IST- MEFT - Física Computacional - 31/12/2014
Prof. Fernando Barão
Grupo B09

Pedro Pereira: 78889
João Alves: 79006

Projecto Física Computacional - Memória Descritiva

Resumo

Neste trabalho estudaram-se acontecimentos físicos relacionados com raios cósmicos. O trabalho engloba simulação, reconstrução e análise estatística deste fenómenos. Na 1ª parte temos a simulação dum detetor terrestre, na 2ª parte temos uma análise das trajetórias das partículas e condicionantes destas através das suas equações diferenciais, onde é necessário implementar métodos numéricos para as resolver.

1 Parte A

Teve-se o cuidado de à priori identificar e classificar as componentes do programa e definir bem a sua independência. Por isso, criou-se a classe Event que tem dupla funcionalidade. Pode ser usada como simulador e reconstrutor de um ou vários eventos e como desenhadora dos eventos e analista (criar histogramas com base em todos os eventos), ou as duas simultaneamente, tal como é feito. Estas duas componentes da classe são independentes, devido ao uso de ficheiros. Isto é, ao gravar todas as informações das simulações num ficheiro, podemos, mais tarde desenhar qualquer um dos eventos criados e analisar um histograma ou só uma parte do espetro total de eventos. Fez-se desta forma inspirando-se um pouco no mundo real, onde por vezes na análise estatística se detetam coisas inesperadas e é necessário uma análise pormenorizada em certas zonas/eventos.

1.1 Particularidades da classe

Parte do código deve-se à tradução das numerações do computador para algo que uma pessoa possa entender e identificar. Verificou-se que o próprio ROOT numerava os pixéis, mas duma forma deficiente para humanos. O primeiro pixel do TH2 era o número 83 para o ROOT, e sempre que transitava de linha, saltava dois valores na numeração. Assim, o array PixID, é algo que grava o número dado pelo ROOT de todos os pixéis que de facto pertencem à matriz, sendo que o indice de cada número é o identificador real. Por esta razão existem os vetores com um H a preceder o nome, são vetores criados após a tradução da numeração do ROOT para a humana, e são estes que são mostrados ao utilizador. Também se verificou que o root considerava cada pixel como um ponto, sem espessura, sendo por vezes necessário dividir por Pixwidth para descomplexar os cálculos, e multiplicando no fim, para retornar algo que faça sentido para um humano. Parte do programa foi, assim, tradução e mudança de coordenadas, entre máquina e homem.

1.2 A classe Event como simulador e reconstrutor

Para dar uso a esta classe instacia-se um objeto dentro dum loop com limite correspondente ao número de eventos que se pretende simular. Dentro do Loop chamam-se as funções MakeEvent e RecEvent. No que diz respeito à simulação do ruído não há nada a apontar. Na simulação do sinal aleatório tentou-se simular ao máximo a realidade, sendo que de acordo com a minha interpretação, no momento do embate 5 pixeis nas imediações do ponto de colisão são ativados aleatoriamente. Isto traduz-se matematicamente em escolher uma de 9C5 Combinações, isto é, uma de 126. Assim definiram-se todas estas, e seleciona-se uma delas através dum número aleatório entre 0 e 125. Esta parte sobrecarrega muito o código em termos de linhas, contudo ao ser utilizado um switch só um dos casos é executado, não prejudicando em termos de eficiência. Em

relação ao sinal do anel, fez um varrimento de 0 a 2π, com um passo de $\frac{1.5}{10R}$, o ângulo em radianos correspondente a um décimo de um troço de circunferencia de 1.5 mm. Para a análise angular, sempre que se entra num pixel novo, grava-se o ângulo atual. Em cada iteração grava-se num vetor auxiliar o número do pixel, através da função GetBin. Para fazer a seleção dos pixeis possíveis de serem ativados, basta notar que os pixeis que se repetem 10 ou mais vezes no vetor inicial, precisaram de um ângulo em radianos superior ao correspondente a um troço de 1.5 mm, cumprindo assim o critério de ativação. Desse novo vetor de pixeis selecionáveis, escolhem-se N0 e faz-se a respetiva tradução. A função MakeEvent recebe todos os vetores das funções anteriores e escreve as variáveis num ficheiro. Para a função que reconstrói o evento, assumiu-se que os pixeis de ruído se distinguem dos de sinal(azul e vermelho), sendo possível obter um valor aproximado para N0. A este valor soma-se 0.5 e assume-se um erro de ± 0.5. Desse valor obtém beta, com um erro dado por:

$$\frac{2\sqrt{5}}{1.3} \cdot \frac{1}{2\sqrt{(20-N0)^3}} \cdot e_{N0} \tag{1}$$

e R com um erro dado por:

$$\frac{40}{\sqrt{(1.3\beta)^2-1}} \cdot (1.3\beta) \cdot e_\beta \tag{2}$$

Seguidamente é feita uma análise da zona central, ativada, onde se regista os valores máximos e mínimos de x e y. Nesta retângulo estará o ponto onde a partícula colidiu. Selecionando um pixel da circunferencia e utilizando o seu ponto central, (px,py) faz-se um varrimento de todos os valores de x para um y fixo do retângulo referido em cima onde se regista os valores de x e y para o qual é minimizada a equação:

$$(x-px)^2 + (y-py)^2 - R^2 = 0 \tag{3}$$

O erro da posição corresponde ao erro da seleção do ponto central do pixel da circunferência como ponto que pertence a esta, 2.5, e a um erro derivado do erro do raio, obtendo-se:

$$2.5 + \sqrt{\frac{eR^2}{2}} \tag{4}$$

Para o cálculo do tempo de reconstrução inicia-se uma variável start com o valor atual de std::clock() e no fim faz-se o cáculo:

$$\frac{1000(std::clock() - start)}{(double)(CLOCKS\_PER\_SEC)} \tag{5}$$

1.3 A classe Event como Desenhador e Analisador

Para usarmos a classe com esta funcionalidade podemos instaciar outro objeto fora do loop, onde chamamos as funções InfoDraw e Hist. InfoDraw dá-nos o desenho do evento e uma informação detalhada deste, comparando diretamente os valores reais com os valores reconstruídos. Hist chama todos os histogramas, lendo no ficheiro todos os eventos. Para chamar só um dos histogramas basta chamar a função correspondente, e para fazer o histograma para uma porção do espetro total de eventos basta colocar restrições no iterador k que varre todas as linhas do ficheiro. Caso queiramos analisar um evento singular com uma dada caraterística, por exemplo um evento cuja diferença entre betas tenha sido um valor **a**, basta colocar uma condição na função que realiza o histograma das velocidades, para quando $beta - betaR = a$ print n, sendo n o contador de eventos, que evolui a cada 24 linhas. Tendo o valor de n, podemos fazer InfoDraw(n), obtendo assim informação detalhada deste evento. Para fazer isto é fulcral nao chamar a função CleanData (como é óbvio, pois queremos os dados que estão no ficheiro, da simulação anterior) e não fazer MakeEvent ou RecEvent.

1.4 Considerações Finais

Concluindo, podemos usar esta classe para gerar e reconstruir n eventos, e fazer uma análise geral através de Hist. Caso haja algum histograma com algo de interesse, podemos descobrir qual é o eveto ou eventos que causam isso, e análisá-los em detalhe através de InfoDraw. Como exemplo, em DetSim.C, temos uma

geração e reconstrução de 10000 eventos e o desenho do primeiro de todos, seguida da análise geral dos eventos.

2 Parte B

2.1 Alínea a

Neste exercicio, pretendia-se a obtençao das linhas de campo magnetico. Para isso, usou-se a funçao MagLines(double x, double y, double z, int n, double h). Esta baseia-se no facto de o vetor campo magnetico ser tangencial as linhas de campo em todos os pontos, para construir estas com base naquelas. Assim, partindo de um ponto inicial (x,y,z), somaram-se vetores campo magnetico a este suficientemente pequenos, sendo que a figura obtida tende para as linhas de campo quando o tamanho dos vetores usado tende para zero. Optou-se por normalizar o vetor antes de o multiplicar pelo passo h, uma vez que o campo magnetico e mais forte quanto mais perto se estiver do centro da terra. Contudo, nessa zona obtem-se tambem as maiores variaçoes das linhas de campo, sendo portanto necessaria mais informaçao nessa zona. Normalizando o vetor, consegue-se isso.

