Folhas de problemas de Física computacional

Problemas selecionados e traduzidos do livro

Computational Physics

Mark Newman

Ano lectivo 2018/2019

Docentes:

João Manuel Viana Parente Lopes

José Miguel Nunes da Silva

João Manuel Borregana Lopes dos Santos

Física Computacional

Ano lectivo 2017/2018

Folha 1 - Programação em Python para Físicos

1. Exercício

A órbita espacial de um corpo em torno de outro, como um planeta em torno do Sol, não é necessariamente circular. Em geral assume a forma de uma elipse, com o corpo por vezes mais próximo e por vezes mais afastado. Se na máxima aproximação de um planeta ao Sol, também chamado de *periélio*, a distância é ℓ_1 e sua velocidade linear v_1, então qualquer outra propriedade da órbita pode ser calculado a partir destes dois valores como se segue.

1. A segunda lei de Kepler diz-nos que a distância ℓ_2 e a velocidade v_2 do planeta no ponto de máximo afastamento, o *afélio*, satisfaz $\ell_2 v_2 = \ell_1 v_1$. Também é sabido que a energia mecânica total, cinética mais potencial gravítica, do planeta com velocidade v a uma distância r do Sol é dada por

$$E = \frac{1}{2}mv^2 - G\frac{mM}{r},$$

onde m é a massa do planeta, $M = 1.9891 \times 10^{30}\,\mathrm{kg}$ a massa do Sol, e $G = 6.6738 \times 10^{-11}\,\mathrm{m^3\,kg^{-1}\,s^{-2}}$ a constante de gravitação universal. Dado que a energia mecânica deve ser conservada, mostre que v_2 é a menor das raízes da equação quadrática

$$v_2^2 - \frac{2GM}{v_1\ell_1}v_2 - \left[v_1^2 - \frac{2GM}{\ell_1}\right] = 0.$$

Uma vez obtido o valor de v_2 podemos calcular ℓ_2 usando a relação $\ell_2 = \ell_1 v_1/v_2$.

2. Conhecidos os valores de v_1, ℓ_1, e ℓ_2, outros parâmetros da órbita são dados por expressões simples obtíveis da geometria elíptica da órbita e das leis de Kepler:

$$\begin{aligned}
\text{Semi-eixo maior:} \quad & a = \frac{1}{2}(\ell_1 + \ell_2), \\
\text{Semi-eixo menor:} \quad & b = \sqrt{\ell_1\ell_2}\,, \\
\text{Período orbital:} \quad & T = \frac{2\pi ab}{\ell_1 v_1}\,, \\
\text{Excentricidade orbital:} \quad & e = \frac{\ell_2 - \ell_1}{\ell_2 + \ell_1}.
\end{aligned}$$

Desenvolva um programa que peça ao utilizador a distância do planeta ao Sol e a sua velocidade no periélio e, depois, calcule e mostre os valores das quantidades ℓ_2, v_2, T, e e.

3. Teste o programa fazendo o cálculo das propriedades orbitais da Terra (para a qual $\ell_1 = 1.4710 \times 10^{11}\,\mathrm{m}$ e $v_1 = 3.0287 \times 10^4\,\mathrm{m\,s^{-1}}$) e do cometa Halley ($\ell_1 = 8.7830 \times 10^{10}\,\mathrm{m}$ e $v_1 = 5.4529 \times 10^4\,\mathrm{m\,s^{-1}}$).

Entre outras coisas, deve obter para a Terra um período orbital de um ano, e de 76 anos para o cometa Halley.

2. Exercício

Os números de Catalan C_n são uma sequência de inteiros 1, 1, 2, 5, 14, 42, 132...com um papel importante em Mecânica Quântica e na Teoria dos Sistemas Desordenados. (Estão na base da demonstração por Eugene Wigner da famosa 'Lei do Semicírculo'.) Estes números podem ser definidos por recorrência:

$$C_0 = 1, \qquad C_{n+1} = \frac{4n+2}{n+2} C_n.$$

Desenvolva um programa que mostre, por ordem crescente, todos os números de Catalan até um milhar de milhão.

