INSTITUTO FEDERAL DE MINAS GERAIS

Bacharelado em Ciência da Computação

Disciplina: Cálculo Numérico

Trabalho Prático

Professor: Diego Mello da Silva

Formiga-MG
1 de julho de 2016

Sumário

1 Introdução

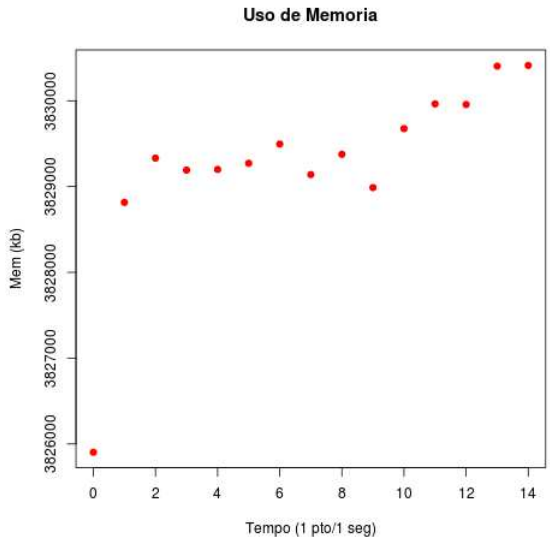
Este documento apresenta a especificação do Trabalho Prático da disciplina de Cálculo Numérico, no valor de 30 pontos, que cobre os assuntos Interpolação e Integração Numérica. O tabalho pode ser feito em grupo de até 2 alunos, com prazo de 4 semanas após anúncio. O trabalho consiste na implementação de um aplicativo capaz de calcular o valor médio de uma função contínua $f(x)$ no intervalo $\big[a, b\big]$. Tal valor pode ser obtido por meio do **Teorema do Valor Médio para Integrais** (TVMI). Por meio deles pode-se obter um valor médio para $f(x)$ no intervalo $\big[a, b\big]$ através da seguinte igualdade:

$$VM = \frac{1}{b-a}\int_a^b f(x)dx$$

Podemos usar essa informação para medir valores médios de diversos fenômenos ao longo do tempo, desde que a função $f(x)$ contínua e derivável no intervalo exista. Para exemplificar no contexto de Ciência da Computação considere um aplicativo que calcula o uso médio da memória do sistema no último minito. A idéia central de funcionamento deste aplicativo consiste em fazer a aquisição, em intervalos de tempo pré-determinados, a memória usada pelo sistema durante um horizonte de tempo determinado (por exemplo, fazer aquisição de 1 observação de uso de memória a cada 1 segundo, durante os últimos 60 segundos). Após a aquisição, cada par ordenado (x_i, y_i) obtido corresponderá à i-ésima leitura de uso de memória do sistema (o valor de x_i é o tempo da aquisição, e o valor de y_i é o uso da memória).

O problema deste tipo de aquisição consiste na informação que é perdida entre duas leituras consecutivas. Para exemplificar, sejam a tabela e figura a seguir, obtidas pela aquisição de 15 observações uso de memória (Kb), com intervalo de leitura de 1 segundo.

$x_i(ponto)$	$y_i(mem/kb)$
0	3825904
1	3828816
2	3829332
3	3829192
4	3829200
5	3829272
6	3829496
7	3829140
8	3829376
9	3828988
10	3829676
11	3829964
12	3829956
13	3830404
14	3830412

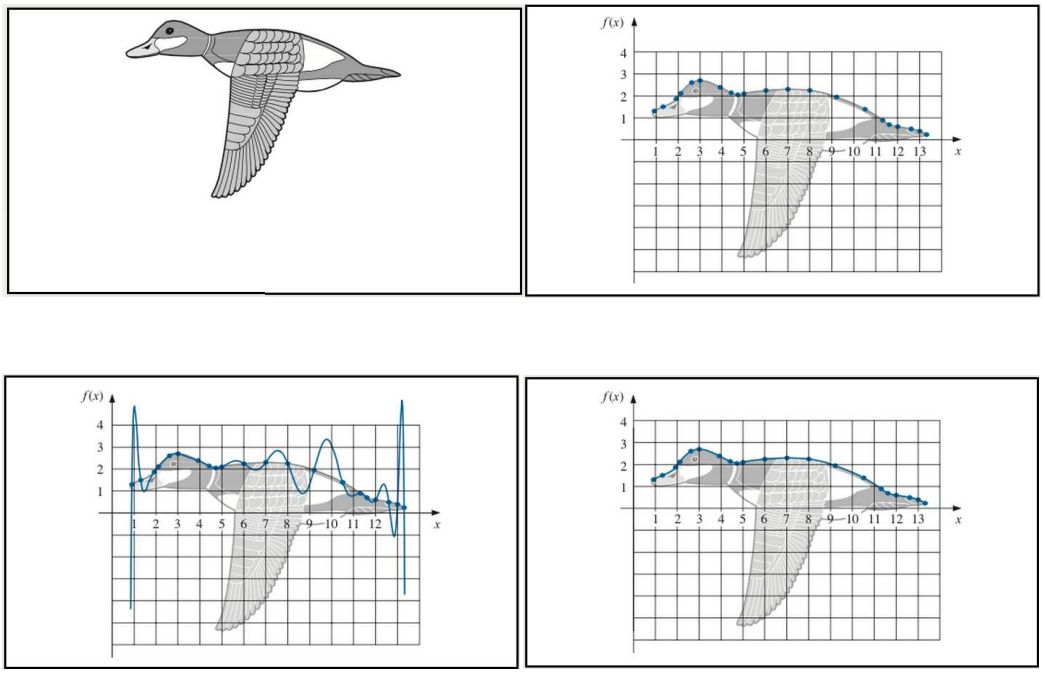


Pelas leituras realizadas sabemos quanto foi o uso da memória, em Kb, de 1 em

1 segundo. No entanto, pelos dados tabelados não é possível dizer exatamente qual foi a quantidade de memória usada entre o tempo de aquisição $t = 2$ segundos e $t = 3$ segundos, por exemplo. Da mesma forma, tampouco é possível dizer qual foi o uso de memória em $t = 4.82$ segundos. Apesar disso, é possível **estimar** qual teria sido o valor de uso da memória nestes exemplos se realizarmos uma interpolação dos valores tabelados.

