MEFT - Programação
1º Ano - 1º Semestre de 2020/2021
Série 6 (14/12/2020)

1. Oscilador Harmónico Amortecido

a) Construa um programa que calcula numericamente, usando o método de Euler-Cromer, a solução da equação diferencial:

$$m\,\frac{d^2x}{dt^2} + b\,\frac{dx}{dt} + K\,x = 0$$

As condições iniciais (x_o e v_o), bem como as constantes (m, K e b) e os instantes t_0 (inicial) e t_1 (final) deverão ser dados, na linha de comando, pelo utilizador.

Escreva os resultados obtidos num ficheiro a duas colunas.

Utilizando o 'gnuplot' ou outro programa que permita a visualização gráfica, mostre o gráfico com os resultado obtidos.

b) Com vista à análise da qualidade dos resultados obtidos acrescente uma nova coluna ao ficheiro dos resultados em que coloca os resultados obtidos com a solução exacta deste movimento.

Deverá, para tal, obter os valores das constantes de integração em função da posição e velocidade (x_o e v_o).

Nota: Designando por

$$\omega_o^2 = \frac{K}{m} \qquad \text{e} \qquad \lambda = \frac{b}{2\ m}$$

a equação acima indicada tem três tipos de soluções diferentes de acordo com a relação entre ω_o e λ:

1. $\omega_o > \lambda$ (sub-amortecido, frequentemente designado apenas por amortecido):

$$x(t) = A\ e^{-\lambda t}\ cos(\omega\ t + \phi) \qquad \text{com} \qquad \omega = \sqrt{\omega_o^2\ -\ \lambda^2}$$

em que as constantes A e ϕ se obtêm a partir das condições iniciais.

2. Caso de $\omega_o < \lambda$ (amortecimento super-crítico):

$$x(t) = e^{-\lambda t}\ (C\ e^{\beta t} + D\ e^{-\beta t}) \qquad \text{com} \qquad \beta = \sqrt{\lambda^2\ -\ \omega_o^2}$$

em que as constantes C e D se obtêm a partir das condições iniciais.

3. Caso de $\omega_o = \lambda$ (amortecimento crítico):

$$x(t) = e^{-\lambda t}\ (C + D\ t)$$

em que as constantes C e D se obtêm a partir das condições iniciais.

2. Construa um programa em GTK+ que cria uma janela de 500x200 e a coloca no centro do ecrã. Esse programa deve receber um número real como argumento e apresentá-lo na janela por ele criada. Pretende-se, em seguida, que o altere de acordo com as ordens dadas pelo utilizador. Para tal, a janela deve ter quatro botões para executar as seguintes tarefas: somar um, subtrair um, multiplicar por dois e dividir por 3.
Para construir este programa, deverá:
1. Criar uma janela ('window'), dar-lhe as dimensões indicadas e colocá-la no centro do ecrã;
2. Associar ao botão 'destroy' o fecho do programa;
3. Criar uma 'box' vertical e inseri-la na janela;
4. Criar um 'label', em que se irá escrever o valor do número, e colocá-lo na 'box' vertical. Para simplificar, pode definir as variáveis associadas ao 'label' e ao número a mostrar como variáveis globais;
5. Criar uma 'box' horizontal e colocá-la no final da 'box' vertical;
6. Criar os quatro botões e colocá-los na 'box' horizontal criada em '5.';
7. Associar a cada um dos botões, quando se carrega ('clicked'), uma função (callback) para executar a referida tarefa;
8. Dar ordem para mostrar as 'widget's criadas;
9. Entrar no ciclo de espera para execução das tarefas.

3. Construa um programa em GTK+ que cria uma janela com um tamanho à sua escolha. Esse programa deve criar uma 'drawing_area' na qual deverá desenhar um círculo. Para dar a ilusão de movimento deve ainda associar ao programa uma função que será executada de 'x' em 'x' milissegundo usando um 'timeout'. Essa função deverá mandar redesenhar 'drawing_area' e, a cada vez que o faça as coordenadas do círculo deverão ser ligeiramente alteradas.