2.2 Alínea b

A equaçao que descreve o movimento da partícula é dada por:

$$\vec{F} = q\vec{v}\times\vec{B} \Leftrightarrow \frac{\mathrm{d}}{\mathrm{d}t}(\gamma m\vec{v}]) = qB_o(\frac{Re}{r})^3\vec{v}\times(3 < \frac{\vec{d}}{d}, \frac{\vec{r}}{r} > \frac{\vec{r}}{r} - \frac{\vec{d}}{d}) \quad (6)$$

Para coordenadas cartesianas, a equação toma a forma:

$$\frac{m\frac{\mathrm{d}}{\mathrm{d}t}\vec{v}}{sqrt(1-\frac{v^2}{c^2})} = qB_o(\frac{Re}{sqrt(x^2+y^2+z^2)})^3\vec{v}\times(\frac{3z\vec{r}}{r^2} - 1\vec{e_z}) \quad (7)$$

Dessa forma, obtém-se o seguinte sistema de equaçoes, para x, y, e z:

$$\begin{cases} \frac{\mathrm{d}}{\mathrm{d}t}\vec{v_x} = \frac{qB_oR_e^3(3z^2v_y - 3yzv_z - v_y(x^2+y^2+z^2)^2)}{m(x^2+y^2+x^2)^{\frac{5}{2}}sqrt(1-\frac{v_x^2+v_y^2+v_z^2}{c^2})} \\ \frac{\mathrm{d}}{\mathrm{d}t}\vec{v_y} = \frac{qB_oR_e^3(3xzv_z - 3v_xz^2 - v_x(x^2+y^2+z^2)^2)}{m(x^2+y^2+x^2)^{\frac{5}{2}}sqrt(1-\frac{v_x^2+v_y^2+v_z^2}{c^2})} \\ \frac{\mathrm{d}}{\mathrm{d}t}\vec{v_z} = \frac{qB_oR_e^3(3v_xyz - 3xv_yz)}{m(x^2+y^2+x^2)^{\frac{5}{2}}sqrt(1-\frac{v_x^2+v_y^2+v_z^2}{c^2})} \end{cases}$$

Fazendo $Z = v_z$, $Y = v_y$ e $X = v_x$, este sistema passa a ser o seguinte:

$$\begin{cases} \frac{\mathrm{d}}{\mathrm{d}t}x = X \\ \frac{\mathrm{d}}{\mathrm{d}t}y = Y \\ \frac{\mathrm{d}}{\mathrm{d}t}z = Z \\ \frac{\mathrm{d}}{\mathrm{d}t}X = \frac{qB_oR_e^3(3z^2Y - 3yzZ - Y(x^2+y^2+z^2)^2)}{m(x^2+y^2+x^2)^{\frac{5}{2}}sqrt(1-\frac{X^2+Y^2+Z^2}{c^2})} \\ \frac{\mathrm{d}}{\mathrm{d}t}Y = \frac{qB_oR_e^3(3xzZ - 3Xz^2 - X(x^2+y^2+z^2)^2)}{m(x^2+y^2+x^2)^{\frac{5}{2}}sqrt(1-\frac{X^2+Y^2+Z^2}{c^2})} \\ \frac{\mathrm{d}}{\mathrm{d}t}Z = \frac{qB_oR_e^3(3Xyz - 3xYz)}{m(x^2+y^2+x^2)^{\frac{5}{2}}sqrt(1-\frac{X^2+Y^2+Z^2}{c^2})} \end{cases}$$

Assim, obtém-se um sistema de equações diferenciais de primeira ordem, ao qual já se pode aplicar o método de Runge-Kutta.

2.3 Alínea c

Para tratar o problema da particula sob a influência do campo magnético terrestre, criou-se a classe MagField, constituída pelas seguintes funções:

class MagField public: MagField(double Q=q, double M=Mp, double B=b, double R=Re); static double* MagForce (double x, double y, double z); void MagLines(double x, double y, double z, double h, int n); void PartTraj1(double x, double y, double z, double vx, double vy, double vz, double ha, int n, char cc='m'); void PartTraj2(double x, double y, double z, double vx, double vy, double vz, double ha, int n, char cc='m'); void Draw(string); private: vector<TPolyLine3D*> vp=NULL; TF1 *f; TF1 *g; TF1 *h;

