Estudantes:

Arthur Fernandes Minduca de Sousa – fernandes.arthur@gmail.com

Carlos Eduardo Buarque Cruz Pimentel – cebcp@cin.ufpe.br

Carlos Henrique Maciel Sobral Timóteo – chmst@cin.ufpe.br

Karina Rodrigues Pereira – krp@cin.ufpe.br

Vínculo:

Mestrado Acadêmico

Disciplina:

Avaliação de Desempenho de Sistemas

Professor:

Paulo Maciel

Atividade:

Resolução da 4ª Lista de Exercícios

1) A planilha de dados Planilha1 apresenta uma amostra aleatória simples dos tempos de percurso Recife-Maceió dos ônibus da companhia A.

- ***Gere o histograma dos dados.***

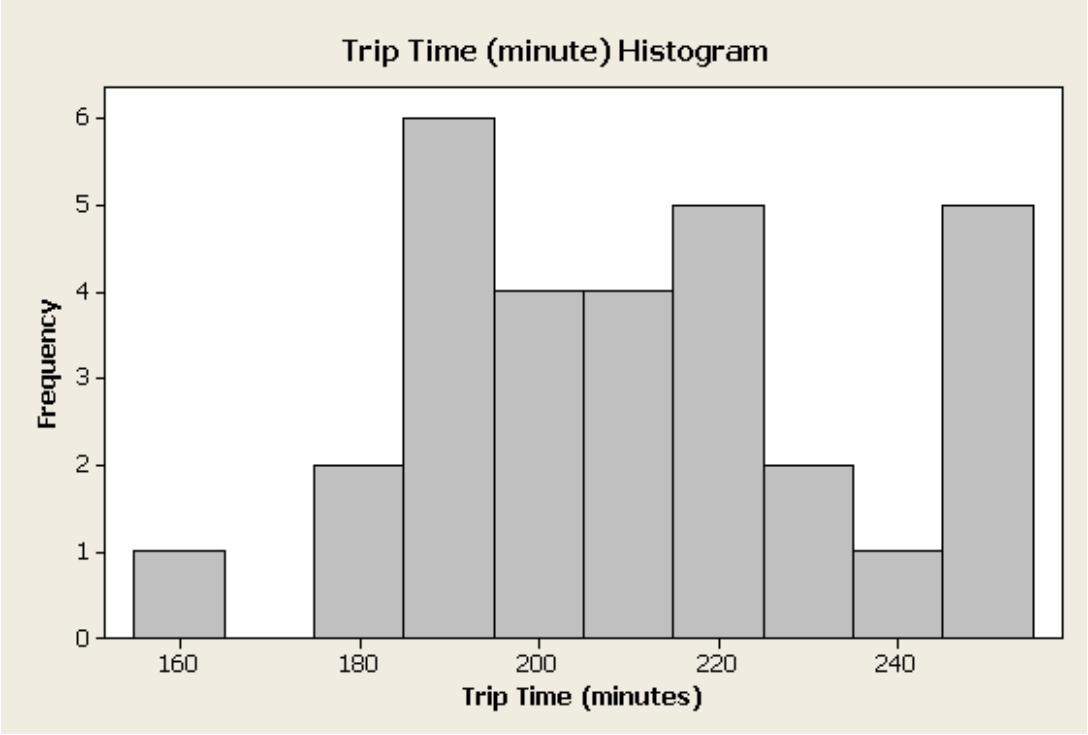


Figura 1 – Histograma dos Dados da Planilha 1

- ***Gere o resumo estatístico dos dados, e forneça a média, mediana, moda (se houver), média das juntas(média do 1 e 3 quartil), média do intervalo (midrange = Max + min / 2), desvio padrão, variância, quartís, intervalo inter-quartil, mínimo, máximo, assimetria e curtose.***

As estatísticas descritivas geradas são apresentadas a seguir:

Descriptive Statistics: Trip Time (minutes)

Trip Time	Média	DesvPadrao	Variância	Mínimo	1° Quartil	Mediana
	212,49	24,58	604,30	156,25	193,89	212,31
	3° Quartil	Máximo	IQR	Moda	FreqModa	Assim	Curtose
	229,99	254,96	36,10	219,17	2	0,06	-0,33
	MedJuntas	MedIntervalo					
	211,94	205,60					

Média das Juntas = Q1+Q3/2 = 211,94
Média do Intervalo = Minimo+Maxino/2 = 205,605

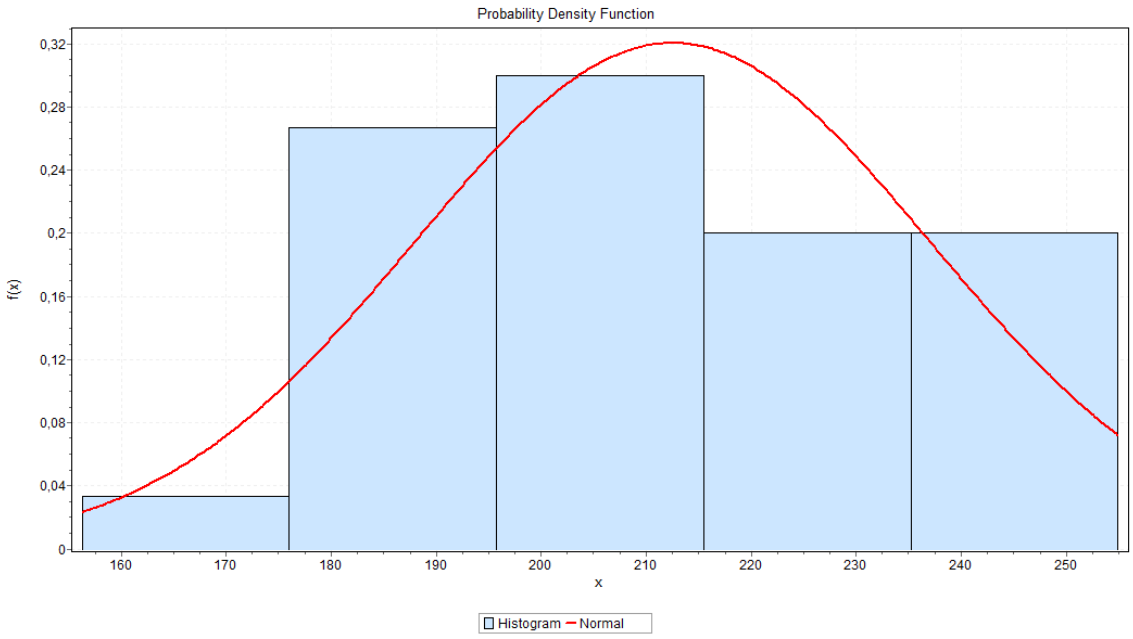
• Faça um resumo textual que interprete os dados analisados e informe se há evidências que indiquem se os dados provêm de uma população com distribuição normal.