Definição 1 (Interpolação) ***Interpolar*** *uma função $f(x)$ consiste em* ***aproximar*** *tal função por outra função $P(x)$, escolhida entre uma classe de funções definida a priori e que satisfaça algumas propriedades. A função $P(x)$ é usada em substituição à função $f(x)$.*

Diversos são os métodos de interpolação. [Campos] apresenta em sua obra conteúdo sobre interpolação polinomial pelos método de Lagrange, Newton e Gregory-Newton; interpolação Spline cúbicas e cúbicas naturais. A interpolação polinomial pode ser obtida com numerimente pelos métodos descritos acima, porém em virtude do grau do polinômio final $P(x)$ que aproxima $f(x)$ os resultados obtidos podem ser muito diferentes do esperado. Para exemplificar[1], seja um exemplo onde pretende-se aproximar a curvatura das costas de um pato selvagem. Na figura abaixo, canto superior esquerdo, está a imagem original do pato; na figura do canto superior direito está a mesma figura de fundo, com alguns pontos amostrados em um plano cartesiano; na figura do canto inferior esquerdo está o resultado da aproximação por meio de um polinômio interpolador (observe os valores do polinômio que fogem do esperado); por fim a figura do canto inferior direito apresenta o resultado da aproximação usando uma Spline, que é basicamente uma função por partes.



[1] Ver `http://www.isomorphism.io/NumericalMethods/lect_poly_interp.html#/61`

A função por partes mais simples é a função linear, onde dados dois pontos do intervalo o valor de $f(x)$ é aproximado por um polinômio de grau 1. Embora seja uma aproximação razoável, não tem o mesmo efeito visual de suavização que uma *spline* provê. Uma *spline* comum que utiliza polinômios de grau 3 entre pares sucessivos de pontos é denominada de *spline* cúbica. Segundo [Campos], o nome deriva do inglês e significa 'longas tiras de madeira' que eram usadas antigamente para interpolar de modo suave pontos de estruturas de aviões e navios. A próxima seção irá descrever os fundamentos básicos sobre *splines* cúbicas, assim como um pseudo-código de algoritmo para obter os coeficientes e avaliar $P(x)$ como uma função por partes.

2 *Splines* Cúbicas

Formalmente, dados $(n+1)$ pontos $(x_i, y_i), i = 0, 1, 2, \ldots, n$, com $x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n$ deseja-se construir n polinômios interpoladores cúbicos $s_i(x)$ denominados de ***splines* cúbicos** $s_i(x)$ que passem por dois pontos sucessivos (x_i, y_i) e (x_{i+1}, y_{y+1}). Desta forma, para cada intervalo $\left[x_i, x_{i+1}\right]$ haverá um polinômio da forma

$$\boxed{s_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i, i = 0, 1, 2, \ldots, (n-1)}$$

que satisfaçam as condições dadas a seguir para garantir que as *splines* cúbicas passem pelos pontos (x_i, y_i) tabelados:

1. $s_i(x_i) = y_i,$ $\qquad i = 0, 1, 2, \ldots, (n-1);$
2. $s_{n-1}(x_n) = y_n;$
3. $s_i(x_{i+1}) = s_{i+1}(x_{i+1}),$ $\qquad i = 0, 1, 2, \ldots, (n-2);$

Para garantir que as curvaturas e inclinações sejam contínuas, impõe-se:

1. $s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}),$ $\qquad i = 0, 1, 2, \ldots, (n-2);$
2. $s_i''(x_{i+1}) = s_{i+1}''(x_{i+1}),$ $\qquad i = 0, 1, 2, \ldots, (n-2);$

Em seu livro, [Campos] traz passo a passo como determinar os coeficientes a_i, b_i, c_i e d_i de cada uma das n equações possíveis utilizando *splines* naturais. No entanto, esta discussão está fora do escopo do presente trabalho de implementação; para detalhes consultar o Capítulo 3, seções 3.8 e 3.9 (para construção de um sistema linear capaz de prover o resultado das derivadas) e 3.11 (para entendimento sobre o algoritmo que, dados os cálculos para construção e resolução do sistema, avaliar um valor de x dado no intervalo dos valores de domínio tabelados). O cálculo das derivadas segundas $s_i''(x_i)$ é realizado por meio de Eliminação de Gauss e substituição retroativa sobre um sistema tridiagonal simétrico, e será apresentado no pseudo-código do Algoritmo 1. A avaliação de $P(x)$ envolve uma busca binária para determinar o intervalo (x_i, x_{i+1}) onde o valor de x é informado, seguido de uma avaliação usando o método de Horner, sendo seu pseudo-código apresentado no Algoritmo 2. Em ambos os casos o método completo contêm as operações de

resolução do sistema e avaliação do polinômio cúbico correspondente embutidas no próprio código.

O Algoritmo 1 calcula as derivadas segundas dos polinômios interpoladores utilizando o número de pontos tabelados n, e uma sequência ordenada de valores tabelados x e y. Neste código a solução do sistema linear tridiagonal será as derivadas segundas em questão; observe que elas serão usadas a posteriori na avaliação de $P(x)$ pelo Algoritmo 2 devendo ser armazenada em alguma estrutura de dados global ou informada via parâmetro. É dever da dupla estabelecer como serão os detalhes de implementação para a tradução deste pseudo-código em linguagem computacional (como passar os parâmetros, em que índice os vetores da linguagem são iniciados, etc).

Algorithm 1 CalculaDerivadaSpline(n, $x[]$, $y[]$)

1: **if** $(n < 3)$ **then**
2: `imprime-erro-menos-3-pontos`
3: **end if**

4: {Sistema Tridiagonal Simétrico}
5: $m \leftarrow n - 2$
6: $Ha \leftarrow x[2] - x[1]$
7: $DeltaA \leftarrow (y[2] - y[1])/Ha$
8: **for** $(i \leftarrow 1$ **to** $m)$ **do**
9: $Hb \leftarrow x[i+2] - x[i+1]$
10: $DeltaB \leftarrow (y[i+2] - y[i+1])/Hb$
11: $e[i] \leftarrow Hb$
12: $d[i] \leftarrow 2(Ha + Hb)$
13: $s2[i+1] \leftarrow 6(DeltaB - DeltaA)$
14: $Ha \leftarrow Hb$
15: $DeltaA \leftarrow DeltaB$
16: **end for**

17: {Eliminação de Gauss}
18: **for** $(i \leftarrow 2$ **to** $m)$ **do**
19: $t \leftarrow e[i-1]/d[i-1]$
20: $d[i] \leftarrow d[i] - t * e[i-1]$
21: $s2[i+1] \leftarrow s2[i+1] - t * s2[i]$
22: **end for**

23: {Substituição Retroativa}
24: $s2[m+1] \leftarrow s2[m+1]/d[m]$
25: **for** $(i \leftarrow m$ **to** 2 **step** $-1)$ **do**
26: $s2[i] \leftarrow (s2[i] - e[i-1] * s2[i+1])/d[i-1]$
27: **end for**
28: $s2[1] \leftarrow 0$
29: $s2[m+2] \leftarrow 0$

30: **return** $(s2)$

O Algoritmo 2 calcula uma avaliação de $P(valor)$ para $valor \in [x_0, x_n]$.

