Trabalho Prático 3: Comparação de Configurações do Algoritmo DE

José Joaquim de Andrade Neto, Matheus Paiva Loures, Raphael Anderson Da Silva

16 de Dezembro de 2024

Descrição do Problema

Este estudo de caso investiga o desempenho de duas configurações do algoritmo de Evolução Diferencial (DE) na resolução de problemas de otimização baseados na função de Rosenbrock, com dimensões variando de 2 a 150. O objetivo é avaliar a eficiência e a robustez das configurações testadas em diferentes escalas de complexidade do problema.

1. Configurações Testadas:

- Configuração 1: `recombination_lbga` e `mutation_rand` com fator de escala `f = 4.5`.
- Configuração 2: `recombination_blxAlphaBeta` ($= 0.1$, $= 0.4$) e `mutation_rand` com fator de escala `f = 3`.

2. Problema de Otimização: A função de Rosenbrock, amplamente utilizada como benchmark, gerada com o pacote `smoof`.

3. Parâmetros do Algoritmo:

- Tamanho da População: `popsize = 5 * dim` (proporcional à dimensão do problema).
- Critérios de Parada: Avaliações máximas `maxevals = 5000 * dim` e iterações máximas `maxiter = 100 * dim`.

4. Análise de Resultados: Os desempenhos médios das configurações foram comparados em diferentes dimensões utilizando métodos estatísticos, a fim de identificar diferenças significativas e inferir a melhor configuração para cenários variados.

Este estudo visa contribuir para a compreensão das implicações práticas das escolhas de recombinação e mutação no algoritmo DE, considerando a escalabilidade em problemas de otimização.

Metodologia

Teste Esploratorio

```r
# Definir a função de Rosenbrock
dim <- 16  # Dimensão do problema
fn <- function(X) {
  if (!is.matrix(X)) X <- matrix(X, nrow = 1)  # Garantir que X seja uma matriz
  apply(X, MARGIN = 1, FUN = smoof::makeRosenbrockFunction(dimensions = dim))
}

# Configurações
mutpars1 <- list(name = "mutation_rand", f = 4.5)
recpars1 <- list(name = "recombination_lbga")
```

```r
mutpars2 <- list(name = "mutation_rand", f = 3)
recpars2 <- list(name = "recombination_blxAlphaBeta", alpha = 0.1, beta = 0.4)

# Parâmetros do problema
popsize <- 5 * dim
probpars <- list(name = "fn", xmin = rep(-5, dim), xmax = rep(10, dim))
selpars <- list(name = "selection_standard")
stopcrit <- list(names = "stop_maxeval", maxevals = 5000 * dim)

# Execução Config 1
resultados_config1 <- ExpDE(
  mutpars = mutpars1, recpars = recpars1, popsize = popsize,
  selpars = selpars, stopcrit = stopcrit, probpars = probpars
)

# Execução Config 2
resultados_config2 <- ExpDE(
  mutpars = mutpars2, recpars = recpars2, popsize = popsize,
  selpars = selpars, stopcrit = stopcrit, probpars = probpars
)

# Verificar os resultados
print(resultados_config1$Fbest)
```

```
## [1] 1824.752
```

```r
print(resultados_config2$Fbest)
```

```
## [1] 56458.12
```

Os resultados obtidos indicam que a primeira configuração do algoritmo, composta pela recombinação recombination_lbga e mutação mutation_rand com fator 4.5, pode apresentar um desempenho significativamente melhor em relação à segunda configuração, que utilizou a recombinação recombination_blxAlphaBeta e mutação mutation_rand com fator 3. Esse indicativo é evidente pelos valores da função objetivo, com a primeira configuração atingindo um valor final consideravelmente inferior ao resultado obtido pela segunda, ressaltando sua maior eficácia na minimização da função de Rosenbrock no cenário analisado.

Execução do Algoritmo para Diversas Dimensões

```r
dim_range <- 2:150
csv_file <- "resultados_dimensoes_corrigido.csv"

if (!file.exists(csv_file)) {
  results_list <- list()
  for (dimesion in dim_range) {
    # Definir a função de Rosenbrock
    fn <- function(X) {
      if (!is.matrix(X)) X <- matrix(X, nrow = 1)  # Garantir que X seja uma matriz
      apply(X, MARGIN = 1, FUN = smoof::makeRosenbrockFunction(dimensions = dimesion))
    }

    # Configurações
    mutpars1 <- list(name = "mutation_rand", f = 4.5)
    recpars1 <- list(name = "recombination_lbga")
```

```
    mutpars2 <- list(name = "mutation_rand", f = 3)
    recpars2 <- list(name = "recombination_blxAlphaBeta", alpha = 0.1, beta = 0.4)

    # Parâmetros do problema
    popsize <- 5 * dimesion
    probpars <- list(name = "fn", xmin = rep(-5, dimesion), xmax = rep(10, dimesion))
    selpars <- list(name = "selection_standard")
    stopcrit <- list(names = "stop_maxeval", maxevals = 5000 * dimesion)

    # Execução Config 1
    resultados_config1 <- ExpDE(
      mutpars = mutpars1, recpars = recpars1, popsize = popsize,
      selpars = selpars, stopcrit = stopcrit, probpars = probpars
    )

    # Execução Config 2
    resultados_config2 <- ExpDE(
      mutpars = mutpars2, recpars = recpars2, popsize = popsize,
      selpars = selpars, stopcrit = stopcrit, probpars = probpars
    )

    # Verificar os resultados
    print(resultados_config1$Fbest)
    print(resultados_config2$Fbest)

    results_list[[length(results_list) + 1]] <- data.frame(
      Dimension = dimesion,
      Best_Config1 = resultados_config1$Fbest,
      Best_Config2 = resultados_config2$Fbest
    )

  }
  # Combinar todos os resultados e salvar em CSV
  results_df <- do.call(rbind, results_list)
  write.csv(results_df, csv_file, row.names = FALSE)
}

results <- read.csv(csv_file)
head(results)
```

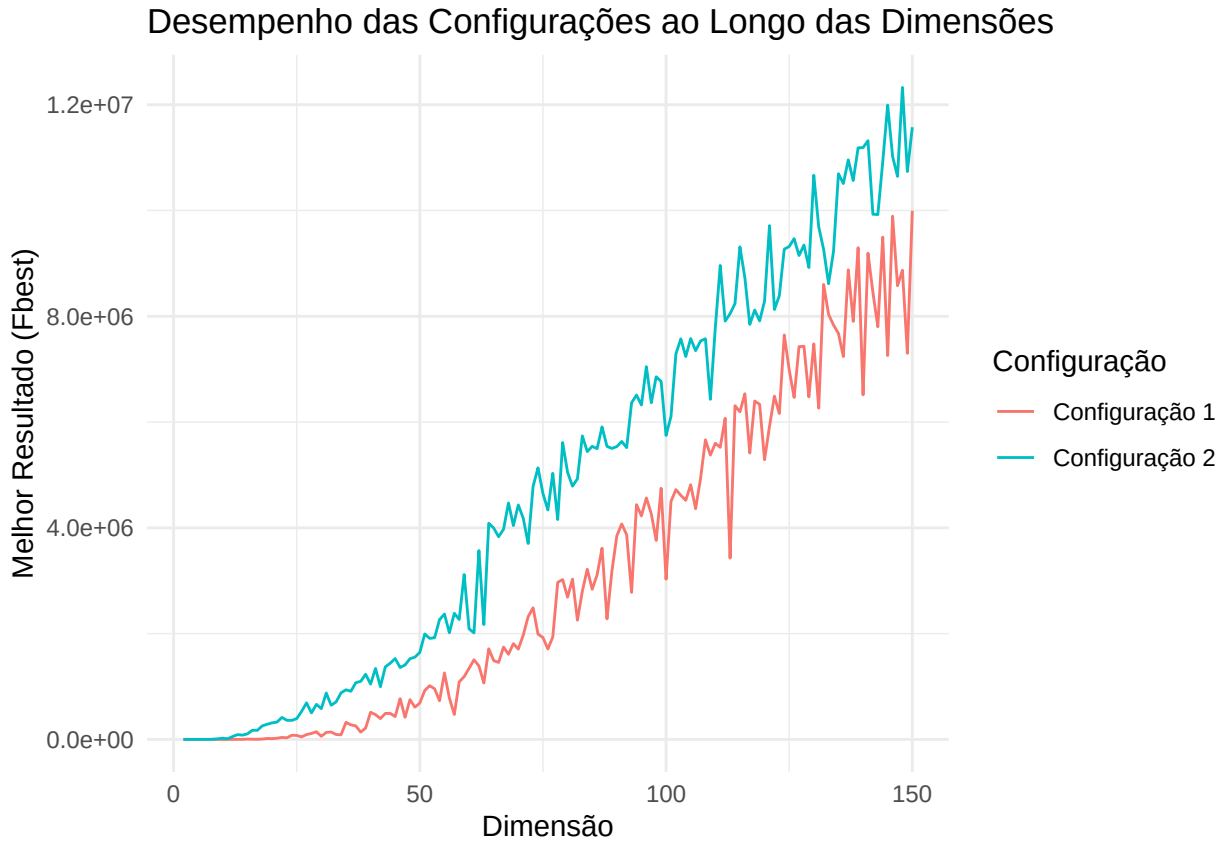
```
##   Dimension Best_Config1 Best_Config2
## 1         2 1.543878e-07 2.040463e-05
## 2         3 4.244184e-02 3.990894e-01
## 3         4 4.451806e-01 2.787668e+00
## 4         5 6.419616e-01 2.925271e+01
## 5         6 8.505534e+00 2.385819e+02
## 6         7 2.459027e+01 5.263520e+02
```