O histograma não apresenta uma forma gráfica que seja tolerável assumir como uma distribuição normal. Entretanto, analisando os dados estatísticos, pode-se concluir que a distribuição dos dados, embora não seja uma normal, aproxima-se muito de tal distribuição. Por definição, a distribuição normal possui méida, mediana e moda iguais. Olhando para esses valores obtidos a partir dos dados da planilha, usando-se o MiniTab, é possível concluir que média e mediana tem valores muito próximos com 212,49 e 212,31, respectivamente. Apesar da moda não ser tão próxima desses valores, ela está dentro do primeiro desvio padrão que é de 24,58. Além disso, observando a média das juntas, tem-se mais evidência de que a distribuição aproxima-se da normal, uma vez que essa é também uma medida de tendência central e está muito próxima dos valores da média e da mediana.

No que diz respeito à forma do gráfico (histograma) de uma distribuição de dados, temos duas medidas para analisá-la: a assimetria e a curtose. No caso de distribuições normais, essas medidas têm valor igual a 0. Para a distribuição dos dados da questão, pode ser visto que esses valores são diferentes de 0, mas próximos desse valor (principalmente a assimetria). Essa é mais uma evidência de que, apesar de não ser uma normal, aproximar para tal a distribuição desses dados é algo bastante plausível.

Para dar mais embasamento a essa conclusão, foi realizado um teste de aderência, o Kolgomorov, usando o EasyFit, que mostrou que a normal fica na 12ª colocação (dentre 60 distribuições) em termos de proximidade com a real distribuição dos dados, com uma confiança de 90,566% como mostra a Figura 2.

Portanto, podemos concluir que, estatisticamente, a amostra, provavelmente, provém de uma população com distribuição normal, mesmo que não seja graficamente aparente, devido ao tamanho reduzido da amostra.



Normal	0,09434	12

Figura 2 – Teste de Kolgomorov para a amostra de dados da Planilha 1.

• ***Utilizando os dados da Planilha1, faça pelo menos 60 amostras aleatórias de tamanho 20, calcule a média destas amostras, gere o histograma das médias e o resumo estatístico.***

Médias

1. 215,004
2. 215,567
3. 217,977
4. 217,910
5. 208,552
6. 209,650
7. 211,916
8. 216,321
9. 211,912
10. 217,572
11. 209,810
12. 211,640
13. 215,346
14. 213,294
15. 209,190
16. 208,640
17. 209,396
18. 208,080
19. 214,997
20. 211,324
21. 217,353
22. 209,352
23. 215,744
24. 218,761
25. 212,266
26. 212,650
27. 210,742
28. 214,116
29. 217,958
30. 211,597
31. 211,900
32. 211,924
33. 215,269
34. 209,826
35. 222,150
36. 215,020
37. 210,184
38. 213,134
39. 209,871
40. 209,636
41. 208,650
42. 208,624
43. 216,406
44. 210,140
45. 210,513
46. 213,406
47. 207,605
48. 214,297
49. 211,792
50. 212,236
51. 209,926
52. 211,060
53. 210,549
54. 205,838
55. 209,355
56. 209,727
57. 210,643
58. 216,163
59. 216,205
60. 209,161

Histograma

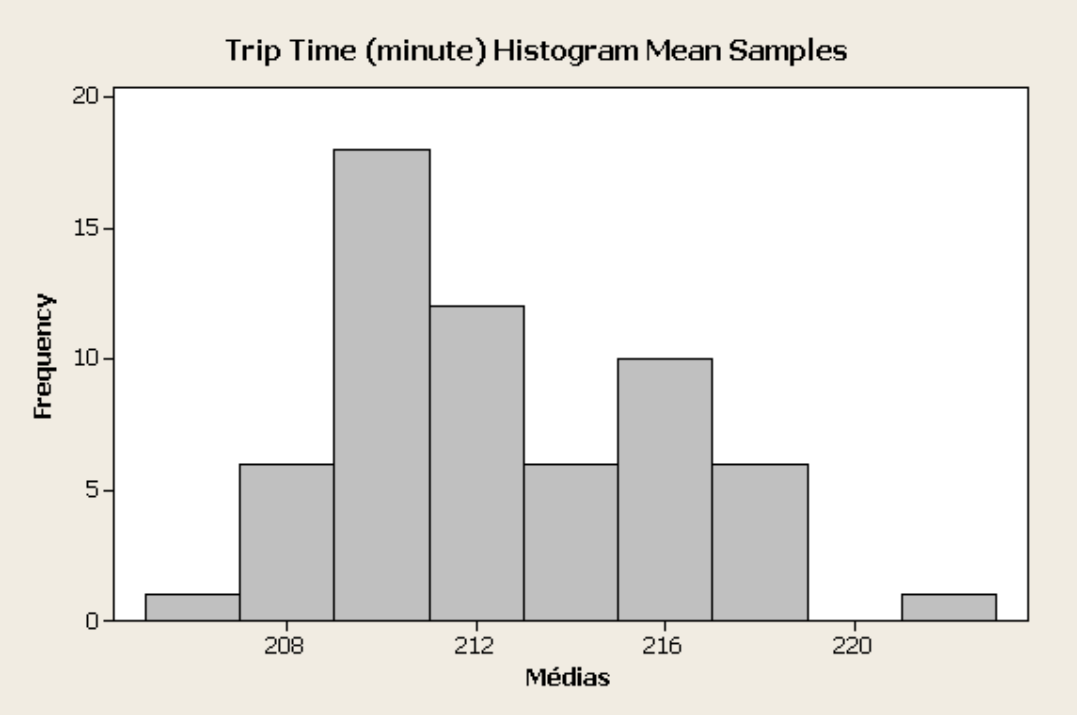


Figura 3 – Histograma das médias de 60 amostras dos dados da Planilha 1.

Descriptive Statistics: Trip Time (minutes)

Trip Time	Média	DesvPadrao	Variância	Mínimo	1° Quartil	Mediana
	212,43	3,38	11,42	205,84	209,75	211,85
	3° Quartil	Máximo	IQR	Moda	FreqModa	Assim	Curtose
	215,21	222,15	5,46	-	0	0,59	-0,20
	MedJuntas	MedIntervalo					
	212,48	213,99					

Média das Juntas = Q1+Q3/2 = 212,48
Média do Intervalo = Minimo+Maxino/2 = 213,995

• Faça um resumo textual sobre a distribuição das médias (obtidos no passo d), comentando os resultados e descreva seu entendimento sobre o Teorema Central do Limite, dado o estudo realizado neste exercício.

Olhando para o histograma das médias da amostra, pode ser dito que ele se aproxima mais de um gráfico da distribuição normal do que o histograma inicial. Por outro lado, olhando para as estatísticas desses dados, pode-se dizer que essa distribuição das médias das amostras não se aproxima tanto de uma normal quanto a distribuição dos dados inicias.

Ainda assim, há semelhança da distribuição das médias das amostras com a distribuição normal. A média, a mediana, a média das juntas e a média dos intervalos da amostra têm valores próximos. A assimetria da amostra é maior que zero, porém muito próxima de zero, o que indica a presença de uma cauda sutilmente alongada à direita; enquanto a curtose é menor que zero, porém relativamente próxima de zero, o que indica um leve achatamento.