Algorithm 2 AvaliaSpline(n, $x[]$, $y[]$, $s2[]$, $valor$)

1: **if** $(valor < x[0]$ **or** $valor > x[n])$ **then**
2: `imprime-erro-fora-intervalo`
3: **end if**

4: {Busca Binária}
5: $inf \leftarrow 1$
6: $sup \leftarrow n$
7: **while** $(sup - inf < -1)$ **do**
8: $indice \leftarrow$ `trunc`$((inf + sup)/2)$
9: **if** $(x[indice] > valor)$ **then**
10: $sup \leftarrow indice$
11: **else**
12: $inf \leftarrow indice$
13: **end if**
14: **end while**

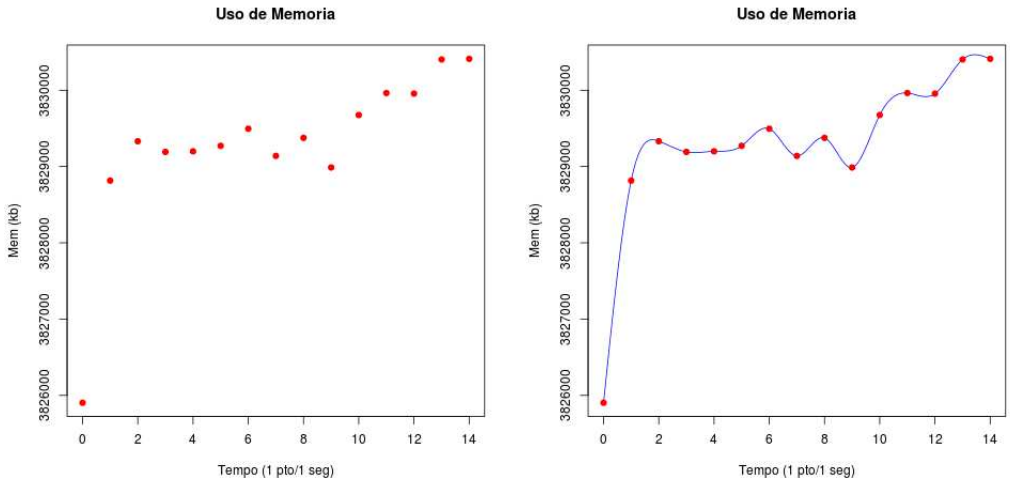
15: {Avaliação de $P(x)$ por Horner}
16: $h \leftarrow x[sup] - x[inf]$
17: $a \leftarrow (s2[sup] - s2[inf])/6 * h$
18: $b \leftarrow s2[inf] * 0.5$
19: $c \leftarrow (y[sup] - y[inf])/h - (s2[sup] + 2 * s2[inf]) * h/6$
20: $d \leftarrow y[inf]$
21: $h \leftarrow valor - x[inf]$
22: $resultado \leftarrow ((a * h + b) * h + c) * h + d$

23: **return** $(resultado)$

Uma vez implementado, o código acima deve ser usado como segue. Primeiramente, com os valores de x_i e y_i tabelados calcula-se o vetor que contêm o valor da derivada segunda dos polinômios de grau 3 para cada intervalo $h = x_{i+1} - x_i$. Isto é feito com uso do Algoritmo 1. De posse deste vetor, para cada valor de x a avaliar $P(x)$ dentro do intervalo tabelado usa-se o Algoritmo 2, que indentifica o intervalo onde $x_i < x < x_{i+1}$, calcula os coeficientes a_i, b_i, c_i e d_i e avalia o polinômio de grau 3 dado por $s_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i$ usando o método de Horner, com fórmula fechada para grau 3. Para o cálculo dos coeficientes deste polinômio usa-se os valores de derivada calculados previamente.

A interpolação por *spline* cúbica será usada para obter uma aproximação ou estimativa de qual foi o consumo de memória do sistema nos intervalos de tempo onde não ocorreram leituras de uso de memória.

Para exemplificar, seja a figura a seguir com os valores do exemplo inicial deste documento. Enquanto que a figura à esquerda contêm apenas os valores lidos durante a aquisição, a figura da direita contêm os valores interpolados para estimar uma leitura em valores dentro do intervalo.



3 Integração Usando Monte Carlo

De posse da *spline* cúbica é possível obter uma estimativa de medida de uso de memória em qualquer valor do domínio considerado. Usaremos esta informação para obter o cálculo da integral numérica necessária para obter o valor médio de uso de memória, segundo o Teorema de Valor Médio para Integrais (TVMI). Para tal será utilizado o método denominado Monte Carlo. O Método de Monte Carlo (MMC) é qualquer método estatístico que se baseia em amostragem aleatória massiva para obter resultados numéricos por meio de simulações repetidas. Dentre as diversas aplicações deste método, existe seu uso na integração numérica definida.

A Integração de Monte Carlo escolhe aleatoriamente as coordenadas (x, y) de n pontos amostrados segundo uma distribuição uniforme. Os valores de x e y são sorteados de acordo com os limites de um retângulo que enquadra a função $f(x)$ cuja integral definida $\int_a^b f(x)d(x)$ está sendo calculada. Após sortear as coordenadas dos n pontos aleatoriamente, com $x_i \in \left[a, b\right]$ e $y_i \in \left[0, h\right]$ onde h é um limite superior para $f(x)$ e $i = 1, 2, \ldots, n$ determina-se quantos pontos sorteados satisfazem $\boxed{y_i \leq f(x_i)}$. Seja a área do retângulo de altura h que inscreve a função $f(x)$ no intervalo $\left[a, b\right]$ dada por $A = (b - a) * (h - 0)$, e que se obtenha o percentual de pontos gerados abaixo da curva então a estimativa de valor de integral definida é dado pela estimativa da área abaixo da curva calculada por:

$$\boxed{\int_a^b f(x)d(x) \approx \left(\frac{\text{ptos abaixo da curva}}{\text{total de pontos}}\right) * h(b - a)}$$

Para exemplificar, seja a função $f(x) = e^x \sin(10x) + 8$, cuja integral definida no intervalo $\left[0.4, 2.0\right]$ é calculada **analiticamente** por 12.48287. Ao executar o cálculo de integração de Monte Carlo sorteando n pontos aleatórios com valor de $x \in \left[0.4, 2.0\right]$ e $y \in \left[0.0, 14.0\right]$, obteve-se os resultados **estimados** apresentados nas figuras a seguir, para $n = 500$, $n = 1000$, $n = 5000$, $n = 25000$, $n = 50000$ e $n = 100000$. Observe como o valor do erro absoluto EA reduz conforme n aumenta.