Visualização dos Resultados

Criar um gráficos para observar tendências nas duas configurações ao longo das dimensões do problema.

```
# Gráfico de linha comparando os desempenhos
ggplot(results, aes(x = Dimension)) +
```

```
  geom_line(aes(y = Best_Config1, color = "Configuração 1")) +
  geom_line(aes(y = Best_Config2, color = "Configuração 2")) +
  labs(title = "Desempenho das Configurações ao Longo das Dimensões",
       x = "Dimensão",
       y = "Melhor Resultado (Fbest)",
       color = "Configuração") +
  theme_minimal()
```



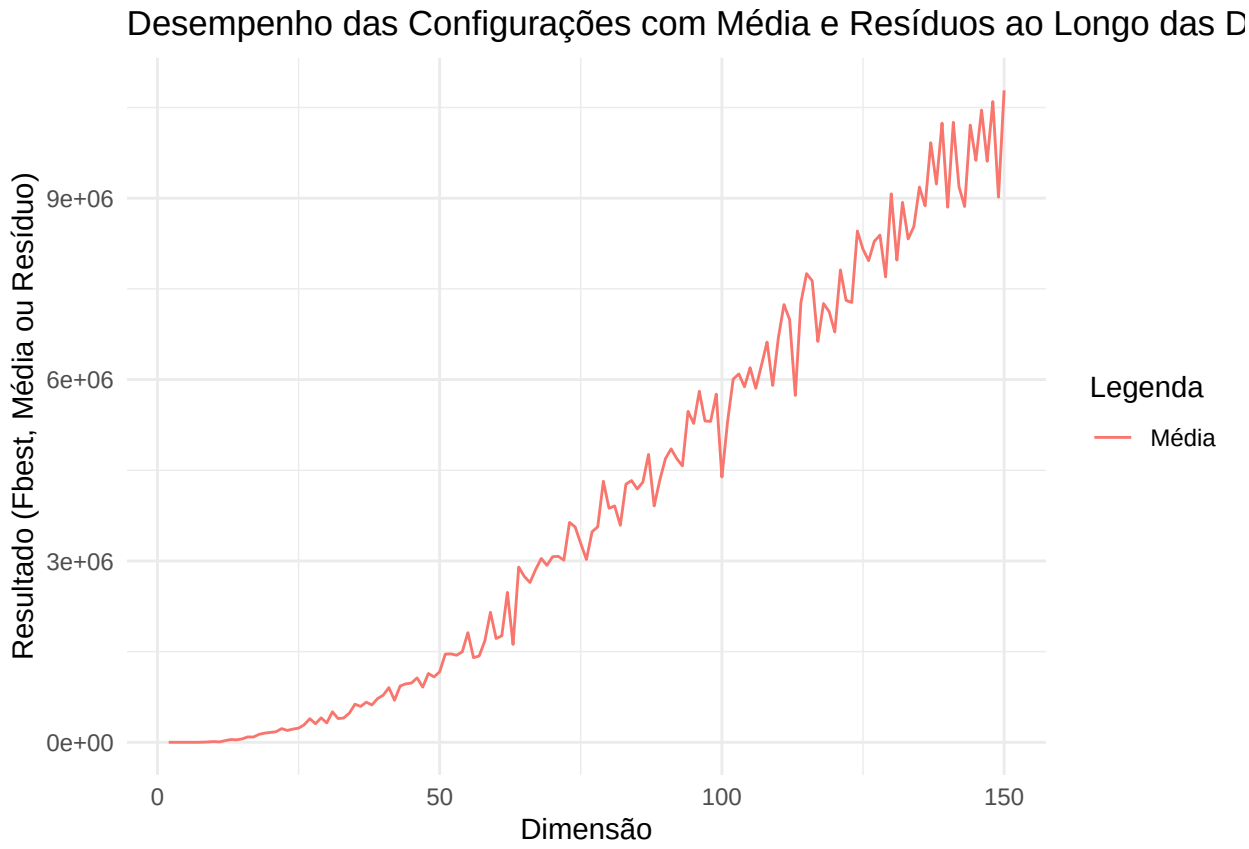
Observa-se que, de maneira geral, a Configuração 1 (recombination_lbga e mutation_rand com fator 4.5) apresenta melhores resultados em comparação com a Configuração 2 (recombination_blxAlphaBeta e mutation_rand com fator 3) em todas as dimensões avaliadas. O desempenho das duas configurações degrada progressivamente com o aumento da dimensão, o que era esperado devido à maior complexidade do problema.

```
# Adicionar novas colunas ao DataFrame
adicionar_colunas <- function(data) {
  # Calcular a média das colunas Best_Config1 e Best_Config2
  data$Mean_Result <- rowMeans(data[, c("Best_Config1", "Best_Config2")])

  # Calcular os resíduos para cada configuração
  data$Residual_Config1 <- data$Best_Config1 - data$Mean_Result
  data$Residual_Config2 <- data$Best_Config2 - data$Mean_Result

  return(data)
}

# Carregar os resultados existentes
```

```r
results <- read.csv(csv_file)

# Aplicar a função para adicionar novas colunas
results <- adicionar_colunas(results)
head(results)
```

```
##   Dimension Best_Config1 Best_Config2  Mean_Result Residual_Config1
## 1         2 1.543878e-07 2.040463e-05 1.027951e-05    -1.012512e-05
## 2         3 4.244184e-02 3.990894e-01 2.207656e-01    -1.783238e-01
## 3         4 4.451806e-01 2.787668e+00 1.616424e+00    -1.171244e+00
## 4         5 6.419616e-01 2.925271e+01 1.494734e+01    -1.430538e+01
## 5         6 8.505534e+00 2.385819e+02 1.235437e+02    -1.150382e+02
## 6         7 2.459027e+01 5.263520e+02 2.754711e+02    -2.508809e+02
##   Residual_Config2
## 1     1.012512e-05
## 2     1.783238e-01
## 3     1.171244e+00
## 4     1.430538e+01
## 5     1.150382e+02
## 6     2.508809e+02
```

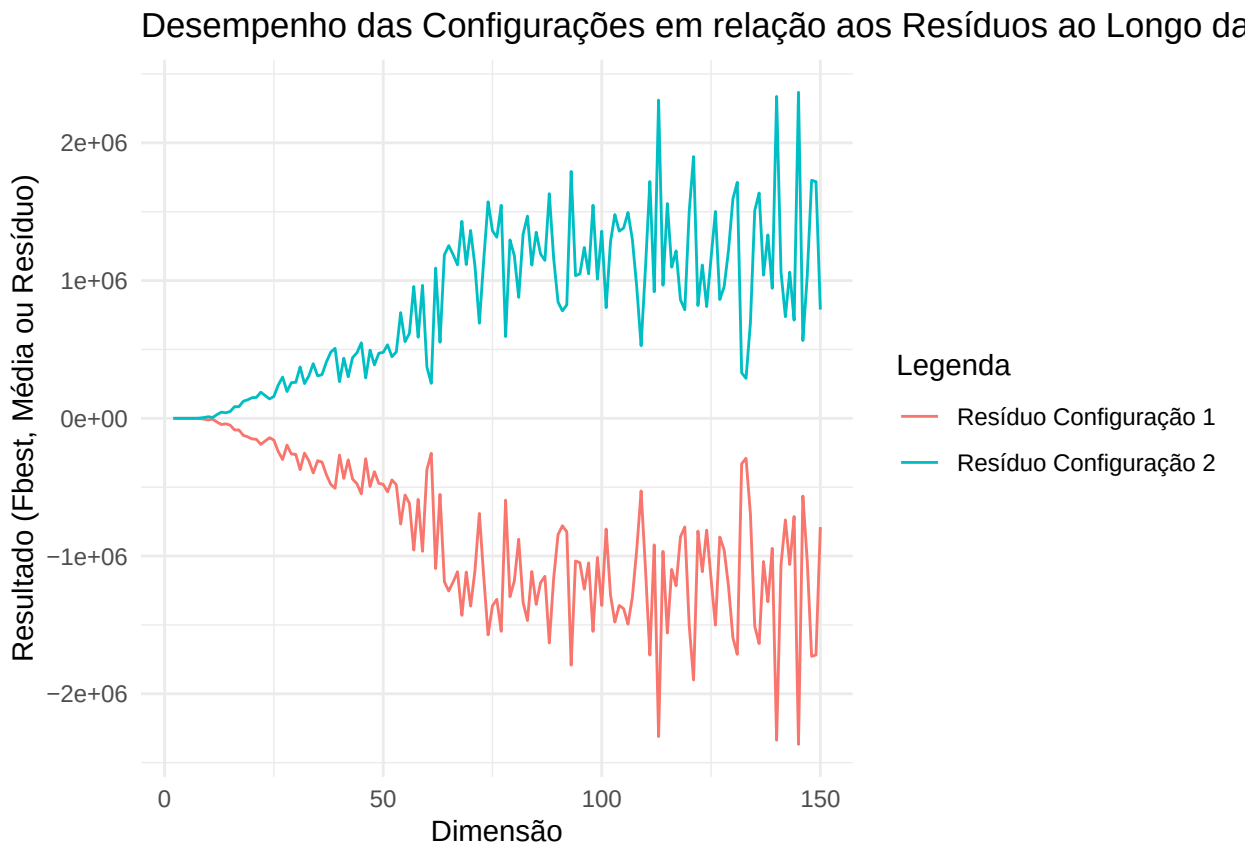
```r
ggplot(results, aes(x = Dimension)) +
  geom_line(aes(y = Mean_Result, color = "Média")) +
  labs(
    title = "Desempenho das Configurações com Média e Resíduos ao Longo das Dimensões",
    x = "Dimensão",
    y = "Resultado (Fbest, Média ou Resíduo)",
    color = "Legenda"
  ) +
  theme_minimal()
```



A análise do gráfico apresentado demonstra que a média dos resultados dos algoritmos aumenta proporcionalmente ao número de dimensões do problema, indicando uma dependência direta entre o desempenho do algoritmo e a complexidade dimensional da função otimizada. Essa relação reflete a escalabilidade dos algoritmos avaliados, já que o aumento da dimensão torna a tarefa de encontrar soluções ótimas mais desafiadora, elevando os valores médios da função objetivo.

