Relatório: IronMan

André Meneghelli Vale - 4898948
andredalton@gmail.com

Marcello Souza de Oliveira - 6432692
mcellor210@gmail.com

17 de abril de 2013

1 Compilação, Sistema e Implementação

O exercício programa entregue, bem como quaisquer testes, foram realizados em sistemas baseados em Debian com diversos tipos de arquiteruras diferentes e em um servidor instalado com SUSE Linux Enterprise Server 11.

1.1 Compilação

Junto com o código, há um `Makefile`. A compilação é feita usando o comando `make`. Por praticidade foram criandas duas targets extras `all` e `clear` que compilam todos os executáveis e limpa todo o conteúdo gerado na compilação automaticamente. Também é possível gerar um relatório automático que atualiza este relatório com os valores calculados para o computador do leitor. Para tanto basta executar o `./relatorio.sh`.

1.2 Como utilizar

Para rodar o programa, após tê-lo compilado, executar:

```
./ep1 -debug [entrada]
```

Onde:
`-debug` significa que o programa irá rodar em modo de depuração e imprimirá uma linha com a posição de todos os atletas a cada minuto. Caso contrário irá imprimir os três primeiros colocados de cada categoria a cada 30 minutos.
`[entrada]` um arquivo que contém os dados sobre a quantidade de atletas e a condição do terreno na prova de ciclismo.
OBS: Os arquivos estão localizados no diretório `entradas/`. O formato do arquivo de entrada deve seguir a descrição do enunciado do problema em `ep1.pdf`, não foram tratados erros de formatação do arquivo de entrada.

1.3 Códigos

O código foi fragmentado em 3 arquivos .c principais e mais dois arquivos de cabeçalho:

- `ep1.c`: Contém as duas funções que definem a classificação e os atletas e mais duas funções principais, as quais são escolhidas em tempo de compilação:
  - `ironMain`: É a função principal para a versão *paralela* e para a versão *concorrente* do problema.

  - `gutsMain`: É a função principal para a versão *iterativa* do problema.

- `tempos.c` e `tempos.h`: Contém a geração aleatória dos tempos de cada atleta, respeitando os limites estipulados no enunciado do problema.

- `defines.c` e `defines.h`: Contém a maioria dos defines, as estruturas de dados usadas no problema e ainda uma função principal para realizar uma simulação interativa do algorítimo da punição. A utilização deste executável será tratada adiante.

1.4 Executáveis

Foram desenvolvidos 3 algorítmos diferentes para solucionar o problema e cada um destes algorítimos foi implementado com precisão de segundos e milisegundos. A seguir segue uma breve descrilção de cada um deles:

Além disso, utilizei a reorganização de contas para melhorar a estabilidade numérica [GSM].

2 Exemplo 1

Nesta seção, algumas considerações e respostas sobre o primeiro exemplo.

2.1 Distância

A distância entre Ax e b é justamente a norma do vetor $b - Ax$ que calculamos com o programa. Seu valor é aproximadamente 1.82216.

3 Exemplo 2

Nesta seção, algumas considerações e respostas sobre o segundo exemplo.

3.1 Formulação do problema

Dado o polinômio cúbico $c_0 + c_1 s + c_2 s^2 + c_3 s^3$, obtemos o sistema:

$$\begin{bmatrix} 1 & s_1 & s_1^2 & s_1^3 \\ & \dots & & \\ 1 & s_{11} & s_{11}^2 & s_{11}^3 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 75.995 \\ \dots \\ 281.422 \end{bmatrix}$$

Onde s_i é $(t_i - 1950)/50$. Substituindo, temos aproximadamente:

$$\begin{bmatrix} 1.00 & -1.00 & 1.00 & -1.00 \\ 1.00 & -0.80 & 0.64 & -0.51 \\ 1.00 & -0.60 & 0.36 & -0.22 \\ 1.00 & -0.40 & 0.16 & -0.06 \\ 1.00 & -0.20 & 0.04 & -0.01 \\ 1.00 & 0.00 & 0.00 & 0.00 \\ 1.00 & 0.20 & 0.04 & 0.01 \\ 1.00 & 0.40 & 0.16 & 0.06 \\ 1.00 & 0.60 & 0.36 & 0.22 \\ 1.00 & 0.80 & 0.64 & 0.51 \\ 1.00 & 1.00 & 1.00 & 1.00 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 75.995 \\ 91.972 \\ 105.711 \\ 123.203 \\ 131.669 \\ 150.697 \\ 279.323 \\ 203.212 \\ 226.505 \\ 249.633 \\ 281.422 \end{bmatrix}$$

3.2 Resultado

Resolvendo o sistema, obtemos o vetor aproximado:

$$\begin{bmatrix} 155.90427 \\ 100.36592 \\ 23.72614 \\ 1.26294 \end{bmatrix}$$

Donde, fazendo as devidas substituições, obtemos o valor aproximado 312.691 para o ano de 2010. O gráfico abaixo dá uma ideia melhor sobre o ajuste da função aos pontos.

Figura 1: Ajuste dos pontos ao polinômio.

4 Exemplo 3

Nesta seção, algumas considerações e respostas sobre o terceiro exemplo.

4.1 Formulação do problema

Dado $z = 0$ e $f = 1$, obtemos o polinômio de duas variáveis $ax^2 + bxy + cy^2 + dx + ey$ que resulta no sistema:

$$\begin{bmatrix} x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 \\ & & \dots & & \\ x_{10}^2 & x_{10}y_{10} & y_{10}^2 & x_{10} & y_{10} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} = \begin{bmatrix} -1 \\ \dots \\ -1 \end{bmatrix}$$

Substituindo, temos aproximadamente:

$$\begin{bmatrix} 1.04 & 0.40 & 0.15 & 1.02 & 0.39 \\ 0.90 & 0.30 & 0.10 & 0.95 & 0.32 \\ 0.76 & 0.23 & 0.07 & 0.87 & 0.27 \\ 0.59 & 0.17 & 0.05 & 0.77 & 0.22 \\ 0.45 & 0.12 & 0.03 & 0.67 & 0.18 \\ 0.31 & 0.08 & 0.02 & 0.56 & 0.15 \\ 0.19 & 0.06 & 0.02 & 0.44 & 0.13 \\ 0.09 & 0.04 & 0.01 & 0.30 & 0.12 \\ 0.03 & 0.02 & 0.02 & 0.16 & 0.13 \\ 0.00 & 0.00 & 0.02 & 0.01 & 0.15 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

4.2 Resultado

Resolvendo o sistema, obtemos o vetor aproximado:

$$\begin{bmatrix} 2.25379 \\ 0.00632 \\ 5.52218 \\ -1.28981 \\ -7.37735 \end{bmatrix}$$

O erro obtido foi de aproximadamente 0.02778. Isso significa que o ajuste deverá ser razoavelmente bom. A imagem evidencia a qualidade do ajuste.

elipse.jpg

Figura 2: Ajuste da elipse gerada a partir do MMQ

Referências

[GSM] Wikipedia, *Gram-Schmidt process*,
http://en.wikipedia.org/wiki/Gram-Schmidt