Séries de Problemas de Física Computacional

Pedro Bicudo, Marco Cardoso e Nuno Cardoso , IST,

19 de Setembro de 2017

Resumo

Estas séries de problemas são realizadas nas aulas práticas (dedicadas à vertente mais computacional) e nas aulas de laboratório (dedicadas à vertente mais física) pelos os alunos de Física Computacional do MEFT no ano letivo de 2017/18.

O objetivo dessas aulas é os alunos treinarem a programar e a realizar algoritmos em C++ bem como aprenderem a resolver numericamente algumas das equações recorrentes da física.

Os problemas estão agrupados por séries, correspondendo às diferentes partes da matéria. Em cada série os problemas oferecem uma complexidade crescente.

As séries não devem ser entregues pelos alunos, mas servem de base aos 4 projetos + 1 mini-teste que os alunos irão realizar.

Para prepararem as aulas práticas e de laboratório, os alunos devem apoiar-se nos slides das aulas teóricas e na bibliografia, ambos disponibilizados na página da disciplina, `https://fenix.tecnico.ulisboa.pt/disciplinas/FC5/2017-2018/1-semestre/pagina-inicial` e podem ainda consultar a internet (neste caso verificando com cuidado a consistência da informação) para realizarem os seus códigos.

I Mapas do intervalo

Partindo de um código em C++ fornecido neste enunciado, pretende-se estudar aplicações, ou mapas, do intervalo que produzam caos e valores aleatórios.

Este trabalho é o pretexto, com cálculos extremamente simples que utilizem poucas linhas do código, para que os alunos treinem o input e o output do C++, de ou para, o teclado e o ecrã em I.2) e I.3), ficheiros e tabelas em I.4) e gráficos em I.6), I.7) e I.8). A precisão dos resultados obtidos é crucial, em particular nos exercícios I.6), I.7) e I.8).

Para sua própria organização deve designar os códigos que for construindo por g#s#c#.cpp sendo o primeiro # o número do seu grupo, o segundo # o número da série e o terceiro # o número do código (ex: g13s1c2.cpp será o segundo código que o grupo 13 realiza para a série 1).

I.1

Deve começar por rever todas as aulas teóricas anteriores, copiar os códigos apresentados para ficheiros .cpp e testar a sua compilação e a sua execução. Depois deverá compara-los com os conceitos aqui referidos.

Relate detalhadamente como decorreu a instalação nos dois laptops do seu grupo da distribuição do linux, do compilador gcc e do software Mathematica para a realização deste trabalho computacional. Seguindo estas descrições deverá ser possível re-instalar as distribuições. Caso a instalação seja demasiado difícil e opte por desistir dela, deve documentar detalhadamente os motivos que o levaram a tal.

I.2

Copie para um editor o código com erros (que designará por g#s1c0.cpp)

```
#include <iostream>
int main(){ int n_max=10;
float r=.7, x=.5; // r eh o parametro de controle
while( int n < n_max ) { /* ciclo das iteradas */  x *= a;
std::cout << "x(" << ++n << ")= " << x << std::endl; }
return 0; }
```

corrija-o (e indente-o) para uma nova versão g#s1c1.cpp, até ele compilar e correr bem. Refira as alterações que teve de fazer ao código para ele funcionar.

I.3

Melhore o input e o output do código, agora designado g#s1c2.cpp para ele ler a partir do teclado, todos os dados a utilizar (número de iteradas, fator r e variável x inicial). Altere também o mapa do intervalo, passando da sucessão geométrica de Malthus para a sucessão logística, ou mapa logístico, conforme o "website"de FC, ou (neste caso verificando com cuidado a consistência da informação) no "site"
`http://en.wikipedia.org/wiki/Logistic_map` .
Comece por demonstrar analiticamente que, para $0 < r < 4$, a sucessão é sempre inferior a 1. Sugestão: mostre que, se num dado passo $0 < x_n < 1$, então o valor seguinte de x também obedece a que $0 < x_{n+1} < 1$. Verifique então numericamente se surgem bifurcações, nos valores críticos do fator r tal como se descreve no referido site.

I.4

Desenvolva o código, designado agora g#s1c3.cpp, para que execute as mesmas operações que o código anterior, mas agora leia os dados n, r e x dum ficheiro e escreva os resultados

noutro ficheiro. Assim treina o input e o output, de e para ficheiros.

I.5

Deve ainda, para dois casos, primeiro para um valor de r onde claramente 1 limite apenas existe e depois para outro valor de r onde claramente existam 4 pontos de acumulação da sucessão x_n, estudar a convergência da sucessão. Em particular deve estimar numericamente, utilizando o código g#s1c3.cpp e observando o respetivo ficheiro de output, quantas iteradas são necessárias para determinar o valor desses pontos de acumulação até à 6^a casa decimal.

I.6

Um novo código, designado g#s1c4.cpp, é agora escrito com o fim de construir um plot semelhante ao da figura das bifurcações do referido "site"do mapa logístico. Neste caso o código g#s1c4.cpp deve incluir um novo ciclo, para varrer o parâmetro r. Pode escrever simplesmente os resultados todos num ficheiro que em seguida deve plotar utilizando o Gnuplot (ou o4 GTK utilizado em Programação). O ficheiro consistirá num número elevado de pares (r, x_i). Sugestão: para cada valor de r corra inicialmente um número de iteradas elevados, igual ou superior ao determinado em I.5), ao fim do qual pode guardar um número grande de valores x_n da sucessão, para representar os x_n relevantes. O gráfico deverá ter a melhor qualidade possível. Comente o gráfico obtido, evidenciando o aparecimento do caos, conforme o "website"de FC, ou (neste caso verificando com cuidado a consistência da informação) no ”site”,

`\url{http://en.wikipedia.org/wiki/Chaos_theory}`

I.7

Já que o mapa logístico cria caos, pretendemos usá-lo agora para construir um pseudo gerador de números aleatórios (PRNG em inglês), conforme o "website"de FC, ou (neste caso verificando com cuidado a consistência da informação) no ”site”,
`http://en.wikipedia.org/wiki/Random_number_generator`
Vamos começar por testar se o mapa logístico, para um valor de $r < 4$ mas tão próximo de 4 quanto possível, gera números aleatórios de uma forma satisfatória no intervalo $0 < x < 1$. Para verificar a qualidade do gerador, considere um valor de r fixo e muito próximo de 4. Tal como em 3) grave então num ficheiro valores a partir de um i suficientemente elevado. A diferença em relação a 3) é que vai criar um código g#s1c5.cpp para gravar no ficheiro uma sucessão de um número elevado de pares de números (x_i, x_{i+1}). Faça um plot do conjunto de pares assim obtidos no plano utilizando o Gnuplot (ou o GTK utilizado em Programação). Observe o padrão de pontos assim criado e comente se o gerador assim obtido terá boa qualidade.