Para determinar qual algoritmo possui um desempenho superior, é essencial considerar a diferença entre as médias das configurações testadas. Apenas ao verificar essa diferença é possível avaliar de maneira conclusiva a eficácia relativa de cada algoritmo, pois a média reflete o desempenho médio em todos os testes realizados. Sem essa análise comparativa das médias, qualquer inferência sobre a superioridade de uma configuração seria inadequada e sujeita a vieses, ignorando o contexto da variabilidade dos resultados.

```
ggplot(results, aes(x = Dimension)) +
  geom_line(aes(y = Residual_Config1, color = "Resíduo Configuração 1")) +
  geom_line(aes(y = Residual_Config2, color = "Resíduo Configuração 2")) +
  labs(
    title = "Desempenho das Configurações em relação aos Resíduos ao Longo das Dimensões",
    x = "Dimensão",
    y = "Resultado (Fbest, Média ou Resíduo)",
    color = "Legenda"
  ) +
  theme_minimal()
```



A análise da evolução dos resíduos apresentados no gráfico revela diferenças consistentes entre as configurações ao longo das dimensões. Observa-se que os resíduos da Configuração 1 apresentam valores predominantemente negativos, enquanto os da Configuração 2 são majoritariamente positivos. Isso indica que, em média, a Configuração 1 obteve resultados melhores (mais próximos ao valor mínimo da função objetivo) em comparação com a Configuração 2.

Cálculo do Tamanho Amostral

```r
pwr.t.test(d = 0.5, sig.level = 0.05, power = 0.8, type = "paired")
```

```
##
##      Paired t test power calculation
##
##               n = 33.36713
##               d = 0.5
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE: n is number of *pairs*
```

O cálculo do tamanho amostral indica que, para realizar um teste t pareado com uma diferença de efeito (d) de 0,5, um nível de significância de 5% (sig.level = 0,05) e um poder estatístico de 80% (power = 0,8), seriam necessárias 33,37 observações pareadas. Como o número de pares deve ser inteiro, é adequado arredondar para 34 pares. Esse resultado assegura que o teste t terá uma probabilidade de 80% de detectar um efeito de tamanho moderado (d = 0,5), caso ele exista, minimizando tanto o risco de erro tipo I (falso positivo) quanto o risco de erro tipo II (falso negativo).

Execução Paralela do Experimento

Nesse momento é realizado os experimentos para as duas configurações do algoritmo de Evolução Diferencial (DE) para diferentes dimensões de problemas de otimização, armazenar os resultados em um arquivo CSV e paralelizar o processamento para maior eficiência. Inicialmente, ele verifica se o arquivo de resultados já existe; caso contrário, cria um novo arquivo e prepara um cabeçalho detalhado. Um cluster paralelo é configurado para processar múltiplas dimensões simultaneamente, realizando 30 execuções para cada configuração por dimensão. Os resultados são salvos incrementalmente no arquivo CSV, incluindo os valores individuais das execuções, as médias por configuração e a média geral. Dimensões que apresentarem erros durante o processamento são registradas para análise posterior. Este método eficiente garante a reprodutibilidade, escalabilidade e organização dos dados.

```r
library(parallel)

# Inicializar a lista para armazenar dimensões com erros
error_dimensions <- list()

csv_file <- "resultados_dimensoes_corrigido_completo_v2.csv"

# Criar o arquivo CSV com cabeçalho, se ele não existir
if (!file.exists(csv_file)) {
  n <- 30
  dim_Total <- 2:150
  # Definir semente para garantir reprodutibilidade
  set.seed(123)
  dim_range <- sort(sample(dim_Total, size = 34, replace = FALSE), decreasing = FALSE)
  print(dim_range)

  # Estruturar o cabeçalho completo
  colunas_config1 <- paste0("Config1_Run_", 1:n)   # Colunas para a Configuração 1
  colunas_config2 <- paste0("Config2_Run_", 1:n)   # Colunas para a Configuração 2
  colunas_finais <- c("Mean_Config1", "Mean_Config2", "Mean_General")  # Médias
  colunas_totais <- c("Dimension", colunas_config1, colunas_config2, colunas_finais)

  write.csv(data.frame(matrix(ncol = length(colunas_totais), nrow = 0, dimnames = list(NULL, colunas_to
            csv_file, row.names = FALSE)

  # Criar o cluster uma vez
  n_cores <- max(1, detectCores() - 2)  # Usar quase todos os núcleos disponíveis
  cl <- makeCluster(n_cores)
  on.exit(stopCluster(cl))  # Garantir que o cluster seja fechado no final

  # Lista para armazenar dimensões com erros
  error_dimensions <- list()

  for (dimension in dim_range) {
    print(paste("Processando dimensão:", dimension))

    tryCatch({
      # Definir a função de Rosenbrock
      fn <- function(X) {
        if (!is.matrix(X)) X <- matrix(X, nrow = 1)
        apply(X, MARGIN = 1, FUN = smoof::makeRosenbrockFunction(dimensions = dimension))
      }
```

```r
    # Configurações
    mutpars1 <- list(name = "mutation_rand", f = 4.5)
    recpars1 <- list(name = "recombination_lbga")
    mutpars2 <- list(name = "mutation_rand", f = 3)
    recpars2 <- list(name = "recombination_blxAlphaBeta", alpha = 0.1, beta = 0.4)

    # Parâmetros do problema
    popsize <- 5 * dimension
    probpars <- list(name = "fn", xmin = rep(-5, dimension), xmax = rep(10, dimension))
    selpars <- list(name = "selection_standard")
    stopcrit <- list(names = "stop_maxeval", maxevals = 5000 * dimension)

    # Exportar as variáveis necessárias para o cluster
    clusterExport(cl, varlist = c("ExpDE", "mutpars1", "recpars1", "mutpars2", "recpars2",
                                  "popsize", "selpars", "stopcrit", "probpars", "fn", "dimension"))

    # Função para executar as configurações paralelamente
    execute_exp <- function(i) {
      resultados_config1 <- ExpDE(mutpars = mutpars1, recpars = recpars1, popsize = popsize,
                                  selpars = selpars, stopcrit = stopcrit, probpars = probpars)
      resultados_config2 <- ExpDE(mutpars = mutpars2, recpars = recpars2, popsize = popsize,
                                  selpars = selpars, stopcrit = stopcrit, probpars = probpars)
      return(list(Config1 = resultados_config1$Fbest, Config2 = resultados_config2$Fbest))
    }

    # Executar paralelamente os experimentos
    results_parallel <- parLapply(cl, 1:n, execute_exp)

    # Separar os resultados em vetores
    results_config1 <- sapply(results_parallel, function(x) x$Config1)
    results_config2 <- sapply(results_parallel, function(x) x$Config2)

    # Calcular as médias
    mean_config1 <- mean(results_config1)
    mean_config2 <- mean(results_config2)
    mean_general <- mean(c(results_config1, results_config2))

    # Montar os resultados como uma linha do DataFrame
    result_row <- data.frame(
      Dimension = dimension,
      as.list(setNames(results_config1, colunas_config1)),
      as.list(setNames(results_config2, colunas_config2)),
      Mean_Config1 = mean_config1,
      Mean_Config2 = mean_config2,
      Mean_General = mean_general
    )

    # Adicionar ao arquivo CSV
    write.table(result_row, csv_file, sep = ",", col.names = FALSE, row.names = FALSE, append = TRUE)

  }, error = function(e) {
    message(paste("Erro na dimensão:", dimension, " - ", e))
    error_dimensions[[length(error_dimensions) + 1]] <- list(dimension = dimension, error = e)
```

```
    })
  }
  # Fechar o cluster
  stopCluster(cl)
}

# Carregar os resultados do arquivo CSV para verificar
results <- read.csv(csv_file)
print(head(results))
```

```
##   Dimension Config1_Run_1 Config1_Run_2 Config1_Run_3 Config1_Run_4
## 1         8      13.14408      30.91694      43.68942      45.64571
## 2        10     123.81571     163.09119     201.31434     224.43417
## 3        15    3774.04976     613.86797    3055.06119    3249.58476
## 4        16    4030.04402    1639.18024    6346.23919    4346.10104
## 5        24   65344.99778   24377.26364   27946.72583   41450.22093
## 6        27   67539.33353   56596.69376  115492.82639   40517.18225
##   Config1_Run_5 Config1_Run_6 Config1_Run_7 Config1_Run_8 Config1_Run_9
## 1      30.40451      41.27377      38.96291      16.91283       55.4681
## 2      74.66324     257.10150     103.47188      88.70755      194.8138
## 3    3321.61787    2088.49935    4892.06166    2927.77223     2579.1435
## 4    2190.77388    3567.21640    3800.78123    2960.52597     2168.9358
## 5   39757.52651   46714.61983   49315.33653   49115.46674    82393.7289
## 6  154924.20877  163227.43160  124707.58402   72843.34334   126083.4019
##   Config1_Run_10 Config1_Run_11 Config1_Run_12 Config1_Run_13 Config1_Run_14
## 1        37.5330       187.3890       24.01504       29.10099       63.75398
## 2       257.3284       237.4272      177.73324      136.49501      154.46406
## 3      3635.1824      5290.4744     3207.80750     2327.48915     1673.94725
## 4      5687.0445      2281.1338     1699.76049     4962.88511     5666.42510
## 5     40234.2262     52443.1723    22396.75411    49371.37554    53146.12949
## 6     68414.8962     81869.8756    87495.30012   112485.07630    64501.25187
##   Config1_Run_15 Config1_Run_16 Config1_Run_17 Config1_Run_18 Config1_Run_19
## 1       62.41858       29.36893       78.64606       27.95884       78.46688
## 2      104.73899      155.47727       57.11502       62.19043      364.13050
## 3     1931.74500     5511.93922     3294.20502     4278.92123     4822.03359
## 4     2863.46707     3919.95480    12429.68123     2490.22781     4717.38323
## 5    21370.48209    64510.58861    56861.17332    40172.91770    46540.22167
## 6   101851.22557    93777.30151    35315.59989    84208.01575    98028.05567
##   Config1_Run_20 Config1_Run_21 Config1_Run_22 Config1_Run_23 Config1_Run_24
## 1       49.87066       45.27865       34.70898       15.13237       76.62001
## 2      203.14273      125.32063       77.36637      438.78095      245.14918
## 3     2487.84840     3750.03436     3549.92953     2761.00187     2788.78224
## 4     3128.03505     5168.25233     8037.55305     1105.46318     1381.99434
## 5    88649.28075    26707.53074    51005.02508    28488.27753    20381.53810
## 6    81950.69975   103651.90843    54487.32190    66102.00606   123914.45395
##   Config1_Run_25 Config1_Run_26 Config1_Run_27 Config1_Run_28 Config1_Run_29
## 1       34.58421       49.29599       82.25666       130.8007       92.33432
## 2      208.46325      183.22095       74.54916       296.6630      188.94881
## 3     2918.98590     4083.33548     1610.25518      6698.6161     1911.43922
## 4     8954.49100     2252.32415     2005.97696      2679.4531     1972.62840
## 5    60237.10308    71459.66519    46264.85545     55247.3023    92271.62285
## 6   125183.54058    82894.77287    93285.28355     95672.2961    50023.42696
##   Config1_Run_30 Config2_Run_1 Config2_Run_2 Config2_Run_3 Config2_Run_4
## 1       41.20557      1183.495      3490.684      811.6012      2131.126
```

```
## 2      163.37708     21261.993      3330.903     4341.1807     13506.019
## 3     3897.08855    127062.910    131800.791   116370.2634    123838.094
## 4     4408.05486    126986.866     77920.869   145666.9886    151825.596
## 5     5999.64194    514327.858    422554.174   506354.3098    494042.905
## 6   148495.00187    444108.062    501048.230   606226.1206    577685.976
##   Config2_Run_5 Config2_Run_6 Config2_Run_7 Config2_Run_8 Config2_Run_9
## 1      2752.987      2336.533      3067.733       820.936      2849.406
## 2      8395.412     22236.570     11752.098     11487.114     10049.689
## 3    127984.708     95839.597    107215.062     46138.064    115582.371
## 4    163552.229    155934.098    164742.593    201699.979    111797.761
## 5    500604.217    447823.276    322287.988    461915.436    483368.106
## 6    609224.265    580767.192    645325.418    607357.572    461930.064
##   Config2_Run_10 Config2_Run_11 Config2_Run_12 Config2_Run_13 Config2_Run_14
## 1       3522.495       1208.097       3209.194       1870.460        2548.54
## 2      11778.522      10540.659       7030.424       3108.393       24651.51
## 3      89292.988     108139.665     127256.922      62954.561      136264.84
## 4     128971.595     164129.694     129636.789     172121.892      173721.01
## 5     373890.016     399746.847     458901.083     314070.969      458577.06
## 6     475116.219     545844.212     522078.550     624695.292      612911.31
##   Config2_Run_15 Config2_Run_16 Config2_Run_17 Config2_Run_18 Config2_Run_19
## 1       1913.260       1980.372       876.1377       1835.688       1880.953
## 2       3571.587       6650.274     21420.2450       7718.099       7282.568
## 3     119035.031     123159.559    150787.5570     118952.300     134157.000
## 4     175119.049     125805.148    143661.6583     131341.606      55464.202
## 5     387391.862     486480.870    479604.8819     440579.106     458303.132
## 6     546128.130     595963.258    569048.4968     498414.170     524294.873
##   Config2_Run_20 Config2_Run_21 Config2_Run_22 Config2_Run_23 Config2_Run_24
## 1       1700.921        894.737       2497.774       1357.754       1957.358
## 2      10350.135      11853.988      16244.747      13553.078      12470.741
## 3     117222.140     133953.888      75477.514     108715.451     162835.442
## 4     145291.472     160860.900     155755.002      98590.062     185900.453
## 5     439839.472     424660.626     433400.837     487345.947     450037.125
## 6     590120.883     479778.593     421651.579     574119.161     576751.162
##   Config2_Run_25 Config2_Run_26 Config2_Run_27 Config2_Run_28 Config2_Run_29
## 1       2408.136       393.5118       5223.239       1887.327       2533.112
## 2      22094.605      9742.4304       8520.950       7239.358      14759.778
## 3      83618.932    127434.3519     119974.544     122221.836     122406.051
## 4     158494.112    187696.2427     125896.527     201564.810     149968.703
## 5     421272.394    511216.0834     445201.911     482893.489     424164.706
## 6     596468.656    488141.0697     550635.223     600184.673     467408.252
##   Config2_Run_30 Mean_Config1 Mean_Config2 Mean_General
## 1       2232.113     52.90526     2112.523     1082.714
## 2       7975.542    178.11652    11497.287     5837.702
## 3     117040.181   3297.75733   115091.087    59194.422
## 4     155904.849   3962.06624   147534.092    75748.079
## 5     339874.711  47339.15902   442357.713   244848.436
## 6     521533.784  92517.97720   547165.348   319841.663
```

```r
# Mostrar dimensões problemáticas
if (length(error_dimensions) > 0) {
  cat("Dimensões com erros:\n")
  print(error_dimensions)
} else {
  cat("Nenhuma dimensão apresentou erros.\n")
```

```
}
```

```
## Nenhuma dimensão apresentou erros.
```