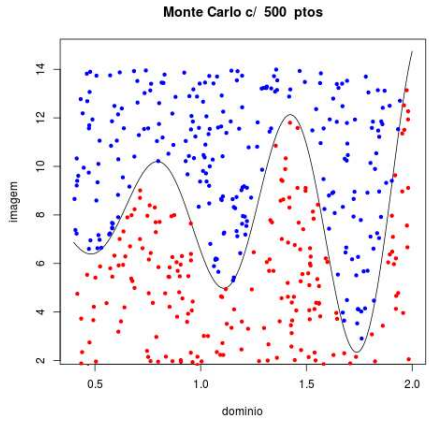


Figura 1: $\int \approx 12.9472,\ EA = 0.4643268$

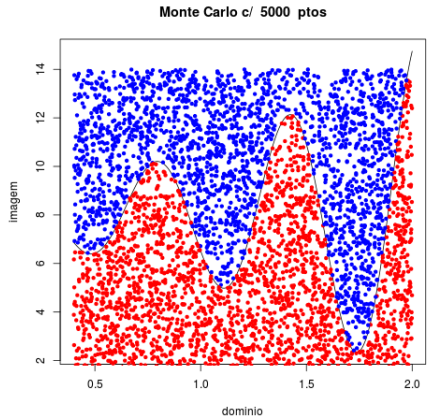


Figura 2: $\int \approx 12.6112,\ EA = 0.1283268$

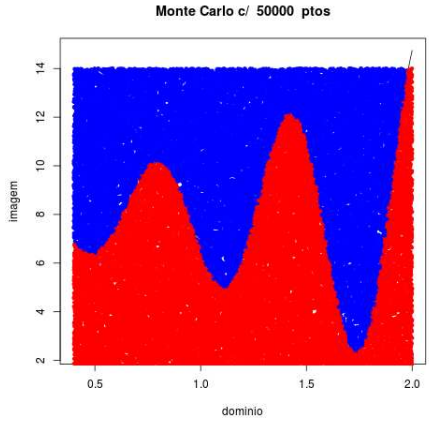


Figura 3: $\int \approx 12.48531,\ EA = 0.002438823$

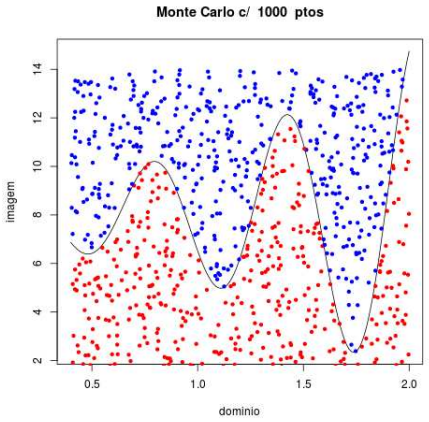


Figura 4: $\int \approx 12.6336,\ EA = 0.1507268$

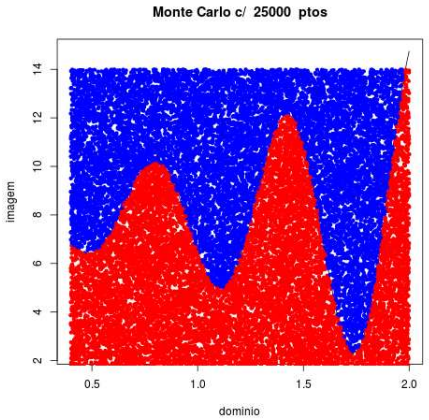


Figura 5: $\int \approx 12.48666,\ EA = 0.003782823$

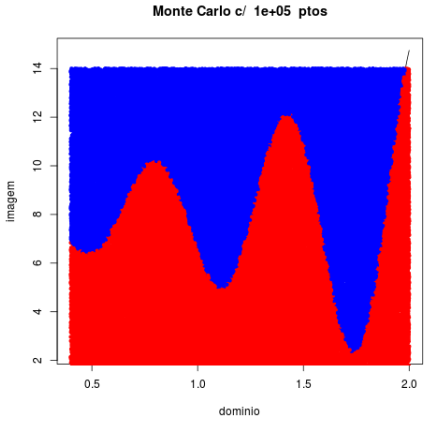


Figura 6: $\int \approx 12.48287,\ EA = 0.000249177$

Neste trabalho o método de Monte Carlo será usado para estimar a integral definida para a função $P(x)$ que interpola os valores de uso de memória obtidos do sistema. Este resultado será usado em conjunto com o Teorema do Valor Médio para Integrais (TVMI) para se obter o uso médio da memória.

O pseudo-código genérico para o Método de Monte Carlo é dado a seguir, no Algoritmo 3. São parâmetros de entrada do método o número de pontos n, os valores limites de x, os valores limites de y e a função $f(x)$ cuja integral está sendo calculada.

Algorithm 3 IntegralMonteCarlo$\big(n,\, x_{min},\, x_{max},\, y_{min},\, y_{max},\, f(x)\big)$

$num.abaixo \leftarrow 0$

for $(i \leftarrow 1$ **to** $n)$ **do**
  $x \leftarrow$ sorteio a partir de $U(x_{min}, x_{max})$
  $y \leftarrow$ sorteio a partir de $U(y_{min}, y_{max})$
  if $\big(y \leq f(x)\big)$ **then**
    $num.abaixo \leftarrow num.abaixo + 1$
  end if
end for

$AreaTotal \leftarrow (x_{max} - x_{min})(y_{max} - y_{min})$

$$Area \leftarrow AreaTotal\left(\frac{num.abaixo}{n}\right)$$

return $(Area)$

Por motivos óbvios será preciso adaptar o pseudo-código acima para operar com a função de avaliação da *spline* cúbica que estima os valores de $P(x)$.

4 Aquisição de Uso de Memória no Linux Ubuntu

A aquisição das leituras de uso de memória será usada neste trabalho para obter o uso médio de memória do sistema. Para tal optou-se por este trabalho desenvolver um *shell script* para o SO Linux Ubuntu, mais especificamente um *script* Bash[2][3], que permite interfacear com o Ubuntu através de uma linguagem de comando que permite mesclar uso de aplicativos linha de comando com variáveis, estruturas de repetição e estrutura de decisão.

O Bash surgiu em 1989 como um *shell* Unix, que se destacou rapidamente e hoje é encontrado em diversas distribuições Unix-*based*, incluindo o Linux Ubuntu, e pode ser instalado em outros SOs (como é o caso do aplicativo Cygwin do Windows e do terminal de algumas versões do OS X). Dentre as características principais destaca-se a possibilidade de guardar o resultado de comandos executados no terminal em variáveis, declaração de arrays, uso de variáveis de ambiente do sistema, realizar cálculos com números inteiros, estruturas de decisão, estruturas de repetição, criação

[2] `http://www.tldp.org/LDP/Bash-Beginners-Guide/Bash-Beginners-Guide.pdf`
[3] `http://www.tldp.org/LDP/abs/abs-guide.pdf`

de funções simples, e integração com qualquer aplicativo linha de comando. Estes elementos foram todos utilizados no *script* de aquisição.

Para realizar a aquisição utilizou-se uma estratégia simples, de consultar um arquivo específico do sistema de arquivos do Ubuntu localizado na pasta `/proc`. Esta pasta não contêm arquivos físicos de verdade, mas sim arquivos virtuais usados pelo sistema operacional para prover ao usuário informações sobre os processos e sistema. É uma forma conveniente e padronizada de fornecer dados sobre processos de maneira dinâmica sem ter que realizar acesso à memória do *kernel* do Linux. Os arquivos presentes na pasta `/proc` atuam como interface para as estruturas internal do kernel, e podem ser usados para obter informações sobre o mesmo; logo é uma forma de comunicação entre o espaço do *kernel* e o espaço do usuário sem a necessidade explícita de uso de *system calls*.