Nota: Para compilar um programa com GTK+ 3, fazer (exemplo com 'prog.c'):

```
gcc `pkg-config --cflags gtk+-3.0` -c prog.c
gcc -o prog prog.o `pkg-config --libs gtk+-3.0` -lm
```

ou num só comando:

```
gcc -o prog prog.c `pkg-config --cflags --libs gtk+-3.0`
```

Oscilador Amortecido – Cálculo das constantes de integração

1.1 $\omega_o > \lambda$ (sub-amortecido, frequentemente designado apenas por amortecido):

A partir da solução, tem-se para a posição e a velocidade:

$$x(t) = A\ e^{-\lambda t}\ cos(\omega\ t + \phi)$$
$$v(t) = -\lambda\ A\ e^{-\lambda t}\ cos(\omega\ t + \phi) - \omega\ A\ e^{-\lambda t}\ sen(\omega\ t + \phi)$$

fazendo $t = 0$, tem-se:

$$x_o = A\ cos\,\phi$$
$$v_o = -\lambda\ A\ cos\,\phi - \omega\ A\ sen\,\phi$$

dividindo ordenadamente a segunda equação pela primeira, tem-se

$$tg\,\phi = -\frac{v_o + \lambda\ x_o}{\omega\ x_o}$$

usando a relação do quadrado da tangente e a equação de x_o

$$tg^2\phi = \frac{sen^2\phi}{cos^2\phi} = \frac{1 - cos^2\phi}{cos^2\phi} = \frac{1}{cos^2\phi} - 1 \qquad \Rightarrow \qquad \left(\frac{v_o + \lambda\ x_o}{\omega\ x_o}\right)^2 = \left(\frac{A}{x_o}\right)^2 - 1$$

de onde resulta:

$$A = \sqrt{x_o^2 + \frac{v_o^2}{\omega^2} + \frac{\lambda^2\ x_o^2 + 2\ \lambda\ x_o\ v_o}{\omega^2}}$$
$$tg\,\phi = -\frac{v_o + \lambda\ x_o}{\omega\ x_o}$$

em que, das duas soluções possíveis da tangente, se escolhe aquela que assegura que a amplitude é positiva nas equações de x_o e v_o.

1.2 Caso de $\omega_o < \lambda$ (amortecimento super-crítico):

A partir da solução, tem-se para a posição e a velocidade:

$$x(t) = C\ e^{-(\lambda - \beta)t} + D\ e^{-(\lambda + \beta)t}$$
$$v(t) = -(\lambda - \beta)\ C\ e^{-(\lambda - \beta)t} - (\lambda + \beta) D\ e^{-(\lambda + \beta)t}$$

fazendo $t = 0$, tem-se:

$$x_o = C + D$$
$$v_o = -(\lambda - \beta)\ C - (\lambda + \beta) D$$

de onde resulta:

$$C = \frac{v_o + (\lambda + \beta)\ x_o}{2\ \beta} \qquad \text{e} \qquad D = -\frac{v_o + (\lambda - \beta)\ x_o}{2\ \beta}$$

1.3 Caso de $\omega_o = \lambda$ (amortecimento crítico):

A partir da solução, tem-se para a posição e a velocidade:

$$x(t) = e^{-\lambda t}\ (C + D\ t)$$
$$v(t) = -\lambda\ e^{-\lambda t}\ (C + D\ t) + e^{-\lambda t}\ D$$

fazendo $t = 0$, tem-se:

$$x_o = C$$
$$v_o = -\lambda\ C + D$$

de onde resulta:

$$C = x_o \qquad \text{e} \qquad D = v_o + \lambda\ x_o$$

Algumas funções básicas de GTK+ 3

- void **gtk_init** (int *argc, char ***argv);
  Inicialização do ambiente GTK+;

- void **gtk_main** (void);
  Executa o ciclo até ser dada a ordem de o terminar (ver gtk_main_quit());

- void **gtk_main_quit** (void);
  Termina um ciclo 'gtk_main';

- GtkWidget* **gtk_window_new** (GtkWindowType type);
  Cria uma janela. 'type' deve ser 'GTK_WINDOW_TOPLEVEL' para uma janela com decorações e 'GTK_WINDOW_POPUP' para um sem elas;

- void **gtk_window_set_default_size** (GtkWindow *window,
  gint width, gint height);
  Define o tamanho da janela;

- void **gtk_window_set_title** (GtkWindow *window, const gchar *title);
  Define o título da janela;

- void **gtk_window_set_position** (GtkWindow *window,
  GtkWindowPosition position);
  Define a posição em que a janela é colocada. No caso se querer a janela centrada, 'position' deve ser GTK_WIN_POS_CENTER;

- void **gtk_box_pack_start** (GtkBox *box, GtkWidget *child,
  gboolean expand, gboolean fill, guint padding);
  Acrescenta 'child' a uma 'box' empacotando-a a partir do princípio;

- void **gtk_box_pack_end** (GtkBox *box, GtkWidget *child,
  gboolean expand, gboolean fill, guint padding);
  Acrescenta 'child' a uma 'box' empacotando-a a partir do fim;