3. Exercício

Em Física da Matéria Condensada a constante de Madelung representa a energia potencial elétrica total de um ião na rede cristalina. Ela depende das cargas dos outros iões vizinhos e das suas localizações. Considere, por exemplo, o caso do Cloreto de Sódio (sal das cozinhas) onde os iões estão dispostos alternadamente numa rede cúbica simples: o de sódio com uma carga positiva $+e$ e o de cloro com uma carga negativa $-e$, onde e a carga do eletrão. Se etiquetarmos cada nodo da rede por três números inteiros (i, j, k), então os iões sódio ficam nas posições onde $i + j + k$ é par, e os iões cloro nas posições onde $i + j + k$ é ímpar.

Consideremos o ião sódio situado na origem, $i = j = k = 0$, e calculemos a constante de Madelung. Se o espaçamento entre os iões for$a$ (constante da rede), então a distância da origem ao ião na posição (i, j, k) é

$$\sqrt{(ia)^2 + (ja)^2 + (ka)^2} = a\sqrt{i^2 + j^2 + k^2},$$

e o potencial na origem criado por esse ião é

$$V(i, j, k) = \pm \frac{e}{4\pi\epsilon_0 a\sqrt{i^2 + j^2 + k^2}},$$

(ϵ_0 é a permitividade do vazio e o sinal da expressão depende de $i + j + k$ ser par ou ímpar). O potencial total na origem V_{total} é então a soma sobre todas as restantes posições. Supondo um cristal de formato cúbico, e centrado na origem, com L iões em todas as direções, vem

$$V_{\text{total}} = \sum_{\substack{i,j,k=-L \\ \text{not } i=j=k=0}}^{L} V(i, j, k) = \frac{e}{4\pi\epsilon_0 a} M,$$

onde M é (no limite $L \to \infty$) a constante de Madelung.

Desenvolva um programa que calcule a constante de Madelung do Cloreto de Sódio. Use um valor suficientemente grande para L mas que lhe permita correr o programa num tempo não superior a um minuto. Compare com o valor exato para o NaCl: $-1,748$.

4. Exercício

O coeficiente binomial $\binom{n}{k}$ é um número inteiro igual a

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n \times (n-1) \times (n-2) \times \ldots \times (n-k+1)}{1 \times 2 \times \ldots \times k}$$

com $k \geq 1$, ou $\binom{n}{0} = 1$ quando $k = 0$.

1. Usando este formulário para o coeficiente binomial, escreva uma função $binomial(n, k)$ que calcula o coeficiente binomial para n e k. Certifique-se de que sua função retorna a resposta na forma de um inteiro (não um float) e forneça o valor correto de 1 para o caso em que $k = 0$.

2. Usando sua função, escreva um programa para imprimir as primeiras 20 linhas do "triângulo de Pascal". A linha n do triângulo de Pascal contém $n + 1$ números, que correspondem aos coeficientes $\binom{n}{0}$, $\binom{n}{1}$, até $\binom{n}{n}$. As primeiras linhas são,

   1 1
   1 2 1
   1 3 3 1
   1 4 6 4 1

3. A probabilidade de uma moeda justa, lançada n vezes, apresenta k resultados de caras é $\binom{n}{k}/2^n$. Escreva um programa que calcula (a) A probabilidade the uma moeda ser lançada 100 vezes e ter como resultado 60 caras, e (b) a probabilidade de ter como resultado pelo menos 60 caras.

5. Exercício

O programa no Exemplo 2.8 do livro adoptado não corresponde ao modo mais eficiente de calcular números primos: verifica cada número para ver se é divisível por qualquer número menor que ele. Podemos desenvolver um programa muito mais rápido para números primos usando as seguintes observações:

1. Um número n é primo se não tiver nenhum fator primo menor que n. Por isso, só precisamos verificar se é divisível por outros primos.

