1º Trabalho de Física Computacional (R. Coelho)

Mestrado Integrado em Engenharia Física Tecnológica
Ano lectivo: 2019/20 Semestre: 1
25 de Novembro de 2019 (9h00-12h00)

Instruções 1

- **Verifique** que possui uma pasta de trabalho ***Trab01*** no directório **do grupo a que pertence** no servidor SVN com a seguinte estrutura em cada um dos 3 problemas

```
├─── Problema1,2,3
│   ├─── bin
│   └─── src
```

- **Não** altere o nome dos ficheiros com *código fonte* ou ficheiros que contêm as *figuras* a serem produzidas em cada problema.
- **Todos** os ficheiros com código fonte (*.cpp), header files (*.h) e Makefile necessários em cada problema devem ser submetidos no servidor SVN.
- **Não submeta no SVN ficheiros binários (**.o* ou *.exe*).**

Instruções 2

- A pasta ***/src*** na pasta de cada Problema deve conter ***todas as classes*** (*header* e *source*) necessárias à resolução do Problema. Inclui quer classes ***já desenvolvidas*** ao longo do semestre quer classes ***novas*** pedidas no Problema.
- A pasta ***/bin*** na pasta de cada Problema serve para conter os ficheiros binários (****.o*** ou ****.exe***) gerados ao executar a Makefile. *Não submeta este ficheiros binários no SVN* !
- Na ***raíz*** da pasta de cada Problema devem estar a *Makefile*, o programa *main* e a(s) Figura(s) pedidas para esse Problema.

PROBLEMA 1 (9valores)

Neste problema pretende-se desenvolver uma classe ***DFT*** para processamento de sinais digitais, definidos como objectos da classe ***Signal***. Nos ficheiros **Signal.h** e **DFT.h** encontramos a declaração dos *data members* e de alguns dos *métodos* de cada uma destas classes.

De acordo com as definições destas duas classes, a classe *Signal* servirá para guardar um sinal $s(t_i)$ (data member ***signal***) digitalizado com uma certa frequência de aquisição (***constante***) e a respectiva base de tempo (data member ***time***) i.e. a sequência de instantes de tempo t_i onde o sinal está definido. Para que a classe seja útil, terá, entre outros, métodos:

- Plot – Fazer o plot temporal do sinal com título fornecido como input.
- Operações aritméticas entre 2 sinais com a mesma base de tempo (mesmo comprimento/instantes de tempo)
  - operator+, operator-
  - operator* (multiplica 2 sinais s1(t) e s2(t) com $s(t_i)= s1(t_i)* s2(t_i)$
  - operator* (multiplica um sinal s1(t) por uma constante **C** i.e. $s(t_i)= s1(t_i)* C$
- operator[] – retorna uma sample do sinal e o instante de tempo correspondente.
- GetTime() e GetValue() – obter a base de tempo e o sinal.
- Sampling_freq() – retorna a **frequência de aquisição** - ***fs*** - do sinal (inverso da separação temporal entre samples consecutivas do sinal), constante ao longo do tempo.
- Construtor que aceite *vectors* com base de tempo e sinal.
- Construtor que aceite *pointers* com base de tempo e sinal.

- Construtor que aceite um nome de ficheiro com duas colunas (1ª para sequência dos instantes de tempo, 2ª para sequência dos valores do sinal).

A classe DFT, por seu lado, servirá para processar o sinal $s(t_i)$ com i=1..N, obtendo informação sobre a potência espectral associada a uma **determinada frequência f_j (em Hz)** definida entre 0 e fs/2. Esta potência espectral é definida através da relação:

$$P_j = \left(\frac{1}{N}\sum_{i=1}^{N} s_i \times \cos(2\pi f_j t_i)\right)^2 + \left(\frac{1}{N}\sum_{i=1}^{N} s_i \times \sin(2\pi f_j t_i)\right)^2 \qquad (1)$$

Para tal, vamos definir na classe DFT, entre outros, os seguintes métodos:

- *DFT(Signal & sig)*
- *Vector<double> MultCOS(Signal & sig, double & f)*– multiplica o sinal por cos(ωt) onde ω=2πf, i.e. $result_i = s_i \times \cos(2\pi f_j t_i)$ com i=1,2,...,N
- *Vector<double> MultSIN(Signal & sig, double & f)* – multiplica o sinal por sin(ωt) onde ω=2πf, i.e. $result_i = s_i \times \sin(2\pi f_j t_i)$ com i=1,2,...,N
- *double Sum(vector<double>)* – somar todas as componentes do vector em argumento.
- *Double GetPowSpec(double f)* – retorna a potencia espectral para uma frequência f (em Hz)
- *Double GetAmpSpec(double f)* – retorna a amplitude espectral para uma frequência f (em Hz). A relação entre potência e amplitude é dada por $P_j = A_j^2$