Dentro da pasta `/proc` existem pastas numeradas, uma para cada processo em execução, nomeadas pelo seu PID (*process identification*). Nestas pastas existem arquivos específicos onde pode-se consultar a linha de comando que foi invocada para iniciar o processo (`/proc/PID/cmdline`), nomes e valores das variáveis de ambiente que afetam este sistema (`/proc/PID/environ`), o arquivo executável que iniciou o processo (`/proc/PID/exe`), os descritores de arquivo usados por este processo (`/proc/PID/fd`), uma imagem da memória virtual usada por este processo (`/proc/PID/mem`), informações básicas sobre o estado do processo e memória usada (`/proc/PID/status`), dentre outros. Existem também arquivos mais gerais, presentes na raíz da pasta `/proc` que podem ser usados para consultar informações sobre o sistema como um todo. São exemplos de informações que podem ser obtidas: quais os barramentos usados do computador (`/proc/bus`), informações sobre o processador e seus núcleos tais como *clock*, *cache* e outros (`/proc/cpuinfo`), dispositivos presentes na máquina (`/proc/devices`), discos lógicos (`/proc/diskstats`), sistemas de arquivo suportados pelo *kernel* (`/proc/filesystems`), resumo sobre como o *kernel* gerencia a memória do sistema (`/proc/meminfo`), dispositivos montados e em qual partição (`/proc/mounts`), entre outros. Alguns possuem permissão de leitura, e podem ter seu conteúdo exibido através do comando `cat`; outros exigem aplicativos especiais para leitura.

Conforme descrito acima, dentro desta pasta existe um arquivo virtual que pode ser usado para consultar o uso de memória do sistema. Trata-se do arquivo `/proc/meminfo`. Um exemplo de seu conteúdo é dado a seguir, obtido apos execução do comando `cat /proc/meminfo`:

```
MemTotal:        3955584 kB
MemFree:          811048 kB
Buffers:          368536 kB
Cached:          1286020 kB
SwapCached:            0 kB
Active:          1795288 kB
Inactive:        1110904 kB
Active(anon):    1300832 kB
Inactive(anon):   262204 kB
Active(file):     494456 kB
Inactive(file):   848700 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
```

```
Dirty:               200 kB
Writeback:             0 kB
AnonPages:       1251680 kB
Mapped:           187676 kB
Shmem:            311404 kB
Slab:             123688 kB
SReclaimable:      86476 kB
SUnreclaim:        37212 kB
KernelStack:        3936 kB
PageTables:        38528 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     1977792 kB
Committed_AS:    4583232 kB
VmallocTotal:   34359738367 kB
VmallocUsed:       358824 kB
VmallocChunk:   34359374812 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:       66288 kB
DirectMap2M:     4040704 kB
```

Dentre os campos existentes, existem dois em especial que interessam para o propósito deste trabalho. São os campos `MemTotal` e `MemFree`, que mostram os valores de memória total disponível (neste caso, 3955584 kB = 3862 mB = 3.77 gB ≈ 4 gB) e o total de memória livre (neste caso, 811048 kB = 792 mB). O total de memória usada é, portanto, obtida pela diferença entre os valores `MemTotal - MemFree`. Para obter estes valores será preciso fazer um *pipe* (comando `|`), repassando a saída do comando `cat` como entrada para o comando `grep`, que filtra e exibe as linhas com a *substring* informada.

```
user@machine$ cat /proc/meminfo | grep "MemTotal:"

MemTotal:        3955584 kB
```

Como resultado, apenas a linha que contêm a *string* `MemTotal` é apresentada no terminal. Ainda assim não interessa a linha toda, mas sim apenas o segundo campo (o primeiro é o nome da variável do sistema, o segundo seu valor, e o terceiro a unidade de medida). Para obter somente a segunda *string* podemos usar o comando `awk`, muito usada para processar textos e manipular arquivos em ambientes Unix. A seguinte linha de comando repassa a saída do arquivo `/proc/meminfo` como entrada para o comando `grep`, cuja saída é repassada como entrada para o comando `awk`, que repassa ao terminal apenas o valor da segunda coluna da linha que contem a *substring* `MemTotal`.

```
user@machine$ cat /proc/meminfo | grep "MemTotal:" | awk ‘{print $2}’

3955584
```

Por padrão o resultado irá para o terminal. No entanto, com uso de *shell script* podemos guardar o resultado do terminal em uma variável fazendo-se a atribuição do comando à variável, como visto no exemplo a seguir. Uma vez armazenado em variável podemos manipular este conteúdo usando comandos específicos do Bash.

```
TOTAL=$(cat /proc/meminfo | grep "MemTotal:" | awk '{print $2}')
```

Outro recurso que este *script* de aquisição irá usar é o direcionamento de saída para arquivos. Serão usados os direcionamento `>`, para criar um arquivo novo e inserir o conteúdo especificado pelo comando `echo`, e o comando `»`, que concatena o conteúdo original do arquivo com o novo conteúdo informado, inserido ao final do mesmo.

O *script* construído com essa linguagem é denominado `mem-collector.sh` e utiliza parâmetros por linha de comando para instruir o *script* como fazer a aquisição. Os parâmetros seguem a seguinte sintaxe:

```
user@machine$ ./mem-collect.sh <num. aquisições> <obs. por rep.> <intervalo> <saida>
```

onde

- **número de aquisições**: consiste em quantas aquisições independentes serão feitas. Cada aquisição gera um arquivo de saída diferente, numerado e com extensão `.dat`;
- **observações por aquisição**: indica a quantidade de leituras a serem realizadas em cada aquisição independente;
- **intervalo**: consiste no intervalo entre duas leituras consecutivas, em segundos. Aceita resoluções menores do que 1 segundo;
- **saída**: preâmbulo do nome do arquivo de saída. A cada aquisição serão concatenados ao final deste preâmbulo o número da aquisição (iniciando-se em zero), seguido da extensão `.dat`

Para exemplificar, seja o exemplo onde deseja-se realizar 12 aquisições, cada uma delas com 100 leituras de uso de memória coletadas a cada 250 ms. O resultado será salvo em arquivos cujo preâmbulo é `coleta-memoria` (e irá gerar 12 arquivos com nomes de `coleta-memoria-0.dat` a `coleta-memoria-11.dat`):

```
user@machine$ ./mem-collect.sh 12 100 0.25 coleta-memoria
```