Esta classe guarda como variáveis privadas três TF1's e um vector de ponteiros para objetos TPolyLine3D. Os objetos TF1 correspondem às três funções definidas por $\frac{d}{dt}X = f(x,y,z,X,Y,Z)$, $\frac{d}{dt}Y = g(x,y,z,X,Y,Z)$ e $\frac{d}{dt}Z = h(x,y,z,X,Y,Z)$, respetivamente. Optou-se por guardá-los como variável interna da classe de modo a permitir à mesma o tratamento de problemas diferentes, como por exemplo a trajetória de um eletrão em vez de um protão. Quanto aos objetos TPolyLine3D, correspondem às trajetórias descritas pela particula. Foram guardadas sobre a forma de um vector, o que permite mais de uma trajetória por objeto.

O construtor da classe permite a inicializaçao dos parâmetros que caraterizam o sistema (carga e massa

da particula, por exemplo), de modo a oferecer à classe a capacidade de tratar qualquer problema semelhante. Note-se que se inicializam os valores pretendidos neste exercicio como default.

Não foi implementado um destrutor uma vez que se utilizou a ferramenta cFCgraphics, fornecida pelo professor, que destrói as variáveis privadas da classe.

Os dois métodos usados para a obtenção da trajetória sao o *PartTraj1* e *PartTraj2*. Estes recebem a posição (x,y,z) e velocidade (vx,vy,vz) inicial da partícula. A *PartTraj1* aplica-lhe o método de RK2 com passo *ha* n vezes, enquanto que o *PartTraj2* usa um método de RK4. Têm ainda uma varíavel de controlo *charcc*, que permite selecionar o modo de paragem. Caso seja diferente de 'r', utiliza o critério descrito anteriormente, tendo apenas o cuidado de parar quando a partícula atinge a terra, visto a trajetória ser interrompida. Caso contrário, toma em consideração a posição da particula não poder ser $r > 25 * Re$. Optou-se por instanciar ambos os métodos de Runge-Kutta, visto o de segunda ordem obrigar à utilização de um passo demasiado pequeno para garantir a convergência da solução (neste caso, a conservação do movimento). Contudo, o método RK4 é consideravelmente mais pesado. Assim, optou-se por realizar ambos, permitindo ao utilizador a escolha do mais apropriado.

A classe possui ainda uma função Draw, que permite desenhar os objetos obtidos. Esta recebe uma variável do tipo string, que permite selecionar o modo de impressão pretendido. A opção implementada por default envolve o desenho da terra e do detetor para cada trajetoria guardada no objeto. Estas são impressas individualmente. Tem-se ainda a opção "together"que imprime todas as trajetórias ao mesmo tempo, a "clean"que não imprime a terra e o detetor, e ainda a "far"e a "close", em que a primeira aumenta o tamanho da janela e a segunda diminui.

2.4 Alínea d

As grandezas encontradas no método aplicado para a resolução do sistema de equações estão em unidades SI. Assim, foi necessário converter a posição do detetor e velocidade da partícula para estas unidades. Sendo que o sistema de equações diferenciais de primeira ordem que descreve o problema está em coordenadas cartesianas, teve-se ainda a necessidade de converter as coordenadas esféricas em coordenadas cartesianas.

Desta forma, obteve-se a posição do detetor $\vec{r} = ((R_t + 500000) \times \frac{sqrt(2)}{2}, 0, (R_t + 500000) \times \frac{sqrt(2)}{2})$, em que R_t corresponde ao raio da terra.

Obteve-se a velocidade da partícula através do momento linear da mesma. Desta forma, conseguiu-se para a particula 1 $\vec{v} = (-2.1 \times 10^7, 0, -2.1 \times 10^7)$, e para a partícula 2 $\vec{v} = (-2.994 \times 10^8, 0, 0)$.

2.5 Alínea e

Para simular o comportamento anterior da partícula, aplicou-se os métodos desenvolvidos nas alíneas anteriores *PartTraj1* e *PartTraj2*. Contudo, utilizou-se um passo negativo, de modo a obter a posição da partícula para tempos $t = t_0 - h$, isto é, $t < t_0$. Assim, prevê-se o comportamento anterior desta.

Referências

[1] *Stroustrup, Bjarne. The C++ Programming Language. Reading, MA: Addison-Wesley, 2013.*

[2] *Barão, Fernando. Slides das Aulas Teóricas. 2014.*