Após a coleta dos dados, o código embaralha as linhas do DataFrame para eliminar possíveis padrões que possam enviesar a análise subsequente. Em seguida, ele ajusta os valores individuais das execuções para calcular os resíduos, subtraindo as médias geral e da configuração correspondente. Essa transformação é essencial para isolar as variações residuais e realizar análises estatísticas precisas, como testes de normalidade. Este procedimento permite comparar o desempenho entre configurações de forma mais robusta e alinhada com os pressupostos estatísticos necessários para análises posteriores.

```
set.seed(123) # Garantir reprodutibilidade
results_shuffled <- results[sample(nrow(results)), ] # Embaralha as linhas
head(results_shuffled)
```

```
##    Dimension Config1_Run_1 Config1_Run_2 Config1_Run_3 Config1_Run_4
## 31       143    8573929.58   8220313.091   7991406.341   9081154.318
## 15        75    2720129.85   2777252.793   2051422.556   2445930.442
## 19        91    3880156.41   2894318.622   3896072.644   4117135.534
## 14        73    2457777.28   2321904.703   2026104.927   1763047.346
## 3         15       3774.05       613.868      3055.061      3249.585
## 10        44     499693.80    431316.201    463229.905    470893.840
##    Config1_Run_5 Config1_Run_6 Config1_Run_7 Config1_Run_8 Config1_Run_9
## 31   8487243.986   8324288.399   8948190.682   7189873.134   8557163.231
## 15   1852511.572   2554250.652   2394176.440   2168471.953   2255811.564
## 19   4171138.018   3378664.677   3170322.472   4006108.323   3919897.420
## 14   1702377.515   2354682.413   2464035.186   1357391.328   2536407.974
## 3       3321.618      2088.499      4892.062      2927.772      2579.144
## 10    465828.934    529208.417    456086.308    739321.056    514073.030
##    Config1_Run_10 Config1_Run_11 Config1_Run_12 Config1_Run_13 Config1_Run_14
## 31    8410507.779    6578065.907    8085404.604    8188539.984    7621463.694
## 15    1614872.641    2256033.255    2321078.745    2490345.653    2655166.553
## 19    3648824.172    3505873.871    3648484.295    3291757.308    3970391.930
## 14    2218576.320    2714446.220    2442291.731    1490319.045    2534716.844
## 3        3635.182       5290.474       3207.807       2327.489       1673.947
## 10     605647.256     666890.190     418538.108     602950.525     461137.668
##    Config1_Run_15 Config1_Run_16 Config1_Run_17 Config1_Run_18 Config1_Run_19
## 31    8612462.395    8880012.758    8698372.503    9150353.250    7279060.269
## 15    2163567.702    2563930.667    2637997.431    1727920.191    2715372.660
## 19    3661824.154    3958874.618    3686251.329    3971651.437    3124642.376
## 14    2720617.610    2003747.574    2525543.948    1897675.284    1563809.001
## 3        1931.745       5511.939       3294.205       4278.921       4822.034
## 10     596171.066     629374.548     524597.459     518876.842     416528.339
##    Config1_Run_20 Config1_Run_21 Config1_Run_22 Config1_Run_23 Config1_Run_24
## 31    8212790.814    8885344.250     8439465.97    9339941.745    9134514.341
## 15    1967961.517    2555442.319     2451442.05    2053761.342    2663244.587
## 19    3552804.035    3799004.440     3715899.69    3068505.476    4084233.706
## 14    1908040.310    2341979.642     1901143.67    2284035.237    1402056.832
## 3        2487.848       3750.034        3549.93       2761.002       2788.782
## 10     623086.933     633768.002      578987.26     509478.202     143835.576
##    Config1_Run_25 Config1_Run_26 Config1_Run_27 Config1_Run_28 Config1_Run_29
## 31    7544232.457    9667571.327    9410024.730    8485665.261    7901007.162
## 15    2519486.243    2810528.820    1820971.422    1361926.038    2961339.556
## 19    3459025.955    3837322.277    3352209.713    4154717.077    3322723.397
## 14    2519277.192    2290942.612    1993805.893    1943628.620    2191647.801
```

```
## 3        2918.986       4083.335       1610.255       6698.616       1911.439
## 10     576356.937     614342.464     528093.684     485919.770     416648.772
##    Config1_Run_30 Config2_Run_1 Config2_Run_2 Config2_Run_3 Config2_Run_4
## 31    9964096.355    11748353.9    10885052.6    11016401.7     9905805.1
## 15    2325965.141     5172002.5     5400357.0     3930044.3     4986208.1
## 19    3699065.306     5283673.1     6085262.2     5927769.3     5604093.6
## 14    1649421.797     3987697.7     4813537.0     4432225.7     3619869.7
## 3        3897.089      127062.9      131800.8      116370.3      123838.1
## 10     541158.608     1475081.0     1295546.9     1433911.9     1376743.4
##    Config2_Run_5 Config2_Run_6 Config2_Run_7 Config2_Run_8 Config2_Run_9
## 31    10733131.6    11149172.7    11472010.2   11138497.26    10958476.6
## 15     4141207.0     5079929.1     4616929.4    4442088.82     5302561.5
## 19     5973827.5     5993360.8     6544966.4    5938936.65     6622760.2
## 14     4804870.7     4206385.5     5131038.2    4596635.64     3554111.4
## 3       127984.7       95839.6      107215.1      46138.06      115582.4
## 10     1426952.8     1309867.7     1381054.9    1363461.22     1438428.8
##    Config2_Run_10 Config2_Run_11 Config2_Run_12 Config2_Run_13 Config2_Run_14
## 31     9475793.18     11280679.5     10418459.5    10776599.88     10923372.9
## 15     5005209.51      4398489.8      4155319.1     4072093.37      5679848.0
## 19     5416513.64      6111107.9      5783120.4     5882577.29      6415164.2
## 14     4500277.76      4541537.5      5216213.4     4550969.91      3495248.0
## 3        89292.99       108139.7       127256.9       62954.56       136264.8
## 10     1294332.14      1367932.3      1351478.8     1394745.96      1428509.6
##    Config2_Run_15 Config2_Run_16 Config2_Run_17 Config2_Run_18 Config2_Run_19
## 31       11366878     11786603.1     10967762.8     11490590.2       11167707
## 15        4441709      4438376.2      4348783.5      5125096.3        4227394
## 19        6219599      6353662.4      5721440.5      5776688.8        6561674
## 14        5280120      4019449.2      3148282.6      4761021.3        4974954
## 3          119035       123159.6       150787.6       118952.3         134157
## 10        1464188      1143464.9      1142325.0      1302864.1        1235374
##    Config2_Run_20 Config2_Run_21 Config2_Run_22 Config2_Run_23 Config2_Run_24
## 31     10274863.9     11116825.5    10979594.91     10940049.5     11436952.6
## 15      4390314.7      3890466.3     4688352.97      4654313.3      5632193.9
## 19      6444535.3      6363776.8     4954277.48      5757218.6      6263159.4
## 14      4526242.5      4391673.9     4903057.93      2821522.4      4883912.0
## 3        117222.1       133953.9       75477.51       108715.5       162835.4
## 10      1391860.6      1049915.0     1458064.37      1349513.1      1482399.3
##    Config2_Run_25 Config2_Run_26 Config2_Run_27 Config2_Run_28 Config2_Run_29
## 31    11407359.75     10169335.0      9868148.3     10351582.3     10070165.0
## 15     4408031.89      5457488.6      4119899.1      5268776.9      5053369.1
## 19     6894203.75      5637079.5      6170410.1      6555905.8      6272291.8
## 14     4444738.43      5135249.6      4821001.5      3617928.1      4891333.3
## 3        83618.93       127434.4       119974.5       122221.8       122406.1
## 10     1257622.94      1461282.3      1222179.3      1539143.6      1410440.8
##    Config2_Run_30 Mean_Config1 Mean_Config2 Mean_General
## 31     11431984.5  8462082.010   10890273.7   9676177.83
## 15      5286867.7  2328610.412    4727124.0   3527867.23
## 19      5446636.8  3664930.023    6032523.1   4848726.55
## 14      4403377.8  2117381.728    4415816.1   3266598.91
## 3        117040.2     3297.757     115091.1     59194.42
## 10      1281316.1   522067.990    1351000.0    936534.01
```

```r
# Remover as três últimas colunas
results_individual <- results_shuffled[, -which(names(results_shuffled) %in% c("Mean_Config1", "Mean_Co
```

```r
# Calcular resíduos
calcular_residuos <- function(row) {
  mean_general <- row["Mean_General"]
  mean_config1 <- row["Mean_Config1"]
  mean_config2 <- row["Mean_Config2"]

  # Selecionar resultados individuais
  individual_results <- row[grep("Config[12]_Run_", names(row))]

  # Calcular resíduos para cada configuração
  residuals_config1 <- individual_results[grep("Config1_Run_", names(individual_results))] - mean_genera
  residuals_config2 <- individual_results[grep("Config2_Run_", names(individual_results))] - mean_genera

  c(residuals_config1, residuals_config2)
}

# Aplicar cálculo dos resíduos para cada linha
residuals_matrix <- t(apply(results, 1, calcular_residuos))

# Criar nomes de colunas descritivos
col_names_config1 <- names(results)[grep("Config1_Run_", names(results_shuffled))]
col_names_config2 <- names(results)[grep("Config2_Run_", names(results_shuffled))]
residual_col_names <- c(
  paste0("Residual_", col_names_config1),
  paste0("Residual_", col_names_config2)
)

# Converter para DataFrame com os novos nomes de colunas
residuals_df <- as.data.frame(residuals_matrix)
names(residuals_df) <- residual_col_names

# Verificar a estrutura do DataFrame de resíduos
head(residuals_df)
```