Uma versão didática do *script* de aquisição utilizado é dado a seguir. Primeiramente, obtem-se os parâmetros informados ao *script* por linha de comando. Em seguida dá-se início às aquisições. Em cada aquisição cria-se um arquivo de saída que utiliza o preâmbulo informado, concatenado com o número da aquisição e extensão. Em seguida, inicia-se as leituras de memória. Obtem-se a memória total e memória livre do sistema, calcula-se a diferença entre as mesmas, direciona o par ordenado (leitura, memória usada) para arquivo de saída e paraliza a coleta por uma quantidade de tempo informada ao comando `sleep`. O processo se repete até que todas as leituras tenham sido realizadas, assim como todas as aquisições previstas e informadas via linha de comando. Para cada aquisição gera-se um arquivo denominado `plot.r` que contêm comandos da linguagem R para gerar um gráfico de dispersão com os valores coletados em formato `.png`.

```
#################################################################################
#                                                                               #
# mem-collect.sh                                                                #
#                                                                               #
# Shell script escrito em BASH que coleta informacoes periodicas de uso de memoria #
# por meio do arquivo /proc/meminfo (campos 'MemTotal' e 'MemFree').            #
#                                                                               #
# Durante o processamento gera arquivo 'plot.r' de forma a gerar grafico de saida  #
# em formato .png com o resultado da aquisição (Memoria vs. Tempo)              #
#                                                                               #
# Requer:                                                                       #
#                                                                               #
#    AWK: sudo apt-get install gawk                                             #
#      R: sudo apt-get install r-base r-base-dev                                #
#                                                                               #
#################################################################################

# Verifica se os parametros informados estao corretos
# Se nao informaram parametros corretamente
if [ $# -ne 4 ];
then
    echo
    echo "Erro.Usar: mem-collect.sh <aquis.> <obs> <interv> <arq saida>"
    echo
    echo "Exemplo: ./mem-collect.sh 3 10 1 dados.txt"
    echo
    exit -1;
fi

# Obtem parametros informados
REPETICOES=$1
NUMPTOS=$2
INTERVALO=$3
ARQSAIDA=$4

# Cria contador de repeticoes
REPCOUNT=0

# Repete a coleta de acordo com o especificado
until [ $REPCOUNT -ge $REPETICOES ];
do
    # Cria o arquivo de saida
    echo -n "" > $ARQSAIDA-$REPCOUNT".dat"

    # Executa a aplicacao a quantidade de repeticoes especificada
    CONTADOR=0
    until [ $CONTADOR -ge $NUMPTOS ];
    do
        echo "Coletando obs $CONTADOR..."
```

```
        # Filtra as linhas de interesse
        TOTAL=$(cat /proc/meminfo | grep "MemTotal:" | awk '{print $2}')
        LIVRE=$(cat /proc/meminfo | grep "MemFree:"  | awk '{print $2}')

        # Calcula a memoria usada pela diferenca da total - livre
        let USADO=TOTAL-LIVRE

        # Salva tabela em arquivo de saida
        echo "$CONTADOR  $USADO" >> $ARQSAIDA-$REPCOUNT".dat"

        # Aguarda o intervalo entre uma coleta e outra
        sleep $INTERVALO

        # Incrementa contador
        let CONTADOR=CONTADOR+1
    done

    # Cria script R automaticamente para plotar o resultado
    # da coleta
    echo '#' > plot.r
    echo "#Script gerado automaticamente por mem-collector.sh" >> plot.r
    echo '#' >> plot.r
    echo "dados <- read.table(file='$ARQSAIDA-$REPCOUNT.dat');" >> plot.r
    echo "png('$ARQSAIDA-$REPCOUNT.png', width=1200);" >> plot.r
    echo 'plot(dados$V1, dados$V2, type="l", main="Uso de Memoria",
         xlab="Tempo (1 pto/'$INTERVALO' seg)", ylab="Mem (kb)");' >> plot.r
    echo 'points(dados$V1, dados$V2, col="red", pch=20);' >> plot.r
    echo "dev.off();" >> plot.r

    # Executa R script do terminal
    Rscript plot.r > /dev/null

    # Incrementa contador
    let REPCOUNT=REPCOUNT+1
done
```

No site da disciplina foi disponibilizado um conjunto de arquivos de entrada disponibilizados no arquivo `dataset.zip`[4] que foram gerados através deste coletor de uso de memória. Para as validações deste trabalho prático pode-se usar estes arquivos, ou criar novos usando o próprio *shell script* descrito neste material. Para tal, escreva as instruções acima em um arquivo ASCII, salve-o com o nome de `mem-collect.sh` e altere sua permissão para execução através do comando `chmod 755 mem-collect.sh`. Para executá-lo, basta invocar o *script* através do terminal (`./mem-collect.sh`) e informar os parâmetros (i) número de aquisições diferentes realizar; (ii) número de observações (leituras) em cada aquisição; (iii) o intervalo entre duas leituras consecutivas, em segundos (pode ser menor que 1 segundo); e por fim o nome do arquivo de saída que irá gerar a tabela de valores (x_i, y_i) para a interpolação *spline*.

5 Resumo do Processo

O processo de cálculo do valor médio de uso de memória pode ser resumido nos seguintes passos:

1. Obter leituras de uso da memória por meio do *script* `mem-collector.sh` ou de algum dos arquivos gerados e salvos em `dataset.zip`;

[4] `https://goo.gl/x6It51`

2. Cálculo das derivadas da *spline* cúbica que aproxima o uso da memória por uma função por partes $P(x)$ com base nos valores de tempo e uso de memória obtidos no Passo 1;

3. Cálculo da área sobre a curva de $P(x)$ no intervalo onde o uso da memória foi coletado, para determinar numericamente o valor da integral definida;

4. Cálculo do uso médio da memória por meio do Teorema do Valor Médio para Integrais (TVMI) dividindo-se o valor da integral numérica estimada por Monte Carlo pela diferença $(b - a)$, que é a amplitude do intervalo de integração;

5. Apresentação do resultado ao usuário.

A próxima seção tratará da especificação da aplicação que resolve este problema, apresentado os requisitos que deverão ser implementados para atingir este objetivo. Na última seção será apresentado o barema de correção com os pontos de cada requisito, totalizando 30 pontos.

6 Especificação

Esta seção apresenta as recomendações gerais e requisitos esperados para a implementação do aplicativo `avg-memory` descrito neste trabalho prático de implementação. Detalhes sobre cada requisito, assim como o barema de correção serão apresentados ao longo da seção.

A implementacao do aplicativo deverá ser feita em um único arquivo de código fonte, denominado `avg-memory.pas` ou `avg-memory.c` de acordo com a linguagem de programação escolhida pelo grupo. Para organizar o código, deve-se fazer uso de procedimentos e funções capazes de implementar os requisitos do *software*, um requisito por função quando for aplicável. As próximas sub-seções apresentam os requisitos deste trabalho prático.

6.1 Requisito 01 - TAD para Sequência Pares Ordenados

Esta aplicação manipula pares ordenados do tipo (x, y) para realizar seus cálculos e plot de gráficos. Para tal pede-se neste requisito que o grupo crie uma estrutura de dados (TAD) capaz de armazenar um par ordenado de valores em ponto flutuante. Além disso, pede outra TAD para armazenar a sequencia de pares ordenados. A TAD de sequência deverá conter:

- O número de pares ordenados da sequência
- Uma coleção de pares ordenados
- O menor valor da coordenada X
- O maior valor da coordenada X
- O menor valor da coordenada Y

- O maior valor da coordenada Y

Esta estrutura de dados será a base para a manipulação dos dados desta aplicação. Além dos campos solicitados, o grupo poderá acrescentar novos campos de acordo com a necessidade.