I.8

Repita o realizado em I.7) para uma nova sucessão, simulada pelo código g#s1c6.cpp. Trata-se de um mapa do intervalo usado para um PRNG simples de programar, com um período elevado. Trata-se de um gerador linear congruencial, conforme o "website"de FC, ou (neste caso verificando com cuidado a consistência da informação) no ”site”
`http://en.wikipedia.org/wiki/Linear_congruential_generator` .
Considere o primeiro gerador definido na tabela do ”site”, com período $2^{32} - 1$. O números a guardar no ficheiro devem estar normalizados, ou seja não deve guardar (x_i, x_{i+1}) mas sim deve guardar $(x_i/2^{32}, x_{i+1}/2^{32})$. Faça um plot utilizando o *Mathemática* (ou o

GTK utilizado em Programação). Compare a qualidade aparente deste gerador com a do exercício I.8).

II Matrizes e integrais

Partindo dos código em C++ da aula teórica e disponíveis na página da cadeira, pretende-se trabalhar com matrizes e calcular integrais, primeiro apenas como treino computacional, e depois num caso relevante para a física. Este trabalho é o pretexto para que os alunos treinem o trabalho com ponteiros, funções e namespaces em C++. A precisão dos resultados obtidos é crucial, em particular nos exercícios II.6) e II.7).

II.1

Deve começar por rever todas as aulas teóricas anteriores, copiar os códigos apresentados para ficheiros .cpp e testar a sua compilação e a sua execução. Depois deverá compara-los com os conceitos aqui referidos.

Explique - matéria da aula teórica 3 - de que diferentes formas (arrays, ponteiros) se pode definir vectores e matrizes em C++, bem como pode aplicar ponteiros para trabalhar com funções em memória dinâmica, referindo as respectivas vantagens e inconvenientes.

Escreva um código que designe por g#s2c1.cpp e que trabalhe com matrizes na memória dinâmica usando ponteiros. Em particular desenvolva um código simples para discretizar uma função matemática $f(x)$ (a que preferir) numa matriz de duas colunas, uma para os valores de x_i discretizados (como preferir) e outra para os respectivos valores da função $f_i = f(x_i)$. O código deve ainda escrever a matrix num ficheiro.

II.2

Explique também -matéria da aula teórica 5 - em que casos é vantajoso o uso do namespace. Escreva um código designado por g#s2c1_v2.cpp (que melhore o código g#s2c0.cpp que pode ver por exemplo no website da disciplina FC) com dois namespaces novos, um por cada aluno do grupo. Os alunos podem inventar o objectivo do código, mas cada namespace deve definir funções diferentes, que serão depois utilizadas no main.

II.3

Agora pretende-se que utilize os ponteiros para calcular integrais a uma dimensão numericamente. Para estes integrais deve utilizar variáveis dinâmicas bem como funções dinâmicas. Deve comparar 3 técnicas numéricas clássicas: do rectângulo, do trapézio e de Simpson. Pode revê-los conforme o "website"de FC, ou (neste caso verificando com cuidado a consistência da informação) nos ”sites”,
`http://en.wikipedia.org/wiki/Numerical_integration` ,
`http://en.wikipedia.org/wiki/Trapezium_rule` ,
`http://en.wikipedia.org/wiki/Simpson%27s_rule` ,
Em C++, cada integral é uma função, que tem com argumentos: - os extremos do intervalo $[a, b]$ e o número n de sub-intervalos de soma, - a função $f(x)$ que pretendemos integrar (utilize um ponteiro para uma função), - a função que calcula o integral numérico, e que retorna: - o valor do integral numérico, sendo este imprimido no écran e num ficheiro. Teste os três códigos g#s2c2rect.cpp, g#s2c2trap.cpp e g#s2c2simp.cpp no caso de integrais polinomiais cujo valor analítico conheça.

II.4

Pretendemos agrora calcular numericamente $\int_{\infty}^{+\infty} x^4 \phi(x)^2 dx$, onde $\phi(x) = \mathcal{N} * \exp(-x^2/(2\alpha^2))$ é escolhida por ser um exemplo de função de onda quântica, sendo α constante, e $\mathcal{N}$ uma normalização. A normalização $\mathcal{N}$ é determinada pelo integral $\int_{\infty}^{+\infty} \phi(x)^2 dx = 1$.

Comece por calcular esse integral analiticamente e determine então $\mathcal{N}$. Calcule também o integral de $x^4\,\phi(x)^2$ analiticamente. No que se segue considere $\alpha = 1$.

II.5

Calcule então numericamente $\int_{\infty}^{+\infty} x^4 \phi(x)^2 dx$,
- onde se substitui os limites $\pm\infty$ por valores finitos $a = -b, b$, sendo b um número grande à sua escolha,
- e arbitrando um dado número n de sub-intervalos de soma.
O código da função main será denominado g#s2c3.cpp, sendo códigos de II.3) usados como funções (no sentido C++) usadas pela main, podendo opcionalmente o código usar uma das 3 técnicas de II.3). Compare então a precisão dos três métodos numéricos. Avalie, para a regra de Simpson, que valor de b e que número de pontos n são necessários para acertar a 4a casa decimal, de acordo com o resultado analítico de II.4).

II.6

Finalmente irá desenvolver ainda uma nova técnica numérica. Esta técnica, de Metropolis Monte Carlo, que utiliza uma cadeia aleatória de Markov, aplica-se quando queremos representar uma distribuição de probabilidade (contínua) por um conjunto discreto de pontos. É poderosíssima para integrais com muitíssimas dimensões como na distribuição de Maxwell-Boltzmann da física estatística. Veja a aulas teórica na página da dsiciplina, podendo consultar também,
`http://xbeams.chem.yale.edu/~batista/vaa/node42.html` ,
`http://en.wikipedia.org/wiki/Metropolis-Hastings_algorithm` ,
estude como podemos substituir o integral de $f(x)\rho(x)$, sendo uma distribuição de probabilidade, por $\sum_{i=0}^{N-1} \frac{f(x_i)}{N}$ e constituindo os x_i um conjunto finito de N pontos que caracteriza o sistema.