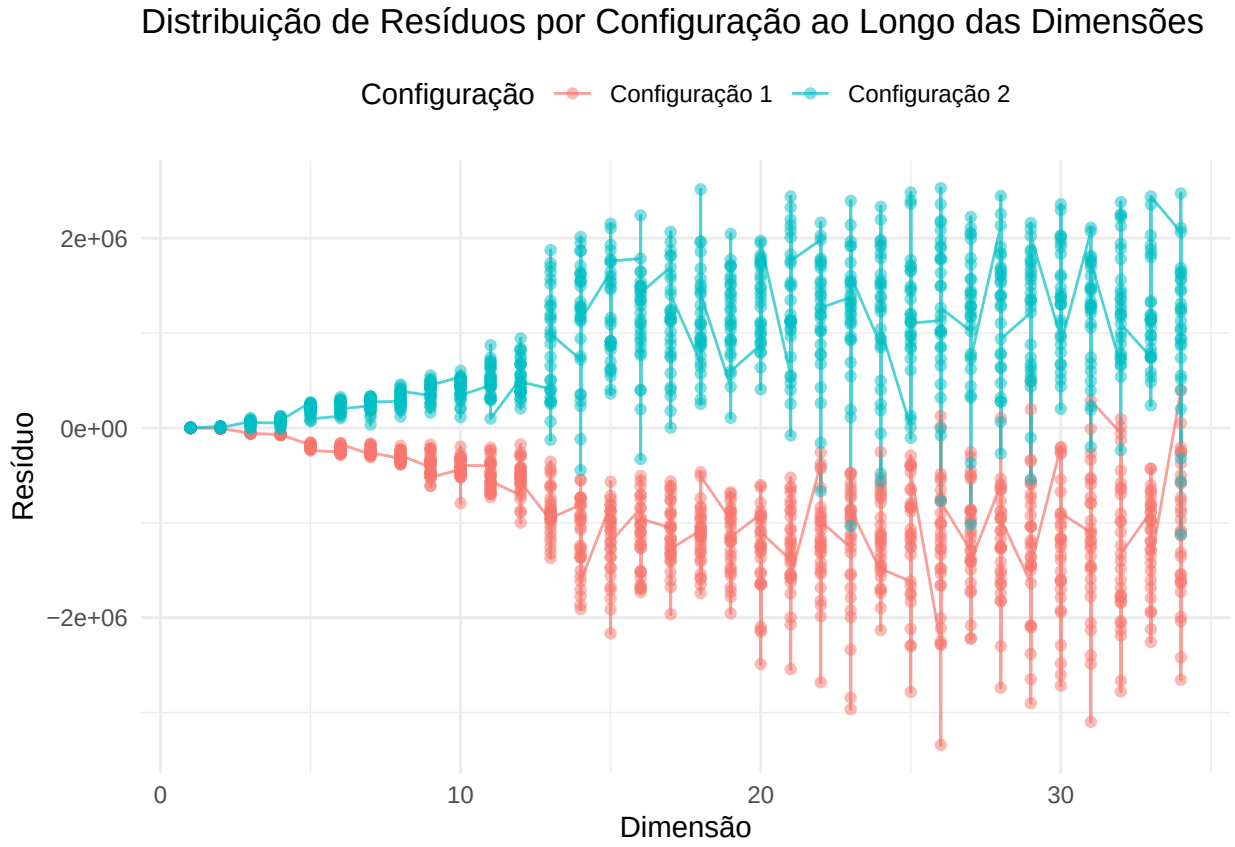
```
##   Residual_Config1_Run_1 Residual_Config1_Run_2 Residual_Config1_Run_3
## 1              -1069.570              -1051.797              -1039.025
## 2              -5713.886              -5674.611              -5636.387
## 3             -55420.372             -58580.554             -56139.361
## 4             -71718.035             -74108.899             -69401.840
## 5            -179503.438            -220471.173            -216901.710
## 6            -252302.329            -263244.969            -204348.836
##   Residual_Config1_Run_4 Residual_Config1_Run_5 Residual_Config1_Run_6
## 1              -1037.068              -1052.309               -1041.44
## 2              -5613.268              -5763.038               -5580.60
## 3             -55944.837             -55872.804              -57105.92
## 4             -71401.978             -73557.305              -72180.86
## 5            -203398.215            -205090.910             -198133.82
## 6            -279324.480            -164917.454             -156614.23
##   Residual_Config1_Run_7 Residual_Config1_Run_8 Residual_Config1_Run_9
## 1              -1043.751              -1065.801              -1027.246
## 2              -5734.230              -5748.994              -5642.888
## 3             -54302.360             -56266.650             -56615.279
## 4             -71947.298             -72787.553             -73579.143
```

```
## 5            -195533.100            -195732.969            -162454.707
## 6            -195134.079            -246998.319            -193758.261
##   Residual_Config1_Run_10 Residual_Config1_Run_11 Residual_Config1_Run_12
## 1                -1045.181                -895.325                -1058.699
## 2                -5580.373               -5600.275                -5659.968
## 3               -55559.240              -53903.948               -55986.615
## 4               -70061.035              -73466.945               -74048.319
## 5              -204614.210             -192405.264              -222451.682
## 6              -251426.767             -237971.787              -232346.363
##   Residual_Config1_Run_13 Residual_Config1_Run_14 Residual_Config1_Run_15
## 1                -1053.613                -1018.960                -1020.295
## 2                -5701.207                -5683.238                -5732.963
## 3               -56866.933               -57520.475               -57262.677
## 4               -70785.194               -70081.654               -72884.612
## 5              -195477.061              -191702.307              -223477.954
## 6              -207356.586              -255340.411              -217990.437
##   Residual_Config1_Run_16 Residual_Config1_Run_17 Residual_Config1_Run_18
## 1                -1053.345                -1004.068                -1054.755
## 2                -5682.224                -5780.587                -5775.511
## 3               -53682.483               -55900.217               -54915.501
## 4               -71828.124               -63318.398               -73257.851
## 5              -180337.848              -187987.263              -204675.518
## 6              -226064.361              -284526.063              -235633.647
##   Residual_Config1_Run_19 Residual_Config1_Run_20 Residual_Config1_Run_21
## 1                -1004.247                -1032.843                -1037.435
## 2                -5473.571                -5634.559                -5712.381
## 3               -54372.389               -56706.574               -55444.388
## 4               -71030.696               -72620.044               -70579.827
## 5              -198308.214              -156199.155              -218140.905
## 6              -221813.607              -237890.963              -216189.754
##   Residual_Config1_Run_22 Residual_Config1_Run_23 Residual_Config1_Run_24
## 1                -1048.005                -1067.582                -1006.094
## 2                -5760.335                -5398.921                -5592.553
## 3               -55644.493               -56433.420               -56405.640
## 4               -67710.526               -74642.616               -74366.085
## 5              -193843.411              -216360.159              -224466.898
## 6              -265354.341              -253739.657              -195927.209
##   Residual_Config1_Run_25 Residual_Config1_Run_26 Residual_Config1_Run_27
## 1                -1048.130                -1033.418                -1000.457
## 2                -5629.238                -5654.481                -5763.153
## 3               -56275.436               -55111.087               -57584.167
## 4               -66793.588               -73495.755               -73742.102
## 5              -184611.333              -173388.771              -198583.581
## 6              -194658.122              -236946.890              -226556.379
##   Residual_Config1_Run_28 Residual_Config1_Run_29 Residual_Config1_Run_30
## 1                -951.9133                -990.3796                -1041.508
## 2                -5541.0387               -5648.7529                -5674.325
## 3               -52495.8061              -57282.9829               -55297.334
## 4               -73068.6260              -73775.4506               -71340.024
## 5              -189601.1338             -152576.8133              -238848.794
## 6              -224169.3666             -269818.2358              -171346.661
##   Residual_Config2_Run_1 Residual_Config2_Run_2 Residual_Config2_Run_3
## 1                100.7813                2407.970               -271.1127
## 2               15424.2908               -2506.799              -1496.5210
```

```
## 3             67868.4881              72606.369             57175.8412
## 4             51238.7867               2172.790             69918.9095
## 5            269479.4217             177705.738            261505.8737
## 6            124266.3993             181206.567            286384.4579
##   Residual_Config2_Run_4 Residual_Config2_Run_5 Residual_Config2_Run_6
## 1               1048.412               1670.273               1253.819
## 2               7668.317               2557.711              16398.868
## 3              64643.672              68790.285              36645.174
## 4              76077.517              87804.150              80186.019
## 5             249194.469             255755.781             202974.840
## 6             257844.313             289382.602             260925.529
##   Residual_Config2_Run_7 Residual_Config2_Run_8 Residual_Config2_Run_9
## 1               1985.019              -261.7779               1766.692
## 2               5914.396              5649.4124               4211.987
## 3              48020.640            -13056.3578              56387.949
## 4              88994.514             125951.8995              36049.682
## 5              77439.552             217066.9996             238519.670
## 6             325483.755             287515.9097             142088.401
##   Residual_Config2_Run_10 Residual_Config2_Run_11 Residual_Config2_Run_12
## 1                2439.781                125.3834                2126.480
## 2                5940.821               4702.9572                1192.722
## 3               30098.566              48945.2426               68062.500
## 4               53223.516              88381.6154               53888.710
## 5              129041.579             154898.4110              214052.646
## 6              155274.556             226002.5492              202236.887
##   Residual_Config2_Run_13 Residual_Config2_Run_14 Residual_Config2_Run_15
## 1                787.7456                1465.826                 830.546
## 2               -2729.3086               18813.803               -2266.114
## 3                3760.1389               77070.413               59840.609
## 4               96373.8129               97972.931               99370.970
## 5               69222.5329              213728.626              142543.426
## 6              304853.6291              293069.649              226286.468
##   Residual_Config2_Run_16 Residual_Config2_Run_17 Residual_Config2_Run_18
## 1                897.6577                -206.5762                 752.974
## 2                812.5719               15582.5433                1880.398
## 3               63965.1366               91593.1348               59757.878
## 4               50057.0691               67913.5792               55593.526
## 5              241632.4337              234756.4458              195730.670
## 6              276121.5956              249206.8341              178572.507
##   Residual_Config2_Run_19 Residual_Config2_Run_20 Residual_Config2_Run_21
## 1                798.2387                618.2074                -187.977
## 2               1444.8660               4512.4328                6016.287
## 3               74962.5778              58027.7174               74759.466
## 4              -20283.8771              69543.3930               85112.821
## 5              213454.6959             194991.0354              179812.190
## 6              204453.2103             270279.2207              159936.930
##   Residual_Config2_Run_22 Residual_Config2_Run_23 Residual_Config2_Run_24
## 1                1415.06                275.0396                874.6439
## 2               10407.05                7715.3766               6633.0396
## 3               16283.09                49521.0284             103641.0194
## 4               80006.92                22841.9829             110152.3737
## 5              188552.40                242497.5108            205188.6892
## 6              101809.92                254277.4982            256909.4993
##   Residual_Config2_Run_25 Residual_Config2_Run_26 Residual_Config2_Run_27
```

```
## 1                1325.422              -689.2022               4140.525
## 2               16256.903              3904.7287               2683.248
## 3               24424.510             68239.9298              60780.122
## 4               82746.033            111948.1637              50148.448
## 5              176423.958            266367.6473             200353.475
## 6              276626.994            168299.4069             230793.560
##   Residual_Config2_Run_28 Residual_Config2_Run_29 Residual_Config2_Run_30
## 1                804.6126                1450.398                1149.399
## 2               1401.6559                8922.076                2137.840
## 3              63027.4142                63211.629               57845.759
## 4             125816.7308                74220.624               80156.770
## 5             238045.0530               179316.270               95026.275
## 6             280343.0103               147566.589              201692.121
```

```r
# Converter residuals_df para formato longo
residuals_long <- residuals_df %>%
  mutate(Dimension = 1:nrow(residuals_df)) %>% # Adicionar coluna de dimensão
  pivot_longer(
    cols = starts_with("Residual_"),
    names_to = "Configuration",
    values_to = "Residual"
  ) %>%
  mutate(
    Configuration_Type = ifelse(
      grepl("Config1", Configuration),
      "Configuração 1",
      "Configuração 2"
    )
  )

# Criar o gráfico
ggplot(residuals_long, aes(x = Dimension, y = Residual, color = Configuration_Type)) +
  geom_line(alpha = 0.7) + # Linhas para visualizar tendências
  geom_point(alpha = 0.5) + # Pontos para cada resíduo
  labs(
    title = "Distribuição de Resíduos por Configuração ao Longo das Dimensões",
    x = "Dimensão",
    y = "Resíduo",
    color = "Configuração"
  ) +
  theme_minimal() +
  theme(legend.position = "top")
```