2. Se um número n é não primo, tendo um fator r, então $n = rs$, onde s também é um fator. Se $r \geq \sqrt{n}$ então $n = rs \geq s\sqrt{n}$, o que implica que $s \leq \sqrt{n}$. Em outras palavras, qualquer não-primo deve ter fatores e, portanto, também fatores primos, menores ou iguais a $\sqrt{n}$. Assim, para determinar se um número é primo, temos que verificar seus fatores primos apenas até e incluindo $\sqrt{n}$ - se não houver nenhum, o número é primo.

3. Se encontrarmos um único fator primo menor que $\sqrt{n}$, então sabemos que o número é primo e, portanto, não há necessidade de checar mais nada - podemos abandonar esse número e passar para outra coisa..

Escreva um programa Python que encontre todos os primos até dez mil. Crie uma lista para armazenar os primos, que começa com apenas um número primo 2 nele. Então, para cada número n de 3 a 10 000, verifique se o número é divisível por qualquer um dos primos da lista, até e incluindo $\sqrt{n}$. Assim que você encontrar um único fator primo, poderá parar de verificar o restante deles - você sabe que n não é um primo. Se você não

encontrar fatores primos $\sqrt{n}$ ou menos, então n é primo e você deve adicioná-lo à lista. Você pode imprimir a lista de uma só vez no final do programa, ou pode imprimir os números individuais à medida que os encontrar.

6. Exercício

Um recurso útil de funções definidas pelo usuário é a capacidade de recursão de uma função para se chamar. Por exemplo, considere a seguinte definição do fatorial $n!$ de um inteiro positivo n:

$$n! = \begin{cases} 1 & \text{se } n = 1, \\ n \times (n-1)! & \text{se } n > 1. \end{cases}$$

Isto constitui uma definição completa do fatorial que nos permite calcular o valor de $n!$ para qualquer inteiro positivo. Podemos empregar essa definição diretamente para criar uma função Python para fatoriais, como este:

```
def factorial(n):
    if n==1:
        return 1
    else:
        return n*factorial(n-1)
```

Note como, se n não for igual a 1, a função chama a si própria para calcular o fatorial de $n - 1$. A este procedimentos chamamos recursão. Se executarmos factorial(5) o computador irá imprimir corretamente a resposta 120.

1. Encontramos os números de Catalan C_n no Exercício 2. Com apenas um pequeno rearranjo, a definição pode ser reescrita na forma

$$C_n = \begin{cases} 1 & \text{se } n = 0, \\ \dfrac{4n-2}{n+1} C_{n-1} & \text{se} n > 0. \end{cases}$$

   Escreva uma função Python, usando recursão, que calcule C_n. Use essa função para calcular e imprimir C_{100}.

2. Euclides mostrou que o maior divisor comum $g(m, n)$ de dois inteiros não negativos m e n satisfaz

$$g(m,n) = \begin{cases} m & \text{if } n = 0, \\ g(n, m \bmod n) & \text{if } n > 0. \end{cases}$$

   Escreva uma função Python que emprega recursão para calcular o maior divisor comum de m e n usando essa fórmula. Use sua função para calcular e imprimir o maior divisor comum de 108 e 192.

Comparando o cálculo dos números de Catalan na parte (a) acima com o do Exercício 2, vemos que é possível fazer o cálculo de duas maneiras, diretamente ou usando recursão. Na maioria dos casos, se uma quantidade puder ser calculada sem recursão, será mais rápido fazer isso, e normalmente recomendamos que você faça essa rota, se possível. Existem alguns cálculos, no entanto, que são essencialmente impossíveis (ou pelo menos muito mais difíceis) sem recursividade. Veremos alguns exemplos mais adiante neste livro.