Pretende-se então construir duas funções,
- uma função para gerar o conjunto dos pontos x_i (use um ponteiro) que caracterizam a distribuição de probabilidade (que como argumentos tem com a distribuição de densidade $\rho(x)$, o número finito n de pontos x_i, e ainda o número de pontos intermédio m e o passo médio r da cadeia aleatória de Markov) que opcionalmente deve imprimir no écran e num ficheiro a taxa de aceitação,
- uma função para calcular o integral $\sum_i \frac{f(x_i)}{N}$ (que como argumentos tem a função matemática $f(x)$ e ainda com o conjunto dos x_i, determinado na função anterior), que escreve no ecran e num ficheiro a matriz (x_i, f_i) e o integral,
- e ainda um programa main g#s2c4.cpp que lê do teclado o input N, m, r .

Deve finalmente aplicar esta técnica a um integral semelhante ao de II.5) com este método de Metropolis Monte Carlo, sendo que
- a densidade de probabilidade é $\rho(x) = \phi(x)^2$
- a função a integrar é $f(x) = (x^2)^2$.
- mas onde tanto em f como em ρ se substitui $x^2 \to {x_0}^2 + {x_1}^2 + \cdots + {x_{N-1}}^2$.

Comece por considerar o caso mais simples de $D = 1$, e compare, digamos para 1000 pontos de integração, a eficiência deste método de Metropolis com o método Simpson de II.3).

Para testar o seu código deve ainda,
- calcular a taxa de aceitação média (que deve ser da ordem de 0.5)
- fazer plots, por exemplo da nuvem de pontos (x_i, x_{i+1}) (pontos sucessivos não devem ser correlacionados ou seja a nuvem de pontos deve ser simétrica).

II.7

O objetivo agora é o de preparar um novo código para trabalhar com um número de dimensões D à sua escolha.

Primeiro considere $D = 2$. Relativamente à alínea II.6), deve fazer a substituição $x^2 \to x^2 + y^2$ no expoente da distribuição $\rho(x, y)$. Pode fazer a mesma substituição

também na função $f(x, y)$, mas recordamos que esta técnica de integração é válida para qualquer função. Neste caso deve gerar mais número aleatório que para $D = 1$ afim de determinar a direção de cada passo aleatório a dar. Deve gerar um conjunto com, no mínimo 10000 pontos de integração, e representar a nuvem de pontos em plots a D=2. Pode mostrar tanto um plot dos pontos (x_i, y_i) como os plots de (x_i, x_{i+1}) e de (y_i, y_{i+1}).

Em seguida, caso ainda tenha tempo disponível para esta série, pode passar para um valor de $D > 10$.

Neste caso a substituição, relativamente à alínea II.6) é, $x^2 \to \sum_{a=0}^{D-1} {x_a}^2$, sendo o a o índice da componente (não confundir com o índice da iteração i). A direção do passo aleatório também deve ser aleatória no espaço a D dimensões.

Depois calcule o integral numericamente, com cerca de 10000 pontos de integração. Determine a precisão do resultado, ou numericamente, ou calculando o integral analítico (que pode calcular no mathematica) e comparando-o com o valor numérico.

Estime quanto tempo demoraria a calcular este integral com o método de Simpson na mesma precisão que aqui atingiu. Estime a partir de que D este método de Monte Carlo é mais eficiente que o de Simpson.

III Derivadas e equações diferenciais ordinárias

Partindo de conceitos e de códigos em C++, apresentados na aula teórica e presentes na página da cadeira, pretende-se trabalhar com derivadas e resolver equações diferenciais, primeiro apenas como treino computacional, e depois num caso relevante para a física. Este trabalho é o pretexto para que os alunos treinem a matéria das aulas anteriores (migração para C++, arrays, ponteiros, funções e namespaces), e também trabalhem com o estruturas/classes, construtores e destruidores. A precisão dos resultados obtidos pode ser avaliada nos problemas III.3), III.4) e III.5), sendo o exercício III.5) o mais difícil.

III.1

Deve começar por rever todas as aulas teóricas anteriores, copiar os códigos apresentados para ficheiros .cpp e testar a sua compilação e a sua execução. Depois deverá compara-los com os conceitos aqui referidos.

Explique - matéria da aula 7 - em que casos é vantajoso o uso de strings e de estruturas, bem como de construtores e de destruidores. Escreva um código designado por g#s4c1.cpp (que melhore o código g#s3c0.cpp que pode ver por exemplo no website da disciplina FC) com strings e com estruturas para criar uma base de dados. O código deve também definir um novo tipo correspondente a vetores, para incluir grandezas vetoriais na base de dados. O código deve ir além do código das aulas, sendo capaz de ordenar a lista de estruturas por comparação com o respectivo 1º dado.

III.2

Agora pretende-se resolver equações diferenciais. Utilizando funções com ponteiros para funções, crie um código que resolva equações diferenciais de 1a ordem com o método de Euler (ver aula), que pode por exemplo estudar no wikipedia em,
`http://en.wikipedia.org/wiki/Euler_method` ,
que discretiza a derivada com diferenças finitas h .O código, g#s3c3.cpp, deve produzir uma tabela com 3 colunas onde apresenta os valores da variável x, da função $f(x)$, e da derivada $f'(x)$. Aplique este código ao decaimento radioativo, de equação $\frac{dN(t)}{dt} = -\lambda N(t)$, onde agora a função é o número de partículas N e a variável é o tempo t. Verifique a precisão da solução para diferentes passos h, por comparação com a solução analítica, "plotando" gráficos das diversas soluções (numéricas e analítica).

III.3

Estenda este código g#s3c3.cpp para um novo g#s3c4.cpp onde utilize o método de Runge-Kutta de 4a ordem (neste caso não é a ordem da EDO mas sim do erro numérico, ver na aula), que também pode estudar em,
`http://en.wikipedia.org/wiki/Runge-kutta` .
Compare as soluções dos método de Runge-kutta e de Euler para a eq. do decaimento radioativo.