Teste de premissa

Teste de Normalidade

```
# Carregar todos os resíduos
residuals_matrix <- as.matrix(residuals_df)

# 1. Teste de normalidade para todos os resíduos combinados
all_residuals <- as.vector(residuals_matrix)
shapiro_all <- shapiro.test(all_residuals)
cat("Teste de Normalidade - Todos os Resíduos:\n")
```

```
## Teste de Normalidade - Todos os Resíduos:
```

```
print(shapiro_all)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  all_residuals
## W = 0.99124, p-value = 9.608e-10
```

```
# 2. Teste de normalidade para resíduos da Configuração 1
config1_columns <- grep("Residual_Config1", colnames(residuals_df), value = TRUE)
residuals_config1 <- as.vector(as.matrix(residuals_df[, config1_columns]))
shapiro_config1 <- shapiro.test(residuals_config1)
cat("\nTeste de Normalidade - Resíduos Configuração 1:\n")
```

```
##
## Teste de Normalidade - Resíduos Configuração 1:
```

```
print(shapiro_config1)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals_config1
## W = 0.95476, p-value < 2.2e-16
```

```
# 3. Teste de normalidade para resíduos da Configuração 2
config2_columns <- grep("Residual_Config2", colnames(residuals_df), value = TRUE)
residuals_config2 <- as.vector(as.matrix(residuals_df[, config2_columns]))
shapiro_config2 <- shapiro.test(residuals_config2)
cat("\nTeste de Normalidade - Resíduos Configuração 2:\n")
```

```
##
## Teste de Normalidade - Resíduos Configuração 2:
```

```
print(shapiro_config2)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals_config2
## W = 0.96144, p-value = 9.05e-16
```