- void **gtk_container_add** (GtkContainer *container,
  GtkWidget *widget);
  Adiciona uma 'widget' a um 'container';

- void **gtk_container_set_border_width** (GtkContainer *container,
  guint border_width);
  Define a largura do 'border' de um 'container';

- GtkWidget* **gtk_label_new** (const gchar *str);
  Cria um 'label' com o texto 'str';

- void **gtk_label_set_text** (GtkLabel *label, const gchar *str);
  Altera o texto do 'label' para 'str';

- GtkWidget* **gtk_button_new_with_label** (const gchar *label);
  Cria um botão com o texto 'label';

- void **gtk_widget_set_size_request** (GtkWidget *widget,
  gint width, gint height);
  Define o tamanho de uma 'widget';

- void **gtk_widget_show_all** (GtkWidget *widget);
  Torna visível a 'widget' e todas as 'widget's nela colocadas;

- #define **g_signal_connect** (instance, detailed_signal, c_handler, data);
  Liga a função 'c_handler' ao sinal 'detailed_signal' do objecto 'instance' e envia 'data' para essa função;

- GtkWidget* **gtk_box_new** (GtkOrientation orientation, gint spacing);
  Cria uma 'box' com orientação dada por 'orientation' que toma os valores: 'GTK_ORIENTATION_HORIZONTAL' ou 'GTK_ORIENTATION_VERTICAL';

- GtkWidget* **gtk_box_set_homogeneous** (GtkBox *box,
  gboolean homogeneous);
  Indica a homogeneidade da 'box';

- GdkWindow* **gtk_widget_get_window** (GtkWidget *widget);
  Retorna a window no caso de esta esteja realizada;

- gboolean **GTK_IS_WIDGET** (GtkWidget *widget);
  Testa se 'widget' é ou não uma widget;

- GtkWidget* **gtk_drawing_area_new** (void);
  Cria uma 'drawing_area'. A função associa ao sinal 'draw' deve ser tipo:
  gboolean *nome* (GtkWidget *widget, cairo_t *cr, gpointer data)

- void **gtk_widget_queue_draw** (GtkWidget *widget);
  Serve para redesenhar 'widget'.

- gint **g_timeout_add** (guint interval, GSourceFunc function,
  gpointer data);
  O primeiro argumento é o tempo em milissegundos entre chamadas da função 'function' (segundo argumento) e o terceiro argumento é um ponteiro para os dados enviados para a função;

- gint **gtk_widget_get_allocation** (GtkWidget *widget, GtkAllocation *allocation);
  Dá as dimensões da 'widget' na estrutura 'GtkAllocation';

- gint **gtk_widget_get_allocation_width** (GtkWidget *widget);
  Retorna a largura da 'widget';

- gint **gtk_widget_get_allocation_height** (GtkWidget *widget);
  Retorna a altura da 'widget';

Funções de Cairo:

- void **cairo_move_to** (cairo_t *cr, double x, double y);
  Coloca-se na posição (x, y);

- void **cairo_line_to** (cairo_t *cr, double x, double y);
  Traça uma linha recta desde o ponto en que se encontra até (x, y);

- void **cairo_arc** (cairo_t *cr, double x, double y, double r,
  double angle1, double angle2);
  Traça um arco de circuferência centrado em (x, y), de raio 'r' entre os ângulos 'angle1' e 'angle2';

- void **cairo_rectangle** (cairo_t *cr, double x, double y,
  double width, double height);
  Traça um rectângulo a partir do ponto (x, y) com comprimento 'width' e altura 'height';

- void **cairo_set_line_width** (cairo_t *cr, double width);
  Define a largura, em pixeis, das linhas (o default é '2.');

- void **cairo_fill** (cairo_t *cr);
  Faz as superfícies a cheio;

- void **cairo_set_source_rgb** (cairo_t *cr, double r, double g, double b);
  Define a cor em RGB com (r, g, b);

- void **cairo_stroke** (cairo_t *cr);
  Atribui aos objectos as características actuais;

- void **cairo_stroke_preserve** (cairo_t *cr);
  Atribui aos objectos as características actuais e, ao contrário de 'cairo_stroke', preserva a sequência do cairo contexto;

- void **cairo_select_font_face** (cairo_t *cr, const char *family,
  cairo_font_slant_t slant, cairo_font_weight_t weight); Fixa a família de fontes a usar 'family' com a inclinação 'slant' e a largura 'weight'.

- void **cairo_show_text** (cairo_t *cr, const char *text);
  Escreve na posição actual o texto 'text', 'text' deve estar multibyte 'utf8'.