Para corroborar ainda mais com a afirmação de há semelhança entre a distribuição das médias e uma distribuição normal, mas que ela é menor do que a observada para a situação anterior, foi utilizado o EasyFit na aplicação do Teste de de Kolgomorov. O teste mostrou que a distribuição normal é a 35ª (de 60) em semelhança, com confiança de 87,372%, conforme mostra a Figura 4.

Entretanto, é preciso levar em consideração que, à medida que esse número de amostras crescer, mais e mais esses dados estatísticos (e, consequentemente, o histograma) se assemelharão com os de uma distribuição normal.

Isso é justamente o que explica o Teorema do Limite Central. Além disso, pelo Teorema, a média das médias da amostra aproxima-se da média da população, bem como a variância vai ficando próxima do resultado da variância da população dividido pelo número de amostras, à medida que o número de amostra tende ao infinito.

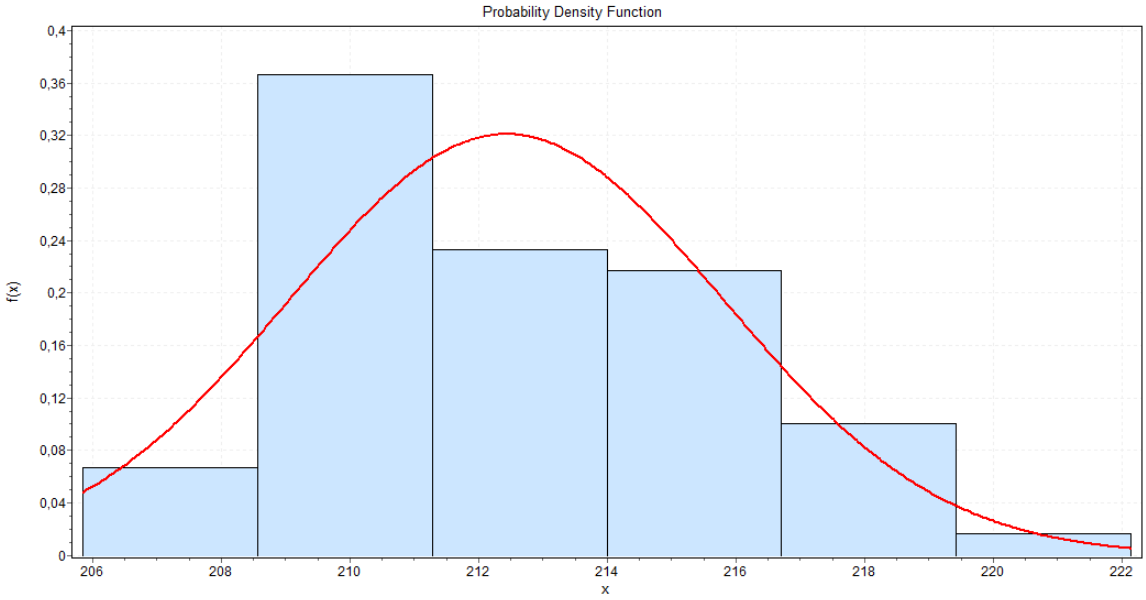


Figura 4 – Teste de Kolgomorov para a amostra de dados das Médias.

2) Suponha que os dados da Planilha2 provêm de uma amostra aleatória simples referente aos tempos de serviço dos caixas do Banco XYZ, localizado no endereço 123.

• Analise os dados através do histograma; do resumo estatístico dos dados que conste a média, mediana, moda (se houver), média das juntas, média do intervalo, desvio padrão, variância, quartís, intervalo inter-quartil, mínimo, máximo, assimetria e curtose.

Histograma

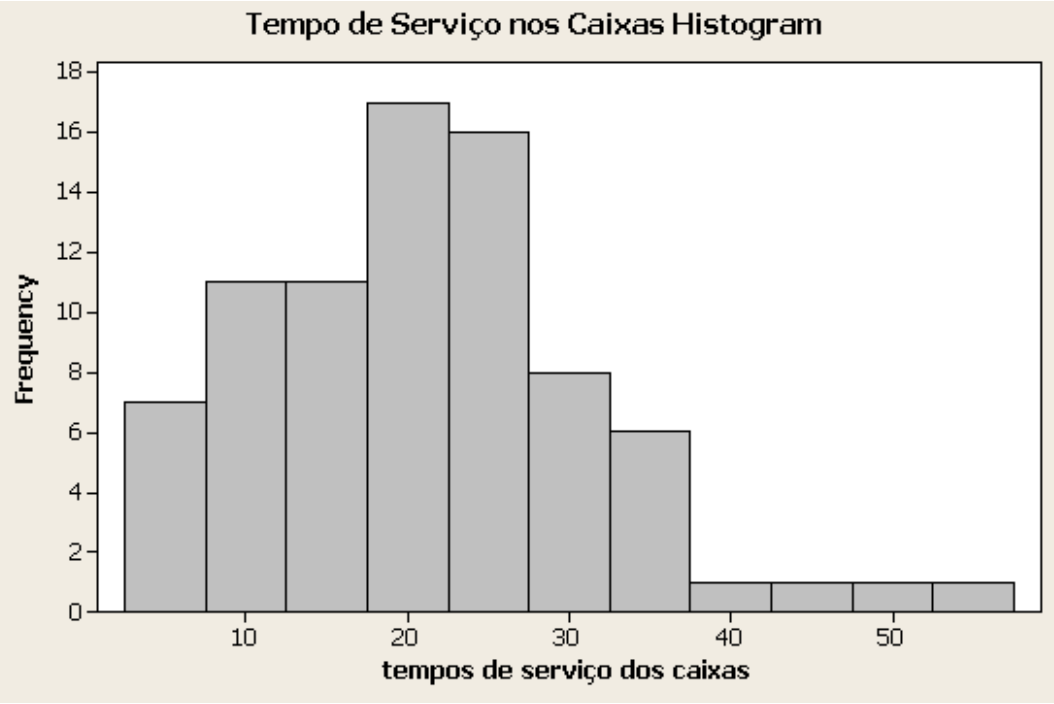


Figura 5 – Histograma dos dados da Planilha 2.