Os resultados dos testes de normalidade indicam que, tanto para os resíduos combinados quanto para os resíduos separados por configuração, a hipótese nula do teste Shapiro-Wilk, que assume que os dados seguem uma distribuição normal, foi rejeitada (p-valor < 0.05). O valor da estatística W, próximo de 0.9 em todos os casos, reflete um desvio considerável em relação à normalidade. Esses resultados sugerem que os resíduos, tanto no conjunto total quanto em cada configuração (Configuração 1 e Configuração 2), não apresentam distribuição normal. Isso pode impactar a escolha de métodos estatísticos subsequentes, especialmente se estes assumirem normalidade nos dados, exigindo alternativas não paramétricas ou transformações nos resíduos para adequação.

Teste de Independência

Para aplicar o teste de independência, o código mencionado calcula uma matriz de correlação, que verifica o grau de relação linear entre os resíduos de diferentes colunas. A interpretação da matriz ajudará a entender se há independência (valores de correlação próximos de 0) ou dependência (valores próximos de -1 ou 1).

```
# Carregar os resíduos normalizados (se necessário)
dados_normalizados <- residuals_matrix

# Calcular a matriz de correlação para os resíduos normalizados
matriz_correlacao <- cor(dados_normalizados, method = "pearson", use = "complete.obs")

# Análise resumida dos valores de correlação
cat("\nResumo da Correlação:\n")
```

```
##
## Resumo da Correlação:
```

```
summary(as.vector(matriz_correlacao))
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
```

```
## -0.8390423 -0.5237967  0.0328368  0.0005534  0.5095243  1.0000000
```

Os resultados da matriz de correlação indicam que os resíduos apresentam alta correlação linear entre si, com valores variando de 0,9583 a 1,0000. A mediana (0,9836) e a média (0,9830) reforçam a forte dependência linear entre as execuções analisadas. Esses resultados sugerem que os resíduos não são independentes, indicando possível interação entre os fatores analisados ou padrões sistemáticos nos dados que influenciam as diferentes execuções. Essa dependência deve ser considerada em análises subsequentes, especialmente se forem utilizados métodos que assumem independência dos resíduos, pois ela pode impactar a validade das conclusões estatísticas. Ajustes no modelo ou métodos específicos para tratar a dependência podem ser necessários.

Teste de Homogeneidade de Variância:

```
dados_longo <- pivot_longer(
  residuals_df,
  cols = starts_with("Residual_Config"),
  names_to = c("Configuração", "Execução"),
  names_pattern = "Residual_(Config\\d+)_Run_(\\d+)",
  values_to = "Resíduo"
)


leveneTest(Resíduo ~ Configuração, data = dados_longo)
```

```
## Warning in leveneTest.default(y = y, group = group, ...): group coerced to
## factor.
```

```
## Levene's Test for Homogeneity of Variance (center = median)
##         Df F value    Pr(>F)
## group    1  11.437 0.0007334 ***
##       2038
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

O Teste de Levene foi aplicado para verificar a homogeneidade de variâncias entre os grupos analisados, com o uso da mediana como medida de centralidade. O teste resultou em um valor de p-valor significativamente menor do que o nível de significância adotado de = 0,05. Dessa forma, rejeita-se a hipótese nula do teste (H), que afirma que as variâncias dos grupos são homogêneas. Isso indica que há diferenças significativas nas variâncias entre os grupos analisados.

Conclusão do Teste de premissa

Os testes estatísticos realizados indicaram que os resíduos não seguem uma distribuição normal (Shapiro-Wilk com p-valor $< 0,05$), apresentam alta dependência linear (média e mediana das correlações próximas a 0,98), e não possuem homogeneidade de variâncias entre as configurações (Teste de Levene com p-valor $<$ 0,001). Esses resultados invalidam o uso do ANOVA tradicional, que assume normalidade, independência e homogeneidade de variâncias. Como alternativa, recomenda-se o uso de métodos não paramétricos, como o Teste de Wilcoxon para comparações pareadas entre as configurações ou o Teste de Kruskal-Wallis para múltiplos grupos.

Teste não paramétricos

Para determinar qual das duas configurações (Config1 e Config2) apresenta melhor desempenho em todos os cenários, optamos pelo Teste de Permutação. Esse método foi escolhido devido às características dos dados, que não atendem às premissas de normalidade e homogeneidade de variâncias, conforme verificado nos testes preliminares (Shapiro-Wilk e Levene). O Teste de Permutação é robusto, não exige suposições

rígidas sobre a distribuição dos dados e permite a comparação direta das médias entre as configurações. Ele utiliza reamostragens para construir uma distribuição empírica da estatística de teste, fornecendo uma análise confiável para identificar se as diferenças observadas entre as configurações são estatisticamente significativas.

Formulação do Teste de Hipótese

A comparação entre as configurações será feita utilizando a diferença média entre os resíduos como estatística de interesse. As hipóteses são formuladas da seguinte maneira:

- Hipótese Nula (H0) : Não há diferença significativa entre as médias das duas configurações.

$$H_0 : \mu_{\text{Config1}} = \mu_{\text{Config2}}$$

- Hipótese Alternativa (H1) : Há uma diferença significativa entre as médias das duas configurações.

$$H_1 : \mu_{\text{Config1}} \neq \mu_{\text{Config2}}$$

A análise será realizada utilizando nível de significância $\alpha = 0.05$, garantindo que a probabilidade de um erro tipo I (rejeitar H0 quando verdadeira) não ultrapasse 5%.

Teste de Hipótese

```
# Inicializar lista para armazenar resultados de cada dimensão
dimensoes <- unique(results_shuffled$Dimension) # Obter valores únicos de dimensão
n_perm <- 100000  # Número de permutações
resultados_dimensao <- data.frame(Dimensao = integer(), p_value = numeric())

# Loop para realizar o Teste de Permutação em cada dimensão
for (dim in dimensoes) {
  dim <- as.numeric(dim)  # Garantir que dim é numérico

  # Filtrar linhas correspondentes à dimensão atual
  linhas_dim <- results_shuffled[results_shuffled$Dimension == dim, ]

  # Verificar se a coluna "Mean_General" existe e extrair sua média
  mean_general <- linhas_dim$Mean_General[1]

  # Extrair resíduos para Config1 e Config2 e subtrair a média global
  config1_cols <- grep("Config1_Run_", colnames(linhas_dim), value = TRUE)
  config2_cols <- grep("Config2_Run_", colnames(linhas_dim), value = TRUE)

  config1 <- as.vector(as.matrix(linhas_dim[, config1_cols])) - mean_general
  config2 <- as.vector(as.matrix(linhas_dim[, config2_cols])) - mean_general

  t_hat <- mean(config1) - mean(config2)
  combined_data <- append(config1, config2)
  n1 <- length(config1)
  n2 <- length(config2)
  t_perm <- numeric(n_perm)

  # Realizar Teste de Permutação
  set.seed(123 + dim)  # Reprodutibilidade
  for (i in 1:n_perm) {
      S_perm <- sample(combined_data)  # Embaralhar os dados
      X_perm <- S_perm[1:n1]  # Novos valores para config1
      Y_perm <- S_perm[(n1 + 1):(n1 + n2)]  # Novos valores para config2
```

```
    t_perm[i] <- mean(X_perm) - mean(Y_perm)
  }

  # Calcular p-valor (H1: mu_config1 > mu_config2)
  p_value <- (1 + sum(abs(t_perm) >= abs(t_hat))) / (1 + n_perm)
  resultados_dimensao <- rbind(resultados_dimensao, data.frame(Dimensao = dim, p_value = p_value))

}
resultados_dimensao$p_value <- formatC(as.numeric(resultados_dimensao$p_value), format = "e", digits =

# Imprimir resultados
print(resultados_dimensao)
```

```
##    Dimensao          p_value
## 1       143 9.9999000010e-06
## 2        75 9.9999000010e-06
## 3        91 9.9999000010e-06
## 4        73 9.9999000010e-06
## 5        15 9.9999000010e-06
## 6        44 9.9999000010e-06
## 7        82 9.9999000010e-06
## 8       100 9.9999000010e-06
## 9        51 9.9999000010e-06
## 10       24 9.9999000010e-06
## 11       92 9.9999000010e-06
## 12      150 9.9999000010e-06
## 13      119 9.9999000010e-06
## 14      110 9.9999000010e-06
## 15      147 9.9999000010e-06
## 16       42 9.9999000010e-06
## 17      138 9.9999000010e-06
## 18       33 9.9999000010e-06
## 19      136 9.9999000010e-06
## 20       28 9.9999000010e-06
## 21       77 9.9999000010e-06
## 22      107 9.9999000010e-06
## 23      125 9.9999000010e-06
## 24       16 9.9999000010e-06
## 25        8 9.9999000010e-06
## 26      148 9.9999000010e-06
## 27       93 9.9999000010e-06
## 28      118 9.9999000010e-06
## 29       10 9.9999000010e-06
## 30       79 9.9999000010e-06
## 31       54 9.9999000010e-06
## 32      104 9.9999000010e-06
## 33       61 9.9999000010e-06
## 34       27 9.9999000010e-06
```

Os resultados obtidos a partir do Teste de Permutação evidenciam diferenças estatisticamente significativas entre as configurações Config1 e Config2 em todas as dimensões analisadas. Com p-valores extremamente baixos, próximos de 9.9999×10^{-6}, para todas as dimensões testadas é possível rejeitar a Hipótese Nula (H_0) com alto grau de confiança ($\alpha = 0.05$). Esses valores indicam que as diferenças observadas entre as médias das duas configurações não são atribuíveis ao acaso e, portanto, apontam um desempenho significativamente

distinto entre Config1 e Config2. Isso reforça a robustez do método aplicado e a confiabilidade dos resultados obtidos.