III.4

O objectivo agora é desenvolvermos códigos para resolver uma E.D.O. de 2a ordem. Comece por explicar como uma E. D. O. de 2ª ordem é equivalente a um sistema de 2 E. D. Os. de 1a ordem, se fizermos uma simples mudança de nome da função. Irá agora adaptar os códigos g#s3c3.cpp e g#s3c4.cpp, para dois novos códigos que resolvem equações diferenciais de 2ª ordem, g#s3c5.cpp com o método de Euler e g#s3c6.cpp com o método de Runge-kutta. Escreva ainda um 3o código g#s3c7.cpp com a técnica de Numerov simplificada (ver aula), que é específica para eqs. diferenciais de 2a ordem sem termo de 1a

ordem. Aplique os 3 códigos à equação do oscilador harmónico simples que pode ver em, `http://en.wikipedia.org/wiki/Harmonic_oscillator` .
Compare os 3 códigos "plotando" gráficos das diversas soluções numéricas, para um passo numérico h que não seja demasiado pequeno.

III.5

Irá agora aplicar os métodos numéricos para Eqs. Diferenciais Ordinárias de 2a ordem a novos sistemas. Cada grupo deve apenas trabalhar num caso, correspondente ao seu número de grupo #,

$\# = 0 \pmod 4$: Eq. de Schrödinger estacionária com o "shooting method". Aplique à eq. de Schrödinger estacionária para o potencial do oscilador harmónico dada por, $-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}\phi + \frac{k}{2}x^2\phi = E\phi$, onde pretendemos determinar a função de onda $\phi(x)$ e a energia E. Considere os valores $m = 1, k = 1, \hbar = 1$, resultando numa equação parametricamente simples. Considere as duas diferentes condições fronteira dadas por $\phi(0) = 1$ e $\phi'(0) = 0$ ou por $\phi(0) = 0$ e $\phi'(0) = 1$. Aplique a técnica de Runge-kutta para obter a solução numérica para um dado valor da energia E. Mas, para resolver corretamente a equação de Schrödinger estacionária, deve ainda garantir que a solução se anula a grande distância x_{max} da origem. A solução irá apenas anular-se para alguns valores (ditos quantificados) da energia E que determinamos com o "shooting method"em que fazemos pontaria para anular $\phi(x_{max})$. Pode considerar $x_{max} = 7$. Para encontrar os valores mais baixos de E, utilize a técnica da bissecção, a ver em,
`http://en.wikipedia.org/wiki/Bisection_method` .
Compare os valores da energia assim obtidos com os valores teóricos $E = 1/2, E = 3/2, E = 5/2, E = 7/2 \cdots$, e faça um plot das respetivas soluções $\phi(x)$.

$\# = 1 \pmod 4$: Sistema de 3 corpos gravíticos a $D = 2$. Escreva a 2a Lei de Newton para 3 corpos com massa que se atraem entre si com a Lei da gravitação de Newton. Considere valores para as massas $m = 1$ e a constante $G = 1$ que resultam numa equação parametricamente simples. Aplique o método de Numerov a este sistema de equações. "Plote"no plano, durante um intervalo de tempo que ilustre bem os movimentos, posições sucessivas das partículas. Ilustre numa figura as trajetórias das partículas ao longo de algumas órbitas, e noutra figura ilustre a existência de caos.

$\# = 2 \pmod 4$: Pêndulo duplo a $D = 2$. Escreva a 2a Lei de Newton para o pêndulo duplo que se move apenas num plano vertical, estando os dois corpos sujeitos apenas às seguintes forças: o peso constante e a tensão das barras. Considere os valores para as massas $m = 1$, para a acelaração gravítica $g = 1$ e para o comprimento das barras $l = 1$ que resultam numa equação parametricamente simples. Aplique o método de Runge-Kutta para este sistema de equações. Ilustre numa figura as trajetórias das partículas ao longo de algumas órbitas, e noutra figura ilustre a existência de caos.

$\# = 3 \pmod 4$: Cadeia de 10 osciladores harmónicos a $D = 1$. Escreva a 2a Lei de Newton para uma cadeia linear de 10 osciladores harmónicos uni-dimensionais, constituída por 10 massas alternadas por 11 molas, estando as massas apenas sujeitas à força elástica das molas. Os extremos da 1a e da 11a mola estão fixos. Considere valores simples para as massas $m = 1$ e as constantes das molas $k = 1$. Aplique o método de Numerov para este sistema de equações. Desenhe um plot no plano (t, x), durante um intervalo de tempo que ilustre bem os movimentos, das posições sucessivas $(t, x_i(t))$ das partículas. Tente encontrar as condições fronteira que resultem em soluções semelhantes a ondas longitudinais estacionárias das 10 massas.

Observação: criou uma ferramenta para resolver muitas das eqs. cruciais da física.

IV Equações diferenciais às derivadas parciais

Partindo de códigos em C++, apresentados na aula teórica ou presentes na página da cadeira, pretende-se resolver equações diferenciais às derivadas parciais (EDDP), num caso simples mas relevante para a física. Mas antes convém começarmos por nos treinarmos resolvendo uma equação algébrica. Este trabalho é o pretexto para que os alunos treinem a matéria das aulas anteriores (migração para C++, arrays, ponteiros, funções, namespaces e estruturas), e agora também aprofundem o trabalho com classes, chegando até ao overload de operadores. Na perspectiva da física, a precisão dos resultados obtidos é crucial, sendo avaliada nas alíneas IV-10) e IV.11).

IV.1

Deve começar por rever todas as aulas teóricas anteriores, copiar os códigos apresentados para ficheiros .cpp e testar a sua compilação e a sua execução. Depois deverá compara-los com os conceitos aqui referidos.

Explique - matéria da aula 9 - brevemente em que casos é vantajoso o uso de classes e de referências. Escreva um código designado por g#s4c1.cpp (que melhore o código g#s4c0.cpp que pode ver por exemplo no site de FCom.) com classes que incluam um constructor e um destructor. Estude diferentes tipos de construtores. Exemplifique ainda - matéria da aula 11 - o overload de operadores num caso simples.

Agora pretende-se resolver equações algébricas (o que constitui o primeiro passo para entendermos a resolução de EDDPs). O objectivo é resolvermos a eq. $x = f(x)$, por iteração da sucessão, $x_{i+1} = f(x_i)$, onde se a sucessão convergir para um limite então temos uma solução da eq. algébrica respectiva.

