, coord_Y, A, k = matrix(c(1, 0, 0, 1), 2, 2)) {
  B <- B_local(coord_X, coord_Y, A)
  K <- (t(B) %*% k %*% B) * A

  return(K)
}
```

Finalmente uma que junte tudo.

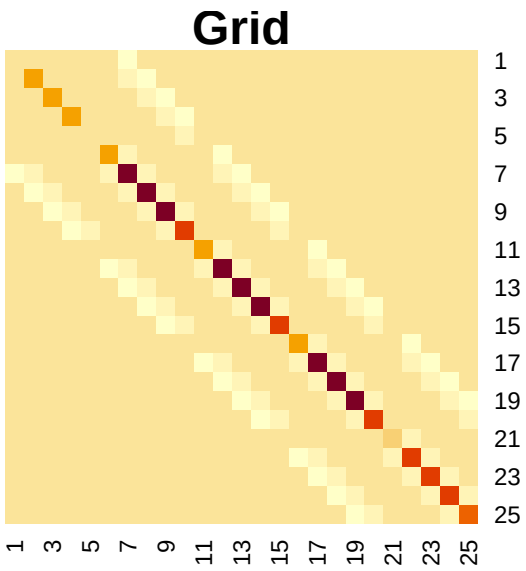
```
grid_final <- function(triangled) {
  x_axes <- triangled$x_axis
  y_axes <- triangled$y_axis
  Xx <- triangled$mx
  vertices <- triangled$mv

  n_triangles <- nrow(x_axes)

  grid_dim_x <- nrow(Xx)
  grid_dim_y <- ncol(Xx)

  grid <- matrix(0, grid_dim_x*grid_dim_x, grid_dim_y*grid_dim_y)

  for (i in 1:n_triangles) {
    X <- x_axes[i, ]
    Y <- y_axes[i, ]
    V <- vertices[i, ]

    if (i %% 2 == 0) {
      x <- c(X[1], X[2], X[3])
      y <- c(Y[1], Y[2], Y[3])
      v <- c(V[1], V[2], V[3])
    } else {
      x <- c(X[1], X[3], X[2])
      y <- c(Y[1], Y[3], Y[2])
      v <- c(V[1], V[3], V[2])
    }

    vx <- c(0, 0, 0)
    vy <- c(0, 0, 0)
```

```
    A <- (1/2) * (x[1] * y[2] + x[2] * y[3] + x[3] * y[1]
                  - x[2] * y[1] - x[3] * y[2] - x[1] * y[3])

    k_loc <- K_local(x, y, A)

    for (j in 1:3) {
      for (k in 1:3) {
        grid[v[k], v[j]] <- grid[v[k], v[j]] + k_loc[k, j]
      }
    }
  }

  for (i in 1:grid_dim_x) {
    for (j in 1:grid_dim_x) {
      if (i == 1 || i == grid_dim_x || j == 1 || j == grid_dim_x) {
        grid[i, j] <- 0
      }
    }
  }

  return(grid)
}
```

Agora, apliquemos:

```
triangled <- get_triangles(5, FALSE)
grid <- grid_final(triangled)
```

```
heatmap(grid, Colv = "Rowv", Rowv = NA,
          symm=TRUE, main="Grid")
```


Grid
1
3
5
7
9
11
13
15
17
19
21
23
25
1
3
5
7
9
11
13
15
17
19
21
23
25