```
# Combinar resíduos de todas as dimensões para Config1 e Config2
config1_global <- as.vector(as.matrix(residuals_df[, grep("Residual_Config1", colnames(residuals_df))]))
config2_global <- as.vector(as.matrix(residuals_df[, grep("Residual_Config2", colnames(residuals_df))]))

# Calcular a diferença média observada global
obs_diff_global <- mean(config1_global) - mean(config2_global)

# Combinar todos os dados em um único vetor para permutação
combined_data_global <- c(config1_global, config2_global)
n_config1_global <- length(config1_global)
n_perm <- 100000  # Número de permutações
perm_diffs_global <- numeric(n_perm)  # Inicializar vetor para armazenar permutações

# Realizar Teste de Permutação Global
set.seed(999)  # Garantir reprodutibilidade
for (i in 1:n_perm) {
  shuffled_data <- sample(combined_data_global)  # Embaralhar os dados
  perm_config1 <- shuffled_data[1:n_config1_global]  # Nova amostra Config1
  perm_config2 <- shuffled_data[(n_config1_global + 1):length(shuffled_data)]  # Nova amostra Config2
  perm_diffs_global[i] <- mean(perm_config1) - mean(perm_config2)  # Estatística de permutação
}


p_value_global <- (1 + sum(abs(perm_diffs_global) >= abs(obs_diff_global))) / (1 + n_perm)
# Exibir resultados globais
cat("### Teste de Permutação Global ###\n")
```

```
## ### Teste de Permutação Global ###
```

```
cat("Diferença média observada (global):", obs_diff_global, "\n")
```

```
## Diferença média observada (global): -1706261
```

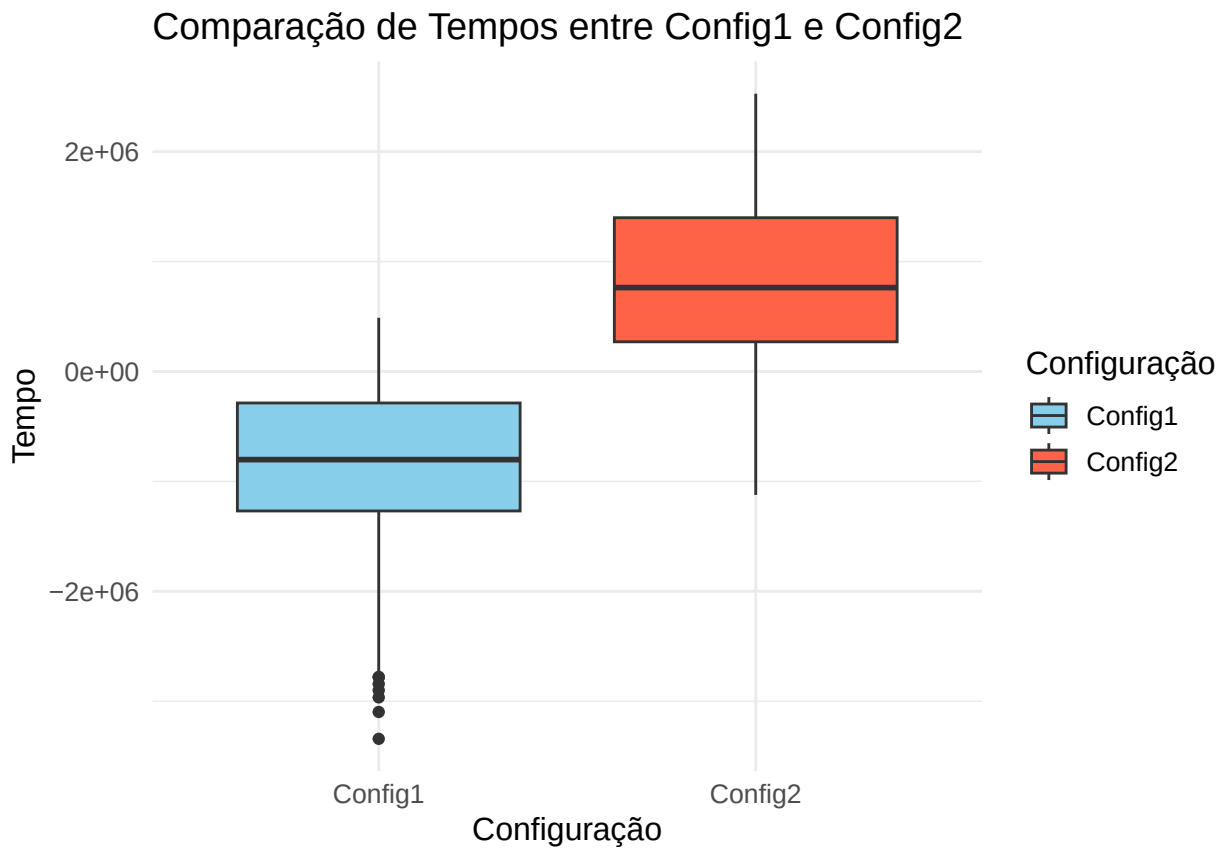
```
cat("P-valor (global):", p_value_global, "\n")
```

```
## P-valor (global): 9.9999e-06
```

Com base nos resultados do Teste de Permutação Global , a diferença média observada de -1.706.261 indica que os tempos da Config2 foram significativamente maiores do que os da Config1 . Como o desempenho é avaliado pelo tempo e quanto maior o tempo, pior a solução , conclui-se que a Config2 apresentou um desempenho muito inferior em relação à Config1 . O p-valor extremamente baixo (9.9999×10) reforça que essa diferença não é fruto do acaso, sendo estatisticamente significativa. Assim, a Config1 se destaca como a melhor solução, apresentando tempos menores e, consequentemente, maior eficiência em comparação à Config2 .

```
# Dados de exemplo: ajuste para usar seus próprios dados
config1_times <- as.vector(as.matrix(residuals_df[, grep("Residual_Config1", colnames(residuals_df))]))
config2_times <- as.vector(as.matrix(residuals_df[, grep("Residual_Config2", colnames(residuals_df))]))

# Criar um dataframe consolidado
data_visual <- data.frame(
  Tempo = c(config1_times, config2_times),
  Configuração = rep(c("Config1", "Config2"), c(length(config1_times), length(config2_times)))
)
```

```r
# Gráfico de Boxplot
ggplot(data_visual, aes(x = Configuração, y = Tempo, fill = Configuração)) +
  geom_boxplot() +
  theme_minimal() +
  labs(title = "Comparação de Tempos entre Config1 e Config2",
       x = "Configuração",
       y = "Tempo",
       fill = "Configuração") +
  scale_fill_manual(values = c("Config1" = "skyblue", "Config2" = "tomato")) +
  theme(text = element_text(size = 12))
```



O gráfico de caixas apresenta uma comparação visual dos tempos entre as configurações Config1 e Config2. Observa-se que a Config1 possui uma mediana significativamente menor, além de uma distribuição mais concentrada abaixo do eixo zero, indicando tempos menores e mais consistentes. Em contraste, a Config2 apresenta valores de tempo consideravelmente mais elevados e maior dispersão, evidenciando um desempenho inferior. A presença de outliers na Config1 sugere alguns valores atípicos, porém eles não comprometem a tendência geral. Assim, a análise confirma que a Config1 supera a Config2, apresentando tempos menores e, consequentemente, melhor desempenho.

Teste de Wilcoxon

```r
# Teste de Wilcoxon
wilcox_result <- wilcox.test(config1_global, config2_global, paired = TRUE, alternative = "less")

# Exibir resultado
print(wilcox_result)
```

```
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  config1_global and config2_global
## V = 264, p-value < 2.2e-16
## alternative hypothesis: true location shift is less than 0
```

O resultado do Teste de Wilcoxon com correção de continuidade reforça as conclusões obtidas anteriormente. Com um p-valor extremamente baixo ($p < 2.2 \times 10^{1}$), rejeita-se a Hipótese Nula (H_0) de que não há diferença significativa entre as configurações. A hipótese alternativa, que indica que os tempos da Config1 são significativamente menores do que os da Config2 , é corroborada. O valor da estatística $V = 264$ e o resultado do teste confirmam que a Config1 apresenta desempenho superior , com tempos mais baixos, enquanto a Config2 demonstra desempenho inferior, validando as análises anteriores de maneira robusta.