IV.2

Para tal basta utilizar o código do mapa do intervalo da 1a série e adapta-lo para uma equação algébrica, por exemplo para, $f(x) = \cos^2(x)$. Um truque para acelarar a convergência consiste utilizar o método da relaxação, substituindo a equação iterativa por, $x_{i+1} = wf(x_i)+(1-w)x_i$, onde w é o parâmetro da relaxação. Para um determinado valor de w, calcule o número de iteradas necessário para chegar a um erro de $|x_{n+1}-x_n| < 10^{-6}$.

IV.3

Avalie, criando um ficheiro e um plot, com o seu novo código, g#s4c2.cpp, qual é o parâmetro w que assegura a mais rápida convergência da eq. algébrica, $x = \cos^2(x)$. No plot mostre, como abcissa w, e como ordenada o número de iteradas necessário para chegar a um erro de $|x_{n+1}\breve{}x_n| < 10^{-6}$. determine assim o w ideal e o respetivo número de iteradas mínimo.

Agora começamos a aplicar o mesmo método a uma EDDP - matéria da aula teórica 8 - em particular à eq. de Poisson, $\Delta V = -\frac{1}{\epsilon_0}\rho$, e onde, por exemplo a dimensões D= 2, $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ é o laplaceano, $V(x, y)$ é o potencial electroestático que pretendemos determinar, (x, y) são a abcissa e a ordenada e $\rho(x, y)$ é a densidade de carga eléctrica.

IV.4

Estude como discretizar o espaço $x \to x_{min} + h \times i$, passando de variáveis contínuas x para índices discretos i . Utilizando diferenças finitas para a segunda derivada, mostre então que a eq. diferencial fica, - a dimensões D=1, $V_{i+1} + V_{i-1} - 2V_i = -\frac{h^2}{\epsilon_0}\rho_i$, - a

dimensões D=2, $V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,y_j-1} - 4V_{i,j} = -\frac{h^2}{\epsilon_0}\rho_{i,j}$, - e derive ainda a equação correspondente a D=3.

IV.5

Mostre ainda como, mudando de unidades, pode simplificar a equação de forma a ter $\epsilon_0 = 1$.

IV.6

Explique como deverá escrever um código que resolva a eq. iterativamente calculando $V_{i,\ldots}$ em função dos valores do array V nos pontos vizinhos. Deve explicá-lo caso a caso, para as diferentes dimensões $D = 1$, $D = 2$ e $D = 3$.
Nota: resolvemos o problema numa superfície retangular, o utilizador deve escolher os passos de discretização em x e y, bem como o número de pontos, e ainda o potencial V na fronteira; deve ainda poder colocar cargas sobre a superfície.
Nota: para o código correr, tal como na alínea IV.2, necessita de arbitrar (escolher) uma condição inicial para o potencial em todos os pontos do retângulo.

IV.7

Escreva um código simples g#s4c3.cpp que ilustre a resolução da equação iterativamente a $D = 1$ para V_i, aplicando-o a condições fronteira em V_0 e $V_{n_{max}-1}$ à sua escolha. Neste caso pode considerar simplesmente uma densidade de carga nula (eq. de Laplace, $\rho_i = 0$).

IV.8

Para se acelerar a convergência da solução existem diversos métodos numéricos, incluindo os de Jacobi, Gauss-Seidel, SOR, que pode estudar no wikipedia em,
`http://en.wikipedia.org/wiki/Jacobi_method` ,
`http://en.wikipedia.org/wiki/Gauss-Seidel_method` ,
`http://en.wikipedia.org/wiki/Successive_over-relaxation` .
Estude esses métodos e diga qual corresponde ao problema do código g#s4c3.cpp. Sem escrever um novo código, explique como faria para programar com as outras 2 técnicas numéricas.

Finalmente aplicamos a técnica numérica entendida em IV.8) para resolver a eq. de Poisson nas 3 diferentes dimensões $D = 1$, $D = 2$ e $D = 3$.

IV.9

Para resolver estas equações deve, ou escrever um código g#s4c4.cpp com 3 diferentes opções, ou então escrevendo 3 códigos diferentes: g#s4c4a.cpp, g#s4c4b.cpp e g#s4c4c.cpp. O código deve, ou pedir ao utilizador as medidas do domínio a estudar, as condições fronteira e a densidade de carga, ou ler estes dados de um ficheiro. Aplique a um caso à sua escolha e produza plots que ilustrem o bom funcionamento do código.

IV.10

Aplique ainda ao seguinte caso. Considere respetivamente um segmento, um retângulo e um paralelepípedo com comprimentos da ordem de 40 unidades. Deve colocar 2 cargas opostas no interior do domínio a estudar, nos pontos no eixo dos x, para $x = 10$ e $x = -10$. Resolva então a equação a a $D = 1$, $D = 2$ e $D = 3$ e mostre que obteve uma boa convergência.

IV.11

Produza plots do campo V ao longo do eixo dos x (com as outras coordenadas nulas), que ilustrem a diferença entre as soluções a $D = 1$, $D = 2$ e $D = 3$. Compare e comente os resultados.

V Equações diferenciais com valores próprios

Partindo do código em C++, desenvolvido nas aulas e presente na página da cadeira, pretende-se resolver a equação de Schrödinger que não só é uma equação diferencial como também é uma equação aos valores próprios como as da álgebra linear. Este trabalho é o pretexto para que os alunos treinem a matéria das aulas anteriores (migração para C++, arrays, ponteiros, funções, namespaces, strings, estruturas, classes e overload de operadores), e agora também trabalhem com a derivação e templates de classes.

V.1 Inversão de matrizes

Deve começar por rever todas as aulas teóricas anteriores, copiar os códigos apresentados para ficheiros .cpp e testar a sua compilação e a sua execução. Depois deverá compara-los com os conceitos aqui referidos.

Explique a vantagem de usar derivação e funções virtuais - matéria da aula 13 - bem como templates de classes e de funções - matéria da aula 15 - em C++. Aproveite das aulas um código simples que trabalhe com vetores, denominando-o g#s5c0.cpp e adapte-o para trabalhar com templates num código g#s5c1.cpp .

V.2

Partindo do código disponível online com uma classe para vetores `VecAnyD`, crie um código g#s5c2.cpp, incluindo uma classe com overload de operadores (aqui não necessita de usar templates) para trabalhar não só com vetores mas também com matrizes reais quadradas de dimensão NxN. As operações usuais das matrizes, tais como a soma de duas matrizes, o produto de duas matrizes, o produto de um número por uma matriz, e o produto de uma matriz por um vetor devem ficar funcionais. Deve ainda fazer o overload para imprimir no ecran ou num ficheiro, em linhas e colunas, uma matriz.