As estatísticas descritivas geradas são apresentadas a seguir:

Descriptive Statistics: Tempo de Serviço dos Caixas

Tempo Serviço	Média	DesvPadrao	Variância	Mínimo	1° Quartil	Mediana
	21,01	10,07	101,32	3,10	13,70	21,20

3° Quartil	Máximo	IQR	Moda	FreqModa	Assim	Curtose
26,20	52,80	12,50	13,7;21,6	3	0,60	0,78

MedJuntas	MedIntervalo
19,95	27,95

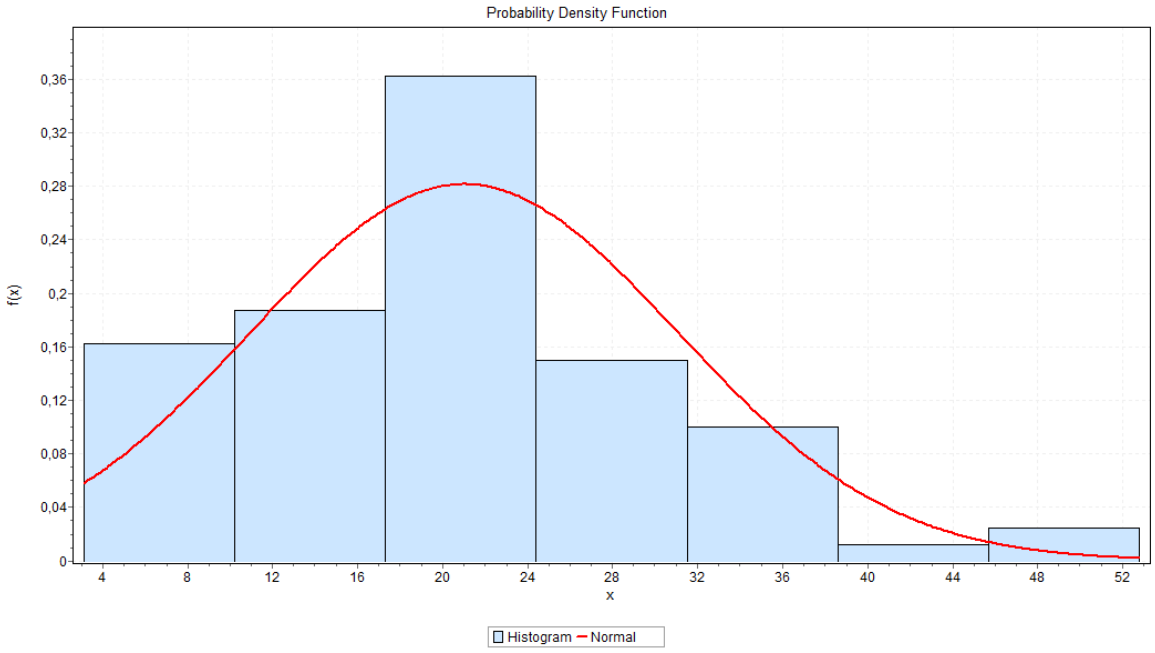
Média das Juntas = Q1+Q3/2 = 19,95
Média do Intervalo = Minimo+Maxino/2 = 27,95

• Faça um resumo textual que interprete os dados analisados, indicando a presença de possíveis outliers e informe se há evidências que indiquem se os dados provêm de uma população com distribuição normal.

Observando apenas o histograma, vê-se uma forte semelhança da distribuição dos dados com a distribuição normal. Além disso, só com histograma, pode ser observado a presença de *outliers*, uma vez que a cauda do gráfico é mais longa à esquerda.

Olhando para as estatíticas dos dados, a proximidade da distribuição de uma normal é comprovada, já que os valores da média, mediana e moda são extremamente próximos com 21,01, 21,20 e 21,6, respectivamente. Além disso, a média das juntas, que é também uma medida de tendência central, tem valor próximo ao das outras medidas anteriormente citadas. Além disso, a assimetria e a curtose com valores que não são muito distante de 0, corroboram com a inferência de que, apesar de não ser uma distribuição normal, a distribuição desses dados aproxima-se muito de uma.

Através da execução do Teste de aderência de Kolgomorov, a distribuição normal ocupou a 9ª posição em termos de semelhança com a distribuição real dos dados (no *ranking* de 60 posições). Esse resultado foi obtido com uma confiança, de 91,194%, conforme mostra a Figura 6.



Normal	0,08806	9

Figura 6 - Teste de Kolgomorov para a amostra de dados da Planilha 2.

No que diz respeito à presença de *outliers*, ela pode ser facilmente comprovada ao se observar o valor máximo dos dados, 52,8, muito maior do que a média. Outro valor que corrobora tal inferência é a média do intervalo, medida muito sensível a *outliers*, que tem um valor elevado em relação à média dos dados. Para ter mais subsídos quanto a afirmação feita, um gráfico boxplot foi feito, como mostrado na Figura 7. Esse gráfico mostra que, nesse sistema, os valores 58,8 e 49,5 são *outliers*.

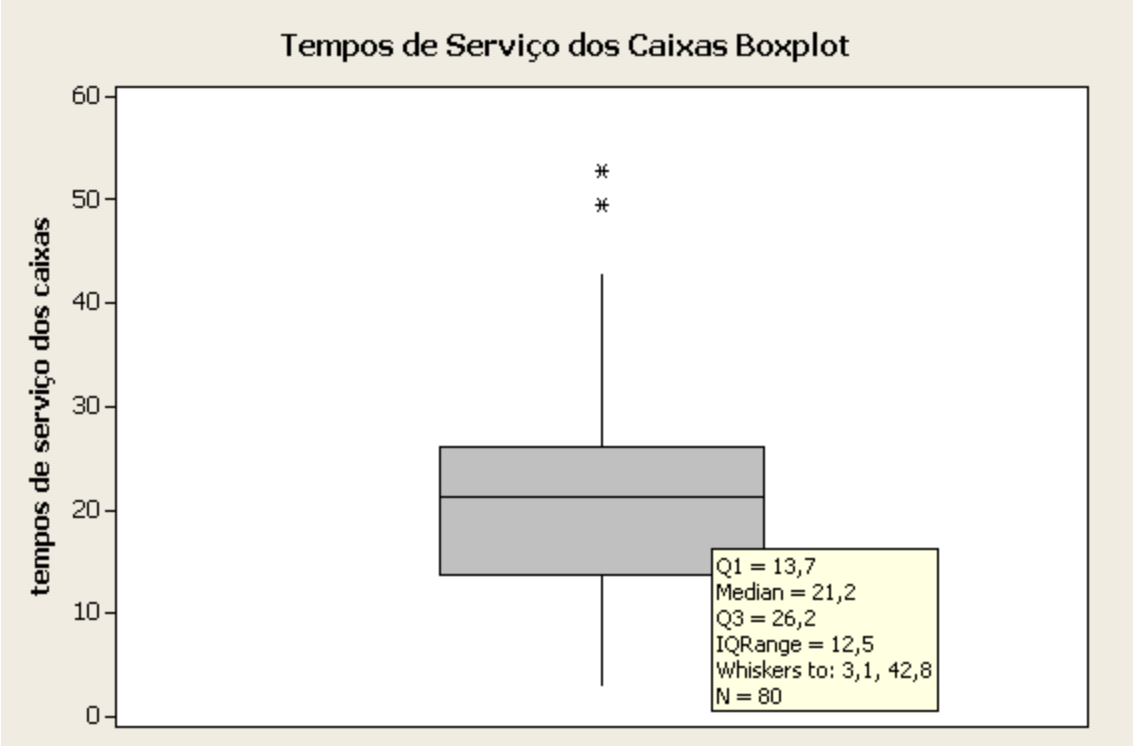


Figura 7 – *Boxplot* indicando a presença de *outliers*.