Objectivos

1. Implementar a classe ***Signal***.
2. Teste a classe ***Signal*** e os seus métodos tal como indicado no programa *main* ***Signal_and_DFT.cpp*** definido abaixo. Complete as instruções necessárias para executar o que é pedido no programa ***Signal_and_DFT.cpp.***
3. Produza a figura do sinal pedido no main. O gráfico terá como título, fornecido pelo utilizador, ***MySignal*** e o ficheiro produzido terá como nome ***MySignal.eps***.
4. Teste a classe DFT tal como indicado no programa ***Signal_and_DFT.cpp (incompleto).***
5. Produza agora a figura ***MySpectra.eps*** que representa a amplitude espectral do sinal.
6. Criar o executável do programa através da instrução:

```
make bin/Signal_and_DFT.exe
```

PROBLEMA 2 (8valores)

Neste problema pretende-se estudar a dinâmica de um objecto de massa **m** Kg que se desloca a uma dimensão entre 2 paredes localizadas em x=0 e x=L tal como indicado na Figura 1.

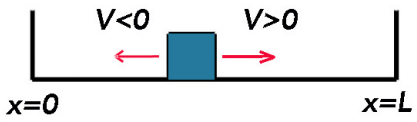

Figura 1 – Dinâmica do objecto de massa-m entre 2 paredes x=0 e x=L com indicação de sentido de velocidade positiva e negativa

Sejam as equações que descrevem a velocidade e a posição do objecto dadas por

$$|v_{n+1}| = \alpha |v_n| \quad \text{e} \quad x_{n+1} = x_n + v_n \delta t \qquad (2)$$

Sempre que o objecto, assumido pontual (dimensão nula), toca numa parede a direção do seu movimento é invertida, conservando a quantidade de movimento. Isto significa que, tendo em conta a discretização temporal (δt controlado arbitrariamente pelo utilizador), caso numa iteração resulte que $x_{n+1}>L$ ou $x_{n+1}<0$, a nova posição deverá ter em conta a reflexão sofrida.

Para resolvermos este problema, vamos criar uma Classe ***InBox1D*** que vai descrever o movimento do objecto entre as duas paredes. Para tal, precisamos dos seguintes métodos, tal como indicados no header file **InBox1D.h** disponibilizado.

- Construtor que aceite *a posição* e *velocidade* iniciais (t=0), posição da parede direita (x=L) e constante de amortecimento da velocidade α.
- AdvanceX – avança a posição X do objecto no tempo (δt segundos) de acordo com a Equação (2) indicada acima.
- AdvanceV – avança a velocidade V do objecto (valor do sinal de acordo com sentido do movimento) no tempo (δt segundos) de acordo com a Equação (2) indicada acima.
- GetX – retorna a posição actual do objecto.
- GetV – retorna a velocidade (incluindo sinal) actual do objecto.

Objectivos

1. Implementar a classe ***InBox1D.***
2. Calcular a evolução temporal (programa main ***Trajectory.cpp***) assumindo os parâmetros da seguinte simulação

   X(t=0)=5.0m, V(t=0)=10.0m/s, L=10m, α=0.99, δt=0.05s, tempo total = 10s

3. Produzir um ficheiro ***Trajectory.eps*** com o gráfico da evolução temporal: X(t) e V(t) cada um em seu sistema de eixos.
4. Nas mesmas condições da simulação pedida em 2., com excepção do tempo total, determine quantas vezes o objecto irá colidir com as paredes ?
   a. Justifique com uma nova figura ***Trajectory_limit.eps*** que permita corroborar a sua conclusão.
   b. Escreva num ficheiro ***README_limit*** que alterações são necessárias no programa *main* e **submeta-o no SVN, na pasta deste Problema**.
5. Criar o executável do programa através da instrução:

```
make bin/Trajectory.exe
```

PROBLEMA 3 (3valores)

Resolva o sistema de equações lineares

a) É possível resolver por eliminação de Gauss sem pivotagem ? Indique-o no programa *main* como comentário.
b) Determine a solução por Eliminação de Gauss ou decomposição LU.

$$\begin{bmatrix} 4 & -2 & 2 & 2 \\ -2 & 1 & 3 & 1 \\ 2 & -2 & 2 & 1 \\ 2 & -1 & 2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \\ 0 \\ 2 \end{bmatrix}$$

Elabore para o efeito um programa main chamado ***Matrix_solver.cpp*** e que utilize as classe **EqSolver** (e todas as suas dependências) que desenvolveu. ***Todas*** estas classes devem estar na pasta ***/src*** deste Problema. O executável do programa através da instrução:

```
make bin/SolveSystem.exe
```