Sugestão: pode utilizar uma classe para matrizes que já tenha criado, ou a classe `Matrix` que está disponível na solução do projeto 3.

V.3

Mostre com casos simples a N=4 que o código está correto. Os exemplos a estudar devem ser fornecidos num ficheiro de dados matvec.dat, e os resultados devem ser produzidos num ficheiro bem como mostrados no ecran do computador. Deve mostrar exemplos que realmente ilustrem que todas as operações que programou estão corretas.

V.4

Pretendemos agora desenvolver um código g#s5c3a.cpp para calcular a inversa de matrizes. Utilize a técnica da Eliminação de Gauss, com o uso de "pivots" para calcular a inversa da matriz quadrada M, dada na aulta teórica 18. Pode rever esta técnica standard que aprendeu em Algebra nos links (sabendo que a informação presente na internet deve ser sempre verificada com cuidado),

`http://en.wikipedia.org/wiki/Gaussian_elimination`
`https://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/video-lectures/lecture-2-elimination-with-matrices/`
`https://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/video-lectures/lecture-3-multiplication-and-inverse-matrices/`

V.5

A partir do momento que o código anterior funcione, deve então estende-lo para um código g#s5c3b.cpp que ainda, como subprodutos, calcule o determinante $\det(M)$ e resolva diretamente a equação algébrica $M \cdot V = B$ onde B é um vetor conhecido e V é o vetor que pretendemos determinar, $V = M^{-1} \cdot B$,

`http://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/video-lectures/lecture-8-solving-ax-b-row-reduced-form-r/`

V.6

Verifique com matrizes e vetores à sua escolha para $N = 4$ que o código g#s5c3b.cpp está correto. Os exemplos a estudar devem ser fornecidos num ficheiro de dados matvec.dat, e os resultados devem ser produzidos num ficheiro bem como mostrados no ecran do computador. Deve mostrar exemplos que realmente ilustrem que todas as operações que programou estão corretas.

Teste ainda para matrizes com N cada vez maior, o tempo que ocódigo leva a correr em função de N, considerando por exemplo $N = 10, 100, 1000 \cdots$

V.7 Maiores vetores próprios

Pretendemos agora calcular alguns valores e vetores próprios de uma matriz real $N \times N$. Iremos utilizar uma técnica (dada na aula teórica) que prima pela simplicidade numérica, e que converge para o valor próprio mais alto e o seu respetivo vetor próprio.

Seja V_0 um vetor inicial, que devemos escolher com uma boa intuição afim de estar o mais alinhado possível com a solução que pretendemos. Começando com V_0, construimos uma sucessão

• iterando a equação, $V_{a+1} = M \cdot V_a$

• bem como a cada passo calcularmos o ratio para a componente j mais significativa do vetor, $\lambda_{n+1} = V_{a+1}[j]/V_a[j]$

• e ainda fechamos a iteração com a substituição, $V_{a+1} \to V_{a+1}/\sqrt{V_{a+1} \cdot V_{a+1}}$.

Mostra-se que então com esta sucessão V_a tende para o valor próprio e λ_a tende para o valor próprio. Desenvolva um código g#s5c4a.cpp que calcule assim este valor e vetor próprio da M, avaliando a precisão da convergência.

V.8

No caso de termos uma matriz hermítica (no caso duma matriz real deve então ser simétrica), a mesma técnica serve ainda para calcular alguns dos vetores próprios seguintes. Denominemos por ordem crescente os vetores próprios como V_0, $V_1 \cdots V_{N-2}\, V_{N-1}$. Com a sucessão dos V_a já determinámos o vetor próprio V_{N-1}. Calculando a diferença $V_{a+1} - V_a$, que denominamos V'_a e que ainda normalizamos, determinamos V_{N-2}. Da mesma forma calculando a diferença $V'_{a-1} - V'_a$ e normalizando-a determinamos V_{N-3}. A cada diferença perdemos precisão, mas com uma boa precisão para V_{N-1} conseguimos determinar os três maiores vetores próprios, e os respetivos valores próprios. Desenvolve um código g#s5c4b.cpp que determine estes vetores e valores próprios.

Em alternativa pode utilizar o método de Gram-Schmidt (dado na aula teórica) para trabalhar com vetores sucessivamente ortogonais aos maiores vetores próprios anteriores.

V.9

Para verificar se o códigos de V.7) e de V.8) estão corretos, vamos determinar os valores próprios de uma forma alternativa. Com o código sub-produto de V.5) calcule o determi-

nante $det(M - \lambda I)$, sendo I a matriz identidade e λ un número real genérico. Os zeros do determinante, também conhecido como soluções da equação secular da matriz, são os valores próprios da matriz. Para uma matriz simétrica (e invertível) à sua escolha, 10×10, Desenhe um plot do determinante em função de λ e verifique graficamente que as soluções das sucessões de V.7) e de V.8) coincidem com os maiores valores próprios.

V.10 Equação diferencial com vetores próprios

Finalmente pretendemos continuar a resolver a eq. de Schrödinger estacionária para o potencial do oscilador harmónico uni-dimensional, $-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}\phi + \frac{k}{2}x^2\phi = E\phi$, já estudada na aula teórica e numa das opções da série 3, para os valores simples dos parâmetros $m = 1, k = 1, \hbar = 1$. Procuramos a função de onda de estados ligados para diferentes valores da energia E. Se considerarmos que a coordenada x pertence a um intervalo $[-\Lambda, \Lambda]$ discretizado nos pontos $x_i = -\lambda + ih$ sendo h o passo da rede e o número de pontos N com $h = -2\Lambda/(N + 1)$, então a função de onda equivale a um vetor de N componentes $\phi_i = \phi(x_i)$. Da mesma forma escrevemos aproximadamente a segunda derivada com diferenças finitas, $\frac{d^2}{dx^2}\phi_i = \frac{\phi(x_{i+1}) - 2\phi(x_i) + \phi(x_{i-1})}{h^2}$.