• Caso detecte outliers, retire-os dos dados e gere novamente o histograma e o resumo estatístico solicitado no passo a.

Histograma sem *outliers*

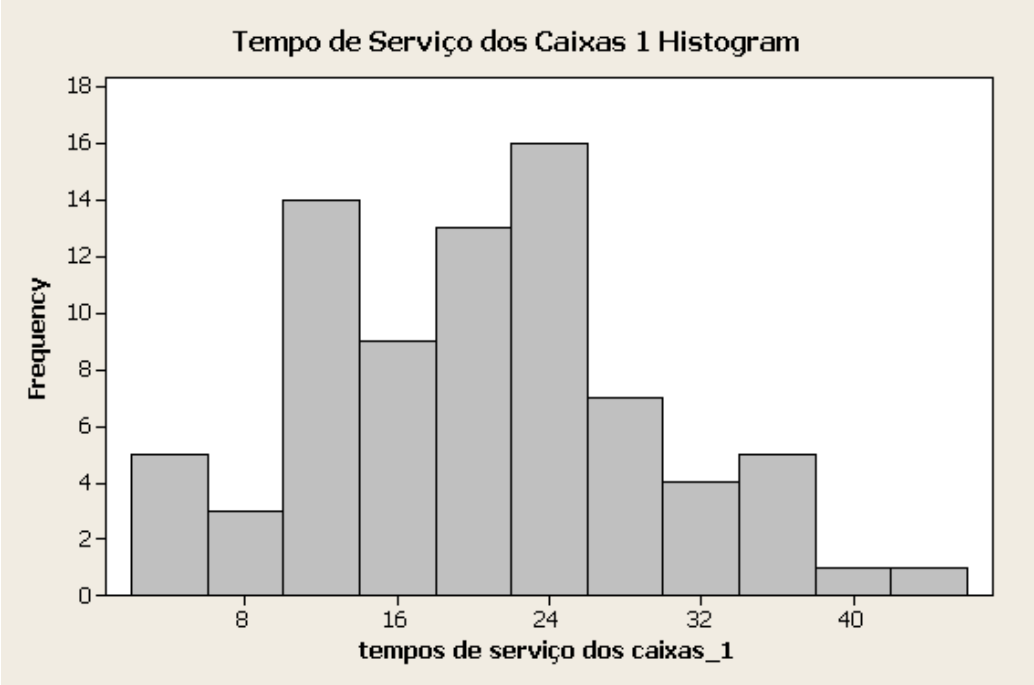


Figura 9 – Histograma dos dados das amostras sem *outliers*.

As estatísticas descritivas geradas são apresentadas a seguir:

Descriptive Statistics: tempos de serviço dos caixas_1

Tempo Serviço	Média	DesvPadrao	Variância	Mínimo	1° Quartil	Mediana
	20,23	8,93	79,67	3,10	13,67	21,10

3° Quartil	Máximo	IQR	Moda	FreqModa	Assim	Curtose
25,83	42,80	12,15	13,7;21,6	3	0,12	-0,37

MedJuntas	MedIntervalo
19,75	22,95

Média das Juntas = Q1+Q3/2 = 19,75
Média do Intervalo = Minimo+Maxino/2 = 22,95

• Faça um resumo textual que interprete os dados analisados e informe se há evidências que indiquem se os dados provêm de uma população com distribuição normal.

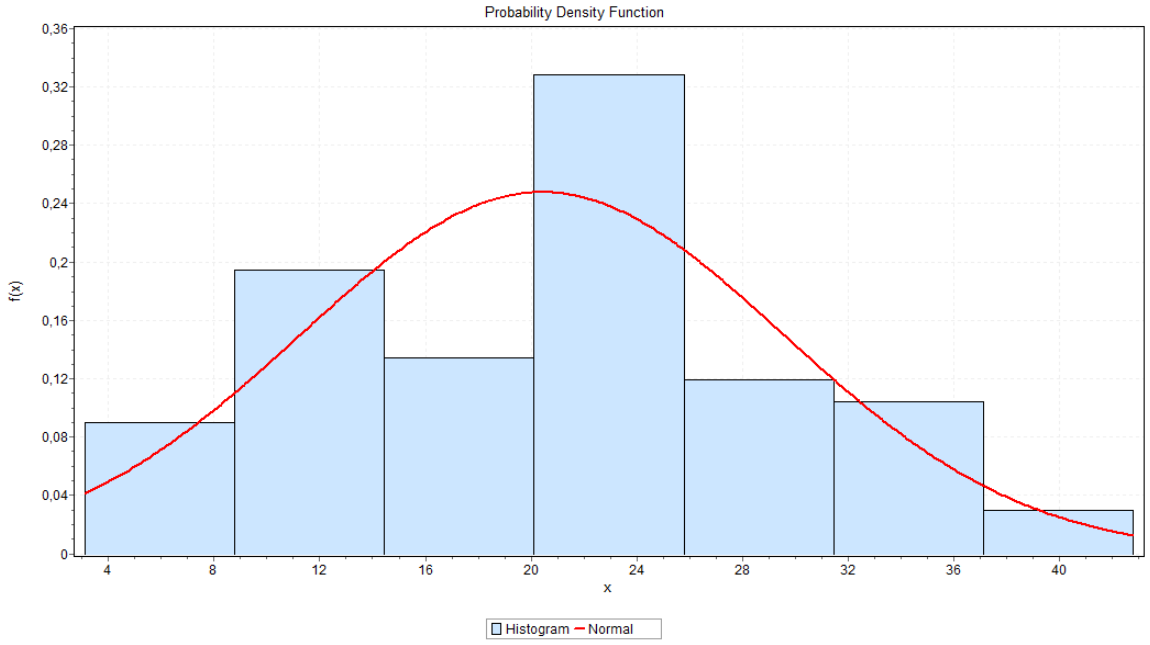
O novo histograma gerado a partir dos dados sem os *outliers* identificados anteriormente mostra uma semelhança maior com a distribuição normal. Uma vez

que os *outliers* foram removidos, a diferença apontada anteriormente na cauda da esquerda já não é mais tão grande.

Passando a observar os dados estatísticos, pode ser comprovado a característica da mediana de ser imune a *outliers*: o valor dela permanece inalterado. Como consequência da diminuição da diferença da cauda à esquerda, o valor da assimetria ficou mais próximo a zero. Ainda sobre as caudas, o histograma mostra que agora elas estão bem truncadas, curtas, e isso pode ser observado com a mudança de sinal que ocorreu no valor da curtose.

Outro ponto mostrado nas estatíscticas dos dados que pode comprovar a maior aproximação que a distribuição passou a ter da normal é o valor da média do intervalo. Como essa medida é muito suscetível a valores fora dos padrões dos dados (característica comum de qualquer média), obviamente ela sofreu modificação do seu valor, passando a ficar muito mais próxima dos valores da média, da mediana e da moda. Uma vez que essa medida é uma medida de tendência central, quanto maior sua proximidade dessas outras medidas, maior a similaridade da distribuição dos dados com a distribuição normal.