6.2 Requisito 02 - Entrada de Dados (Arquivo)

A entrada e saída de dados será feita por meio de arquivos e parâmetros informados via linha de comando (ver no *slide* de introdução à disciplina como pode-se implementar em C e Pascal a leitura de argumentos por linha de comando). A sintaxe esperada para o `avg-memory` é:

```
user@machine:$ ./avg-memory <arquivo entrada> <arquivo saída>
```

São requeridos 2 (dois) parâmetros, que correspondem:

- caminho do arquivo de entrada que contêm pares ordenados (x_i, y_i), para $i = 1, 2, \ldots, n$ coletados a partir do *script* `mem-collector.sh`, em formato ASCII, com duas colunas sendo a primeira o número da leitura, e a segunda o valor de memória usada no momento em que ocorreu a leitura.
- caminho do arquivo de saída contendo *script* em linguagem R que será gerado automaticamente pelo calculador para plotar o resultado da análise em gráfico no formato `.png`.

Neste requisito deve-se implementar a leitura de arquivo de entrada. Para tal será preciso (i) determinar a quantidade de linhas do arquivo e (ii) realizar a leitura dos pares ordenados (x_i, y_i) em estrutura de dados na aplicação. A leitura da quantidade de linhas pode ser feita lendo-se caracter por caracter do arquivo, contabilizando-se a quantidade de caracteres `'\n'` (incluídos no final de cada linha de arquivos ASCII). Sugere-se que o arquivo seja lido duas vezes; a primeira para determinar a quantidade de linhas, e a segunda para obter os dados da aquisição informada via linha de comando.

6.3 Requisito 03 - Cálculo das 2a. Derivada da *Spline* Cúbica

Para estimar os valores de $P(x)$ não-tabelados pode-se usar a aproximação por meio de uma *spline* cúbica. Para tal é preciso implementar um algoritmo que construa um sistema linear tridiagonal simétrico com valores específicos cuja solução resulta na derivada segunda de cada um dos intervalos da *spline*, que é uma função por partes. As operações relacionadas com este requisito são: a geração do sistema, sua transformação usando Eliminação de Gauss e sua resolução por meio de substituição retroativa. Neste requisito pede-se para implementar o Algoritmo 1, que realiza as três operações em um único pseudo-código.

6.4 Requisito 04 - Função que Avalia $P(x)$ usando Splines

O cálculo das derivadas segunda, descrito no Requisito 03, permite interpolar os resultados de $P(x)$ quando avaliando a função em um valor não tabelado. Sabendo que $P(x)$ é uma função por partes, é preciso construir um algoritmo que identifique qual intervalo de $P(x)$ está o valor que devemos interpolar (para ser eficiente pode-se usar uma adaptação da busca binária), seguido do cálculo dos coeficientes a_i, b_i, c_i e d_i usados para determina a interpolação $s_i(x) = a_i(x-x_i)^3+b_i(x-x_i)^2+c_i(x-x_i)+d_i$ por meio do método de Horner. O Algoritmo 2 descrito em seção anterior apresenta um pseudo-código com estes cálculos, e deve ser implementado para cumprir este requisito. Uma vez implementado será usado no Requisito 06, para avaliar $f(x)$ numericamente, e no Requisito 09 pelo método de Monte Carlo, para gerar os vetores `xspl` e `yspl` no plot do gráfico de uso médio de memória.

6.5 Requisito 05 - Gerador de Números Uniforme $U(min, max)$

Em quase todas as linguagens computacionais existe um gerador de números inteiros uniforme. É comum que estes geradores sejam capazes de gerar números aleatórios inteiros no intervalo $\left[0, MAX\_RAND\right]$ que sejam equiprováveis. Neste requisito pede-se para construir uma função $U(min, max)$ que seja capaz de gerar resultados em ponto flutuante dentro de um intervalo $\left[min, max\right]$ especificado.

Isso pode ser feito (i) normalizando o resultado inteiro para que o mesmo esteja contido no intervalo $\left[0, 1\right]$ e (ii) redistribuindo o resultado obtido em escala no intervalo $\left[min, max\right]$. Obtem o resultado em um intervalo $\left[0, 1\right]$ pode ser obtido realizando-se uma divisão de ponto flutuante entre o valor sorteado no intervalo $\left[0, MAX\_RAND\right]$ pelo maior valor possível de ser gerado, MAX_RAND. Logo,

$$\boxed{U(0,1) = \frac{U(0, MAX\_RAND)}{MAX\_RAND}}$$

De posse da construção de $U(0, 1)$ pode-se gerar números aleatórios uniformes entre $U(min, max)$ por meio de

$$\boxed{U(min, max) = U(0, 1)(max - min) + min}$$

6.6 Requisito 06 - Integral Numérica por Monte Carlo

Conforme apresentado neste material, para calcular o uso médio de memória do sistema é preciso determinar o valor da integral definida da função $P(x)$. Uma das formas mais simples de calcular numericamente a integral definida de uma função é por meio de Integração por Monte Carlo, já descrito neste documento e cujo pseudo-código foi apresentado no Algoritmo 3. Neste requisito pede-se que a integração numérica descrita seja implementada para obter o valor de $\int_a^b f(x)d(x)$.

6.7 Requisito 07 - Uso Médio de Memória usando TVMI

O Teorema do Valor Médio para Integrais (TVMI) permite obter um valor médio de uma função $f(x)$ no domínio $\big[a, b\big]$. De posse do valor de $\int_a^b f(x)d(x)$ obtida do Requisito 07 é possível obter o valor médio da função em questão. Para o caso específico deste trabalho, que mede uso de memória do sistema e cujo valor de $f(x)$ é aproximado por $P(x)$ via interpolação *spline* cúbica, o valor médio resultante será considerado o valor médio de uso da memória de um sistema computacional. Neste requisito este valor deverá ser calculado e armazenado para apresentação em terminal (Requisito 08) e na construção de *script* em linguagem R para plotar o uso da memória médio e instantâneo (Requisito 09).