Assim mostre que o operador 2a derivada equivale a uma matriz que atua no vetor ϕ_i, ou seja escreva $-\frac{1}{2}\frac{d^2}{dx^2}\phi_i$ numa forma matricial $K \cdot \phi$. Mostre também que a eq. de Schrödinger é uma equação matricial aos valores próprios, explicando quais são o vetor, a matriz, e o vetor próprio. Nota, a matriz é designada $H = K + V$ pois é denominada Hamiltoniano. Mostre ainda que a matriz H é real e simétrica (ou seja é hermítica). Isto implica que os vetores próprios da eq. de Schrödinger estacionária são ortogonais. Exemplifique a matriz H considerando o caso particular de $N = 5$, com as condições fronteira $\phi(3h) = \phi(-3h) = 0$.

V.11

Para resolver a eq. de Schrödinger basta resolver a eq. aos valores próprios de H. Os valores próprios que nos interessam são os mais baixos. Para usarmos a técnica desenvolvida em V.10) começamos por inverter a matriz H, com o código desenvolvido em V.5). Em seguida aplicamos a esta matriz $M = H^{-1}$ o código desenvolvido em V.7) e V.8) afim de determinarmos os 3 maiores valores próprios e respetivos vetores próprios. Estes vetores próprios V_0, V_1, V_2 coincidem com os vetores próprios ϕ_0, ϕ_1, ϕ_2 de H, e os respetivos valores próprios E_0, E_1, E_2 de H são o inverso dos menores valores próprios λ_0, λ_1, λ_2 de M. O código g#s5c5.cpp aqui desenvolvido deve ainda escrever os vetores próprios, já normalizados, num ficheiro e no ecran, e os valores próprios noutros ficheiros.

V.12

Nesta alínea verificamos se os resultados obtidos em V.11) estão corretos. Comece por representar graficamente as funções de onda, ou seja os diferentes vetores próprios $\phi_n(x_i)$ em função dos valores discretizados x_i. Note que o estado fundamental, de $n = 0$, é uma função gausseana, e que os pontos numéricos devem descrever bem esta função, ou seja os limites do intervalo $[-\Lambda, \Lambda]$ não devem ser nem demasiado grandes nem demasiado pequenos.

Compare também os valores próprios que obtem com os obtidos analiticamente, dados por $E_n = \hbar\sqrt{\frac{k}{m}}(n + \frac{1}{2})$ sendo $n = 0, 1, 2...$ o número quântico do oscilador harmónico. Que valores para h deve tomar se pretender ter o 1o valor próprio (ou seja o mais baixo E_0, correspondente a $n = 0$) correcto até à 4a casa decimal? Note que o limite contínuo, para o qual obtemos os resultados exatos, é obtido quando $\Lambda \to \infty$ e $h \to 0$.

E ainda calcule numericamente os produtos internos dos diferentes vetores, e represente-os na forma de uma matriz M, de dimensão 3×3, de elementos da matriz $M_{n,n'} = \sum_i \phi_{ni}\phi_{n'i}$. Nota: devido à funções de onda constituírem uma base ortonormada, esta matriz deverá ser próxima da matriz identidade.

VI EDDP em 2+1 dimensões, também com CUDA

Partindo dos códigos em C++ desenvolvidos nas aulas e presentes na página da cadeira, pretende-se resolver a equação de Gross-Pitaevski que é uma equação às derivadas parciais no tempo e no espaço. Este trabalho é o pretexto para que os alunos treinem a matéria das aulas anteriores (migração para C++, arrays, ponteiros, funções, namespaces, strings, estruturas/classes, overload de operadores, derivação e templates). Também, para quem quiser ir mais longe, há aqui a oportunidade de trabalhar com programação paralela, OPENMP e CUDA.

VI.1

Deve começar por rever todas as aulas teóricas anteriores, copiar os códigos apresentados para ficheiros .cpp e testar a sua compilação e a sua execução. Depois deverá compara-los com os conceitos aqui referidos.

Explique - matéria da aula 17 - a vantagem de usar templates e contentores da standard template library C++. Construa um código simples g#s6c1a.cpp que exemplifique o uso da biblioteca da STL para números complexos, e ainda que trabalhe com templates e contentores da STL.

VI.2

Explique - matéria da aula de OPENMP - em que caso(s) é vantajosa a computação paralela, bem como refira algumas dificuldades inerentes à computação paralela. Construa um código simples g#s6c1b.cpp, com openmp, que exemplifique o cálculo paralelo.

VI.3

Construa um código g#s6c2a.cpp uma classe em C++, com overload de operadores, para trabalhar com números complexos, incluindo, a soma, o produto, a diferença, a divisão, e o valor absoluto. Inclua ainda a parte real e a parte imaginária.

VI.4

Teste a sua classe para complexos, usando o código g#s6c1a.cpp para comparar resultados.

VI.5

Crie um código g#s6c3a.cpp em C++, usando o overload de operadores para complexos desenvolvido em VI.3), que resolva a equação de Gross-Pitaevskii a $d = 2 + 1$ dimensões (Schrödinger não linear para condensados de Bose Einstein, com duas dimensões espaciais e uma temporal),

$$(i - \gamma)\frac{\partial \psi}{\partial t} = -\frac{1}{2}\nabla^2\psi + \frac{x^2 + y^2}{2}\psi + G\left|\psi\right|^2\psi - i\Omega\left(x\frac{\partial}{\partial y} - y\frac{\partial}{\partial x}\right)\psi \ . \tag{1}$$

onde o laplaceano tem derivadas no espaço apenas, e onde a função $\psi = \psi(x, y, t)$ é complexa e está normalizada. A sua norma é definida pelo integral,

$$\mathcal{N} = \int \bar{\psi}(x, y, t)\psi(x, y, t)dxdy \ . \tag{2}$$

A discretização da equação $\psi(x, y, t) \to \psi^n_{i\,j}$ é feita no espaço com as técnicas já estudadas para a EDDP de Poisson na série 4 e no tempo com as técnicas já estudadas para as EDO do decaimento radioativo da série 3. Para **initializar** a resolução da equação considere o seguinte,

- parâmetros,
  trabalhe em unidades adimensionais,

$$\gamma = 0.01 \ , \tag{3}$$
$$\Omega = 0.85 \ , \tag{4}$$
$$G = 1000 \ . \tag{5}$$

- tamanho da rede,

  trabalhe com uma rede com 128 x128 pontos nas dimensões espaciais, correspondendo a uma caixa de dimensões $[-10, 10]\times[-10, 10]$, rede para a qual os parâmetros foram afinados,