Através da execução do Teste de Kolgomorov, a distribuição ormal ocupou a 3ª posição no *ranking*. Em termos de confiança, foi obtido 90,566%, conforme mostra a Figura 10.



Normal	0,0815	3

Figura 10 – Teste de Kolgomorov para a amostra de dados da Planilha 2 sem *outliers*.

3) Descreva detalhadamente os benchmarks Dhrystone, Whetstone e Linpack.

Podemos dizer que **benchmark** é o ato de executar um programa de computador, um conjunto de programas ou outras operações, a fim de avaliar a performance relativa de um objeto, normalmente executando uma série de testes padrões e ensaios. Também é comumente usado para os próprios programas (de benchmarking) desenvolvidos para executar o processo. Normalmente, benchmarking é associado com avaliação de características de performance de um hardware de computador como, por exemplo, a performance da operação de ponto flutuante de uma CPU, mas há circunstâncias em que a técnica também é aplicável a software. Benchmarks de software são feitos, por exemplo, em compiladores ou sistemas de gerenciamento de banco de dados. Benchmarks provêm um método de comparação da performance de vários subsistemas dentre as diferentes arquiteturas de chips e sistemas. Benchmarking é útil para o entendimento de como o gerenciador de banco de dados responde sob a variação de condições.

Esses programas de teste de desempenho podem ser classificados de acordo com a classe de aplicação para a qual são destinados: computação científica, sistemas distribuídos, aplicações multimídia, processamento de sinais. Benchmarks são desenvolvidos para imitar um determinado tipo de comportamento em um componente ou sistema. Benchmarks "sintéticos" fazem isso através de programas especialmente criados para impor o tipo de comportamento desejado no componente. Em oposição, benchmarks "de aplicação" executam programas do "mundo real" no sistema. Embora normalmente os benchmarks de aplicação ofereçam uma melhor medida para a performance no "mundo real" para um dado sistema, benchmarks sintéticos ainda são usados no teste de componentes individuais, como um disco rígido ou um dispositivo de rede.

Dhrystone

O Dhrystone foi publicado pela primeira vez em 1984 por Reinhold Weicker da Siemens em C, Pascal e ADA, na tentativa de medir e comparar o desempenho dos computadores. O objetivo seria que com o Dhrystone, fosse criado um benchmark pequeno e representativo para programação de sistemas (no caso de operações aritméticas). O código do Dhrystone é em grande parte representado por aritmética simples, operações com strings, decisões lógicas e acessos de leitura e escrita de memória com intenção de refletir as atividades da CPU nas aplicações de computação de propósito mais geral. O resultado é determinado através do cálculo do tempo médio que um processador leva para executar as todas as iterações de um laço que, por sua vez, contém uma seqüência fixa de instruções que o compõem.

O programa compara o desempenho do processador ao de uma “máquina de referência”, o que é considerado por muitos como uma vantagem em relação à

utilização direta da métrica MIPS. A justificativa para tanto é que usar uma máquina de referência efetivamente compensa as diferenças na complexidade das instruções, onde comparar os números de MIPS de uma arquitetura RISC com os de uma CISC não é considerado válido por muitos pesquisadores. Como os processadores atuais são capazes de executar várias instruções por ciclo e dependem muito da velocidade e quantidade de memória *cache*, o resultado serve apenas como uma referência de desempenho bruto, que não indica necessariamente o desempenho do processador em aplicativos reais.

Inicialmente a indústria adotou o VAX 11/780 como uma máquina de referência de 1 MIPS. O VAX 11/780 alcança 1757 D/S (Dhrystone por segundo). O resultado do benchmark é calculado mensurando o número de D/S para o sistema, e dividindo-se este número por 1757 (da máquina de referência).

Podemos considerar que algumas das vantagens aparentes do Dhrystone também são fraquezas significativas. Os números do Dhrystone refletem na verdade o desempenho do compilador da linguagem C e suas bibliotecas, provavelmente mais do que o desempenho do próprio processador e seu projeto foi baseado na análise de vários outros programas escritos em diferentes linguagens e por diferentes autores, porém voltados à programação de sistemas operacionais e compiladores. Esta é uma característica bastante relevante, pois diferentes classes de aplicações enfatizam diferentes tipos de operações. De acordo com a freqüência das operações nos diferentes programas analisados, foram construídas algumas tabelas que deram origem ao Dhrystone. Em seu código original, em Ada, ele contém 100 sentenças entre o início e o fim do contador de tempo, balanceadas em relação a tipos de sentenças, tipos de dados (operandos) e sua localidade (global, local,parâmetro, constante). O programa contém uma distribuição de 51% de atribuições, 32% de sentenças de controle e 17% de chamadas de funções e procedimentos. O corpo do programa contém 12 procedimentos e, durante um laço (isto é, 1 Dhrystone), as 100 sentenças que compõem o laço são executadas.

O Dhrystone oferece como saída duas métricas: tempo em microsegundos e Dhrystone por segundo. As métricas são inversamente proporcionais, isto é, tempo em microsegundos – quanto maior o valor pior o desempenho; Dhrystone por segundo – quanto maior o valor melhor o desempenho. Seu objetivo é substituir o benchmark denominado Whetstone, mais antigo e menos confiável. O Dhrystone, como a maioria dos benchmarks, consiste em trechos de código padronizados, que são revistos periodicamente para minimizar vantagens que possam favorecer injustamente a determinadas combinações de hardware, compilador e ambiente. O Dhrystone se concentra no tratamento de strings e não utiliza operações com ponto flutuante. A exemplo da maioria dos testes de benchmark, ele é fortemente influenciado pelo projeto do hardware e do linker, a otimização do código, o uso de cache de memória, os estados de espera e os tipos de dados inteiros.

Para o Dhrystone existem duas versões do sistema, a 1.1 e a 2.1.

Versão 1.1 - Nesse caso de tratamento do relógio do PC, o mesmo é atualizado pela função clock() que executa a uma taxa de 18 vezes por segundo, com uma resolução de 0,05. As unidades utilizadas pelo benchmark na versão 1.1 são: Micro segundos, Dhrystones_Por_Segundo e Vax_Mips. O calculo para cada uma das unidades é dado a seguir:

```
Microseconds = benchtime * 1000000 / (double) Loops;
Dhrystones_Per_Second = (double) Loops / benchtime;
Vax_Mips = Dhrystones_Per_Second / 1757.0;
A função de contagem de inteiros é definida como:
Loops = Loops * 2;
      count = count - 1;
      Array2Glob[8][7] = 10;
       start_time();
       for (i = 0; i < Loops; ++i)
       {
             Proc5();
             Proc4();
             IntLoc1 = 2;
             IntLoc2 = 3;
             strcpy(String2Loc, "DHRYSTONE PROGRAM, 2'ND STRING");
             EnumLoc = Ident2;
             BoolGlob = ! Func2(String1Loc, String2Loc);
             while (IntLoc1 < IntLoc2)
             {
                   IntLoc3 = 5 * IntLoc1 - IntLoc2;
                   Proc7(IntLoc1, IntLoc2, &IntLoc3);
                   ++IntLoc1;
             }
             Proc8(Array1Glob, Array2Glob, IntLoc1, IntLoc3);
             Proc1(PtrGlb);
             for (CharIndex = 'A'; CharIndex <= Char2Glob; ++CharIndex)
                   if (EnumLoc == Func1(CharIndex, 'C'))
                         Proc6(Ident1, &EnumLoc);
             IntLoc3 = IntLoc2 * IntLoc1;
             IntLoc2 = IntLoc3 / IntLoc1;
             IntLoc2 = 7 * (IntLoc3 - IntLoc2) - IntLoc1;
             Proc2(&IntLoc1);
       }
       end_time();
       benchtime = secs;
```