6.8 Requisito 08 - Saída de Dados (Terminal)

Após processar o documento e determinar o valor médio de uso de memória o aplicativo `avg-memory` irá apresentar a saída de dados em terminal e em arquivo. Este requisito trata somente da saída em console. Seja um arquivo de entrada no formato especificado neste documento que contêm as seguintes observações:

```
0    3825904
1    3828816
2    3829332
3    3829192
4    3829200
5    3829272
6    3829496
7    3829140
8    3829376
9    3828988
10   3829676
11   3829964
12   3829956
13   3830404
14   3830412
```

Ao executar o aplicativo especificado neste documetno via terminal espera-se obter como saída para este exemplo hipotético o formato que segue.

```
user@machine$ ./avg-memory dados.txt saida

Number of Samples    : 15
Average Memory Usage : 3829375.844 Kb

Run 'Rscript saida.r' to generate Average Memory Usage Chart
```

6.9 Requisito 09 - Saída de Dados (R Script)

Além de saída no terminal, o aplicativo deverá gerar automaticamente um *script* na linguagem R para gerar o gráfico de uso de memória. Este *script* conterá (i) os

pontos originais, (ii) os pontos interpolados, (iii) o valor médio. O arquivo de saída utilizará o parâmetro informado por linha de comando acrescido na extensão `.r`, e conterá os seguintes comandos como referência:

```
#
# Generated automatically by "avg-memory" application
#

# Original points (x coordinates)
xorig <- c(
        0,
        1,
        2,
        3,
        4,
        5,
        6,
        7,
        8,
        9,
        10,
        11,
        12,
        13,
        14
);

# Original points (y coordinates)
yorig <- c(
        3825904,
        3828816,
        3829332,
        3829192,
        3829200,
        3829272,
        3829496,
        3829140,
        3829376,
        3828988,
        3829676,
        3829964,
        3829956,
        3830404,
        3830412
);

# Spline points (x coordinates, sampling interval = 0.01)
xspl <- c(
        0,
        0.01,
        0.02,
        0.03,
        0.04,
        0.05,
        0.06,
        0.07,
        0.08,
        .
        .
        .
        13.92,
        13.93,
        13.94,
        13.95,
        13.96,
        13.97,
        13.98,
        13.99,
```

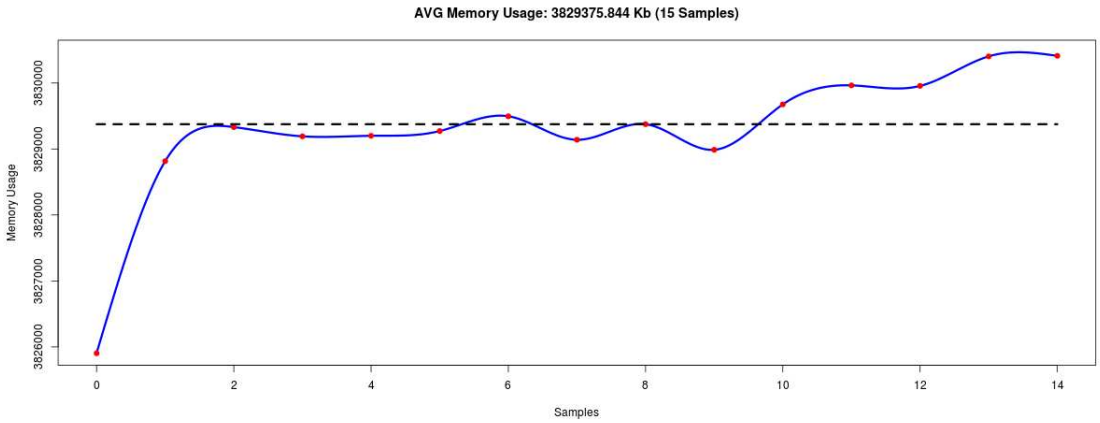
```
        14
);

# Spline points (y coordinates, sampling interval = 0.01)
yspl <- c(
        3825904,
        3825939.03823,
        3825974.072909,
        3826009.100485,
        3826044.117407,
        3826079.120125,
        3826114.105085,
        3826149.068738,
        3826184.007532,
        .
        .
        .
        3830423.480106,
        3830422.061103,
        3830420.635696,
        3830419.204799,
        3830417.769329,
        3830416.330199,
        3830414.888324,
        3830413.444619,
        3830412
    );

    # Average Memory Usage
    AvgMemory <- 3829375.844;

    # Plot the values in .png file
    png(file="saida.png", width=1200);
    title <- paste("AVG Memory Usage: 3829375.844 Kb (15 Samples)");
    plot(xspl, yspl, type="l", col="blue", main=title, xlab="Samples", ylab="Mem. Usage", lwd=3);
    points(xorig, yorig, pch=19, col="red");
    lines( c(min(xorig), max(xorig)), c(AvgMemory, AvgMemory), col="black", lty=2, lwd=3);
    dev.off();
```

Este parágrafo descreve como criar o *script* de saída. Os pontos originais serão usados para criar os vetores `xorig` e `yorig`; os pontos interpolados serão usados para criar os vetores `xspl` e `yspl` (neste exemplo, a quantidade de pontos interpolados no intervalo $\left[0, 14\right]$ de 0.01 em 0.01 contabilizam 1401 observações e ocupam espaço no documento, o ... omite a grande maioria das observações interpoladas); o valor de uso médio da memória obtido pelo TVMI será usado para preencher a variável `AvgMemory`, e o nome do arquivo de saída será usado no comando `png` para gerar um arquivo com extensão `.png`. Para criar este arquivo de saída construa *strings* linha a linha, no formato dos comandos dados acima, e salve-as no arquivo de saída com extensão `.r`. Executar `Rscript saida.r` resulta em:



6.10 Requisito 10 - Documentação e Corretude

Para que o trabalho prático tenha cumprido com sua proposta é preciso que a implementação do aplicativo `avg-memory` gere resultados corretos. Neste requisito deve-se verificar o resultado dos métodos para diferentes entradas de dados através da inspeção visual da saída resultante da execução do script em linguagem R. Além da corretude, é preciso que o código fonte esteja bem documentado por meio de comentários, descrevendo todas as estruturas de dados, modularização dos requisitos e partes importantes do código. Somente pontuará neste requisito o grupo que completar os testes com 100% de corretude nos resultados.

7 Barema

A tabela a seguir descreve o barema que contêm o valor de cada requisito do trabalho. Cabe ressaltar que trabalhos fora do especificado (codificação, linguagem, compilação e ambiente) não serão corrigidos e valerão zero.

Requisito	**Ptos**
Req 01 - TAD para Sequência de Pares Ordenados	2.0
Req 02 - Entrada de Dados (Arquivo)	3.0
Req 03 - Cálculo da 2a. Derivada da Spline Cúbica	2.0
Req 04 - Função que Avalia $P(x)$ usando Splines	2.0
Req 05 - Gerador de Números Uniformes $U(min, max)$	1.0
Req 06 - Integral Numérica por Monte Carlo	2.0
Req 07 - Uso Médio de Memória usando TVMI	1.0
Req 08 - Saída de Dados (Terminal)	2.0
Req 09 - Saída de Dados (R Script)	5.0
Req 10 - Documentação e Corretude	10.0
TOTAL	**30.0**

8 Bibliografia Recomendada

Referências

[Campos] CAMPOS FILHO, Frederico Ferreira. Algoritmos Numéricos, 2^a edição. Editora LTC (Grupo GEN), 2007. ISBN: 85-21615-37-8.