- condição fronteira,

$$\psi^n_{0\,j} = 0.0 \tag{6}$$
$$\psi^n_{n_x-1\,j} = 0.0 \tag{7}$$
$$\psi^n_{i\,0} = 0.0 \tag{8}$$
$$\psi^n_{i\,n_y-1} = 0.0 \tag{9}$$

- condição inicial a $t = 0$,

$$\psi^0_{i\,j} = 1.0 \text{ quando } 0 < i < n_x - 1 \ , 0 < j < n_y - 1 \ , \tag{10}$$

  podendo em alternativa optar por adicionar uma pequena contribuição aleatória a este valore

- normalização,
  para completar a condição inicial, e em cada iteração no tempo é necessário voltar a renormalizar a função de onda ψ , para que $\mathcal{N} = 1$, calculando o integral (2) numericamente, e substituindo

$$\psi(x, y, t) \to \frac{\psi(x, y, t)}{\sqrt{\mathcal{N}}} \ . \tag{11}$$

A cada **iteração** no tempo t deve realizar os seguintes passos,

- primeiro determine o lado direito da Eq. (1), que no espaço a $D = 2$ inclui 4 diferentes derivadas em x ou y, que deve calcular com diferenças finitas centradas, no espaço a 2D, x e y determinadas com,

$$\frac{\partial\psi}{\partial x} = \frac{\psi^n_{i+1,j} - \psi^n_{i-1,j}}{2\Delta x} \tag{12}$$
$$\frac{\partial\psi}{\partial y} = \frac{\psi^n_{i,j+1} - \psi^n_{i,j-1}}{2\Delta y} \tag{13}$$
$$\frac{\partial^2\psi}{\partial x^2} = \frac{\psi^n_{i+1,j} - 2\psi^n_{i,j} + \psi^n_{i-1,j}}{(\Delta x)^2} \tag{14}$$
$$\frac{\partial^2\psi}{\partial y^2} = \frac{\psi^n_{i,j+1} - 2\psi^n_{i,j} + \psi^n_{i,j-1}}{(\Delta y)^2} \tag{15}$$

- no tempo, temos uma equação diferencial da forma $\frac{\partial \psi}{\partial t} = F(\psi)$ onde $F(\psi)$ denota o lado direito da Eq. (1); e resolvemos esta equação diferencial no tempo com diferenças finitas avançadas:
  - a equação de evolução no tempo (índice superior) em 1ª ordem é da forma,

$$\psi_{i,j}^{n+1} = \psi_{i,j}^{n} + \Delta t F(\psi_{i,j}^{n}) \ , \tag{16}$$

  - mas esta eq. no tempo só converge pelo critério de CFL desde que tenhamos $\Delta t < 0.5 \Delta x \, \Delta y$,
  - e ainda, para termos bons resultados, se usarmos o método de Runge-Kutta de pelo menos 3ª ordem,

$$\psi_{i,j}^{(1)} = \psi_{i,j}^{n} + \Delta t F(\psi_{i,j}^{n}) \tag{17}$$

$$\psi_{i,j}^{(2)} = \frac{3}{4}\psi_{i,j}^{n} + \frac{1}{4}\psi_{i,j}^{(1)} + \frac{1}{4}\Delta t F(\psi_{i,j}^{(1)}) \tag{18}$$

$$\psi_{i,j}^{n+1} = \frac{1}{3}\psi_{i,j}^{n} + \frac{2}{3}\psi_{i,j}^{(2)} + \frac{2}{3}\Delta t F(\psi_{i,j}^{(2)}) \tag{19}$$

- e finalmente renormalizamos ψ com a Eq.(11) antes de darmos o passo seguinte em t, ou de guardarmos a solução $\psi_{i,j}^{n}$ num ficheiro.

É Importante que construa este código pouco a pouco e que teste cada passo antes de avançar para o passo seguinte. Deve **notar** os erros que efetuou e a forma como os detetou. É impossível fazer este código algo completo sem cometer erros, pelo que se espera uma interessante lista de erros.

VI.6

Estenda agora o seu código para um novo código g#s6c3b.cpp, implementando o OpenMP. Comece por explicar a sua estratégia para paralelizar o código, ou seja que parte do código pretende dividir paralelamente pelos diferentes threads, e que parte pretende manter em série. Quantifique quantas vezes mais rápido corre o código com OPENMP que o código em C++ simples,

VI.7

Facultativamente, estenda o código anterior para um código g#s6c4.cu em CUDA usando a "Global memory". Começe por criar uma classe para criar um novo tipo, variável complexa com duas entradas. Em seguida, irá trabalhar com um array unidimensional de complexos, transformando o array a 2 dimensões num array 1 dimensão, com índice a variar entre 0 e $n_x n_y - 1$.
Nota: Para calcular a normalização pode adaptar o código "reduction"que vem com o pacote SDK ou o reduction da Thrust library.

VI.8

Crie um filme com o "density plot"resultante da densidade, isto é $|\psi(x,y)|^2$, a variar com o tempo. Para tal, guarde em sucessivos ficheiros cujo nome inclua o respetivo tempo, um frame de 10 em 10 unidades de tempo, até um tempo final de 1000 unidades. Deve "upload"ar o vídeo para o YOUTUBE. Identifique o vídeo com o nome do trabalho e do grupo, por exemplo FCOM13S6T1G0_LeonorJoão.

Nota: Use o script de GNUPLOT "criar_png_video.sh"disponível na página da cadeira, se tiver dúvidas consulte o ficheiro "help.txt".

Para quem prescinda de usar CUDA, pode usar o código de VI.7) para realizar o vídeo.

VI.9

Para quem pretenda treinar muito mais o cálculo paralelo gastando horas extaordinárias de trabalho **como se fosse um hobby de CUDA** e candidatar-se a um ponto de bonus há que fazer o seguinte:

- repetiro problema VI.8) substituindo primeiro a "Global memory"por "Texture memory"e depois por "Shared memory", e verificar em cada caso quantas vezes mais rápido corre o código que em C++ simples,
- estendendo o código para 3 dimensões espaciais - mas mantendo uma dimensão apenas no tempo - em que o termo da rotação se mantém em torno do eixo dos z,
- deve ainda fazer uma pequena apresentação de 15' sobre o trabalho.

Para aprofundar a matéria, em seguida a estudar os slides e da página da cadeira, pode estudar os seguintes links,
http://openmp.org/wp/
http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html
http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html