Os testes envolvidos neste benchmark são: testes aritméticos, testes de overhead. As assinaturas a seguir resumem os procedimentos de testes:

```
void Proc0();
void Proc1(RecordPtr PtrParIn);
void Proc2(OneToFifty *IntParIO);
void Proc3(RecordPtr *PtrParOut);
void Proc4();
void Proc5();
void Proc6(Enumeration EnumParIn, Enumeration *EnumParOut);
void Proc7(OneToFifty IntParI1, OneToFifty IntParI2, OneToFifty *IntParOut);
```

```
void Proc8(Array1Dim Array1Par, Array2Dim Array2Par, OneToFifty IntParI1, OneToFifty IntParI2);
```

Versão 2.1 - A diferença está basicamente no tipo de teste que é feito em cima de leitura/escrita de arquivos, entre outros e também a forma que o autor chama de otimização de sistema. As funções assim como escritas na versão 1.1 são resumidas como a seguir:

```
void Proc_1 (REG Rec_Pointer Ptr_Val_Par);
void Proc_2 (One_Fifty *Int_Par_Ref);
void Proc_3 (Rec_Pointer *Ptr_Ref_Par);
void Proc_4 ();
void Proc_5 ();
void Proc_6 (Enumeration Enum_Val_Par, Enumeration *Enum_Ref_Par);
void Proc_7 (One_Fifty Int_1_Par_Val, One_Fifty Int_2_Par_Val, One_Fifty *Int_Par_Ref);
void Proc_8 (Arr_1_Dim Arr_1_Par_Ref, Arr_2_Dim Arr_2_Par_Ref, int Int_1_Par_Val, int Int_2_Par_Val);
Boolean Func_2 (Str_30 Str_1_Par_Ref, Str_30 Str_2_Par_Ref);
```

Também tem mudanças na medida de início de fim da estrutura de loop do benchmark que ao invés do `start_time()` e `end_time()` como na versão 1.1, a versão 2.1 definem como tipos: `Begin_Time`, `End_Time`, `User_Time`. Cada um desses tipos busca a função da macro `timer.h` da linguagem C.

Exemplo:

```
Begin_Time = dtime();
```

Mais um exemplo de que varia o exemplo de acordo com o compilador e a linguagem que vai ser utilizada. O benchmark originalmente foi escrito na linguagem ADA que não precisa de várias macros para sua execução. Na transição para a linguagem C/C++ esta influencia diretamente no resultados.

Ao final destes testes é apresentado um resumo dos resultados obtidos, onde temos uma variável Array2Glob, que possui um valor fixo pré-definido afim de atribuir um peso diferente para cada um dos testes executados além de apresentar o número de repetições executados durante os testes. Os resultados gerais são mostrados de forma mais simplificada, tornando possível a comparação entre diferentes máquinas com maior facilidade.

Exemplo:

```
Compiler        Watcom C/ C++ 10.5 Win386
Optimisation      -otexan -zp8 -5r
      10000 runs   0.00 seconds
     100000 runs   0.38 seconds
     200000 runs   0.66 seconds
     400000 runs   1.43 seconds
     800000 runs   2.80 seconds
    1600000 runs   5.55 seconds

Array2Glob8/7: O.K.     1600010
Microseconds for one run through Dhrystone:         3.47
```

```
Dhrystones per Second:                  288288
VAX  MIPS rating =                      164.08

Dhrystone Benchmark, Version 2.1 (Language: C)

Compiler        Watcom C/ C++ 10.5 Win386
Optimisation    -otexan -zp8 -fp5 -5r
Register option not selected

     10000 runs   0.05 seconds
    100000 runs   0.44 seconds
    200000 runs   0.94 seconds
    400000 runs   1.86 seconds
    800000 runs   3.68 seconds
   1600000 runs   7.36 seconds

Final values (* implementation-dependent):

Int_Glob:      O.K.  5  Bool_Glob:     O.K.  1
Ch_1_Glob:     O.K.  A  Ch_2_Glob:     O.K.  B
Arr_1_Glob[8]: O.K.  7  Arr_2_Glob8/7: O.K.     1600010
Ptr_Glob->             Ptr_Comp:       *    -5959980
  Discr:       O.K.  0  Enum_Comp:     O.K.  2
  Int_Comp:    O.K.  17 Str_Comp:      O.K.  DHRYSTONE PROGRAM, SOME STRING
Next_Ptr_Glob->        Ptr_Comp:       *    -5959980 same as above
  Discr:       O.K.  0  Enum_Comp:     O.K.  1
  Int_Comp:    O.K.  18 Str_Comp:      O.K.  DHRYSTONE PROGRAM, SOME STRING
Int_1_Loc:     O.K.  5  Int_2_Loc:     O.K.  13
Int_3_Loc:     O.K.  7  Enum_Loc:      O.K.  1
Str_1_Loc:                     O.K.  DHRYSTONE PROGRAM, 1'ST STRING
Str_2_Loc:                     O.K.  DHRYSTONE PROGRAM, 2'ND STRING

Microseconds for one run through Dhrystone:         4.60
Dhrystones per Second:                  217391
VAX  MIPS rating =                      123.73

The above results (less calibration details) are appended to file Dhry.txt,
        the version 2 results format being slightly different.
```

Linpack

Este benchmark foi produzido por Jack Dongarra, Jim Bunch, Cleve Moler e Gilbert Stewart, do pacote “LINPACK” das rotinas de álgebra linear. Tornou-se o benchmark básico para aplicações científicas desde a metade de 1980 com uma inclinação para o desempenho do supercomputador.

A versão original foi produzida em Fortran, porém uma versão em “C” apareceu posteriormente. A versão padrão “C” opera em matrizes 100x100, com precisão dupla ou simples e os tipos de loops podem ser rolled/unrolled. A versão

pré-compilada é de precisão dupla, rolled, otimizada ou não otimizada. Também existem as versões para DOS e OS/2.

O Linpack faz uso das bibliotecas BLAS (Basic Linear Algebra Subprograms) para realização de vetores básicos e operações de matrizes. É uma medida da potência de computação de ponto flutuante de um sistema. Ele mede o quão rápido um computador resolve um denso N por N sistema de equações lineares Ax = b, o qual é uma tarefa comum em engenharia. A solução é obtida por eliminação Gaussiana com giro parcial, com operações de ponto flutuante 2/3·N3 + 2·N2. O resultado é reportado em milhões de operações de pontos flutuantes por segundo (MFLOP/s, as vezes simplesmente chamado FLOPS).

Para sistemas de memória distribuída em larga escala, Linpack de alto desempenho, uma implementação portátil do benchmarck Linpack de alto desempenho, é utilizado como uma medida de desempenho para qualificar supercomputadores no ranking da lista dos computadores mais rápidos do mundo, TOP500. Agora também existe o ranking Green500 que lista as máquinas baseadas em eficiência energética, em FLOPS por Watt. O benchmark HCP roda para matrizes de diferentes tamanhos N procurando pelo Nmax no qual o máximo desempenho Rmax é obtido. O benchmark também reporta o problema de tamanho N1/2 onde metade do desempenho (Rmax/2) é alcançado.

Uma das rotinas que mais consomem tempo no Linpack é a DAXPY. Arquitetos de computadores projetam sistemas para aperfeiçoar DAXPY com o intuito de obter uma alta avaliação no Linpack. Existem reclamações não verificadas de que o Linpack não seja um bom benchmark para supercondutores porque ele não estressa a interconexão entre nós, mas foca em unidades aritméticas de pontos flutuantes e memória cachê.

Característica interessante do Limpack são a referenciação de duas rotinas : DGEFA e DGESL (estas são rotina que trabalha com ponto flutuante de 64 bits; já as SGEFA e SGESL trabalham normalmente com expressões de ponto flutuante de 32 bits). DGEFA realiza a decodificação parcial do vetor, e DGESL usa esse tipo de decodificação para resolver um determinado sistema de equações lineares. A maior parte das execuções de ponto flutuante gira em torno de $O(n^3)$, este é o tempo gasto em DGEFA. Uma vez que a matriz foi decomposta usando o DGESL, que trabalha com tempo de $O(n^2)$ operações de ponto flutuante.

Os resultados dos Benchmarks não devem ser utilizados como medidas de desempenho total do sistema (a não ser que a análise foi realizada suficiente para a carga de trabalho de um determinado interesse), mas sim como pontos de referências para novas avaliações.

O desempenho é freqüentemente medido em termos de Megaflops, Gigaflops, ou Teraflops. Incluem geralmente tanto adições e multiplicações na contagem das operações de ponto flutuante por segundo, e os valores dos operandos são assumidos de 64 bits de ponto flutuante. Atualmente, pode se medir o desempenho de um supercomputador chegando a Petaflops (1015flops/s).

Analisando o algoritmo Linpack e observando a forma de como os dados são referenciados, vemos que cada passo do processo de fatorização existe operações vetoriais que modificam completamente uma sub-matriz de dados. Esta

atualização faz um bloco de dados ser lido, atualizando e escrevendo de volta para a memória principal.

O número de operações de ponto flutuante é 2 / 3n3, e o numero de dados referenciados, ambos loads e stores, é 2 / 3n3. Assim, para cada par soma/ multiplicação temos de efetuar um load(carregar) e store(armazenar) dos elementos, infelizmente não obtendo a reutilização dos dados. Mesmo que as operações sejam completamente vetoriais, há uma movimentação significativa dos dados, resultando em um baixo desempenho. Em computadores vetoriais a um desempenho bem inferior ao pico estimado. Em computadores super-escalares isso resulta em uma grande quantidade de dados movimentados e atualizados. Para alcançar taxas de alto desempenho esta operação-memória-referência deve ser maior.

Whetstone

Esste benchmark foi desenvolvido na década de 70 pelo National Physical Laboratory no Reino Unido. Inicialmente foi desenvolvido em Algol 60, mas só ao ser implementado em FORTRAN que ele passou a ser amplamente usado e a definir padrões na indústria para a medição de desempenho de sistemas. Destinado a verificar o desempenho de processamento de computadores de pequeno e médio porte na execução de tarefas científicas, conta atualmente com implementações em diversas linguagens de alto nível como C e Java.

Antes do Whetstone, as análises de desempenho comparativas de processamento entre diferentes máquinas era realizado com a execução de instruções específicas de cada máquina, o que gerava resultados e análises pouco práticas. Além disso, otimizações feitas pelo compiladores não era levadas em conta nos testes. O Whetstone foi um grande avanço na época pois propunha uma comparação e análise de desempenho de processamento de diferentes máquinas a partir de teste realizados em linguagem de alto nível, abstraindo assim, das especificidades das diversas arquiteturas e dos eventuais ganhos promovidos pelos compiladores.

O benchmark Whetstone é simples e possui uma implementação padrão bastante pequena. Ele realiza diversos testes com inteiros e valores de ponto-flutuante para obter seus resultados. Apesar dessa simplicidade, ele pode ser expandido através de módulos e, com isso, ser capaz de analisar o desempenho da máquina diante de características de uma linguagem e de outros elementos importantes para uma boa performance computacional.

A simplicidade e maneira resumida através das quais a implementação padrão é feita são as principais vantagens desse benchmark. Essa mesma economia no tamanho do código que garante a vantagem do Whetstone sobre outros benchmarks, é o que gera sua principal desvantagem: por ser extremamente pequeno, ele só é capaz de analisar o que está na memória cache da máquina.

Como saída, o Whetstone, em sua forma mais básica, produz duas medidas: tempo em segundos e MIPS (Milhões de Instruções Por Segundo). Na

métrica tempo em segundo, quanto maior o valor, pior o desempenho; na MIPS, por sua vez, quanto maior o valor, melhor o desempenho. Existem versões do benchmark que produzem resultados em MOPS (Milhões de Objetos Por Segundo) e MFLOPS (Milhões de Instruções de Ponto Flutuante Por Segundo).

O algoritmo do Whetstone é composto por uma laço de repetição for onde as seguintes funções são chamadas (considerando uma implementação em C):

1. void whetstones(long xtra, long x100, int calibrate);
2. void pa(SPDP e[4], SPDP t, SPDP t2);
3. void po(SPDP e1[4], long j, long k, long l);
4. void p3(SPDP *x, SPDP *y, SPDP *z, SPDP t, SPDP t1, SPDP t2);
5. void pout(char title[22], float ops, int type, SPDP checknum, SPDP time, int calibrate, int section);
6. void printIt(SPDP val);

A primeira função, void whetstones(long xtra, long x100, int calibrate), funciona como o calibrador dos testes. Nela, as funções de número 2, 3, 4 e 5 são chamadas e instruções inteiras de valores aleatórios são geradas para compor matrizes que serão utilizadas na execução de testes. Também são geradas variáveis com valores de ponto flutuante. Já a função da linha 6, apenas serve para imprimir os resultados obtidos.

Diante das suas características, o benchmark Whetstone conseguiu encontrar vida longa e passou a ser incorporado em diversos outros tipos de testes e até hoje já foi adaptado para diversas linguagens e sistemas operacionais.