Experimentação de cálculo numérico para a obtenção da Constante de Catalan

Alonso Ali
Pontifícia Universidade Católica do Rio Grande do Sul

October 25, 2016

1 Introdução

No contexto de computação científica é comum a necessidade de utilização de constantes matemáticas que contém uma quantidade de dígitos irrepresentável em um sistema de ponto flutuante. A solução disto é aproximar o valor desejado através de um cálculo iterativo, onde quanto mais iterações realizadas, maior será a exatidão do valor obtido, entretanto, nunca exato.

Neste trabalho estaremos comparando dois métodos para o cálculo da constante de Catalan: o método de Broadhurst e o método matemático comum. O método estipulado por Broadhurst traz a hipótese de convergir mais rápido que o método comum e, portanto, categorizaria este procedimento como um método mais eficaz em contexto computacional para a aproximação da constante. Compararemos os dois métodos através de um teste experimental a fim de comprovar esta hipótese.

2 A Constante de Catalan

A Constante de Catalan, denominada em homenagem a Eugène Charles Catalan e geralmente denotado em contexto matemático como G, aparece muitas vezes em estimativas de funções combinatórias e em certas classes de integrais definidas. Não se sabe até a data deste trabalho se G é um número irracional apesar de já se conhecer a existência de 200,000,001,100 dígitos [1]. A Constante de Catalan pode ser calculada através do somatório

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)^2} = 1 - \frac{1}{3^2} + \frac{1}{5^2} - \frac{1}{7^2} + ...$$

G também pode ser obtido através de integrais definidas, como

$$\int_1^{\infty} \frac{\ln t}{1+t^2} dt$$

$$\int_0^1 \frac{\arctan t}{t} dt$$
$$\int_0^{\frac{\pi}{4}} \frac{t}{\sin t \cos t} dt$$
$$-\int_0^1 \frac{\ln t}{1+t^2} dt$$

Numericamente,

$$G = 0.915965594177\ldots$$

3 O problema

É bem conhecido as limitações impostas por um sistema de ponto flutuante por conta de recursos finitos, como a memória de um computador. Isto introduz um problema para a representação de números com quantidade de dígitos suficientemente grande ou infinita, como π, o número de Euler (e) e a constante de Catalan. A solução para isto é a representação destes números através de um cálculo iterativo e convergente para tal valor, onde cada iteração representa uma aproximação para o número, de maneira que mais iterações impliquem em maior exatidão.

Sabendo que G pode ser representado por um somatório infinito fica evidente a necessidade de um cálculo iterativo para a sua representação em um sistema de ponto flutuante. A partir disso deve-se analisar a eficiência deste cálculo, medindo a velocidade de convergência e a exatidão do valor obtido, a fim de avaliar a performance do método utilizado.

4 Cálculo numérico computacional de G

Dentro do âmbito computacional científico é comum a utilização de um ambiente matemático com o propósito de fazer uso de ferramentas numéricas que manipulam recursos computacionais a fim de aproximar a realidade matemática do contexto computacional. Nesta seção estaremos apresentando o ambiente matemático numérico escolhido, introduzindo as duas maneiras de se computar a constante de Catalan e avaliando-as.

4.1 *NumPy*

NumPy (*Numerical Python*) é um módulo da linguagem *Python* criado para permitir o cálculo numérico de vetores N dimensionais de números utilizando *Python*. A utilização deste módulo recentemente tem se tornado mais recorrente em pesquisas de computação científica por combinar a simplicidade de desenvolvimento de código *Python* e a performance de cálculos computacionais utilizando FORTRAN e C, tendo em vista que os cálculos são realizados através de chamadas de subrotinas destas linguagens a partir de um ecosistema *Python* [3].

Table 1: Resultados do primeiro método com precisão 15

Iteração	**Resultado**	**Erro Relativo (em %)**
1	1	9.1744063703906
2	0.888888888888889	2.9560832263195
3	0.928888888888889	1.4108930284962
4	0.908480725623583	0.8171560811036
5	0.920826404635928	0.5306760963086
6	0.912561941826011	0.3715917249343
7	0.918479101589325	0.2744106796242
8	0.91403465714488	0.2108089042442
9	0.917494864757337	0.1669572077641
10	0.914724781654844	0.1354649705472
...	...	...
103	0.91597737579329	$1.286250940679451 \times 10^{-3}$
104	0.915954038025781	$1.261635973145409 \times 10^{-3}$
105	0.915976931274562	$1.237720872399955 \times 10^{-3}$

4.2 Primeira solução

Sabemos que a constante de Catalan em contexto matemático é calculada pelo somatório [2]

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)^2}$$

Como primeira solução propomos o cálculo iterativo onde cada iteração refere a adição de um termo da série alternada convergente mencionada. O código deste método em nosso ambiente fica:

```
#retorna as n primeiras iteracoes da constante de catalan
def catalan(n):
   seq = np.zeros(n)
   seq[0] = 1
   for i in xrange(1,n):
      seq[i] = seq[i-1] + (1.0/(2 * i + 1)**2) * (-1)**i
   return seq
```

Note que o código é compacto, facilitando sua escrita e compreensão, uma característica importante adquirida da utilização de *Python* como linguagem. O primeiro passo do algoritmo é inicializar um vetor de tamanho n contendo apenas zeros e então atribuir a primeira posição do vetor como o primeiro termo da sequência. Após isso cada posição do vetor representará uma iteração do nosso método, o cálculo é feito dentro do laço e representa o termo geral do somatório. A Tabela 1 apresenta resultados do método utilizando precisão 15.

Sabendo que

$$G \approx 0.915965594177219015054603514932384110774$$

nota-se que em 105 iterações obteve-se apenas 5 dígitos exatos. Isto serve para demonstrar uma convergência lenta deste método, que no âmbito computacional remete uma má eficiência.

4.3 Segunda solução

Em 1998, Broadhurst [4] introduziu um método de rápida convergência para o cálculo da constante de Catalan. Vejamos o cálculo a seguir.

$$
\begin{aligned}
G = 3\sum_{n=0}^{\infty}\frac{1}{2^{4n}}\left(-\frac{1}{2(8n+2)^2}+\frac{1}{2^2(8n+3)^2}-\frac{1}{2^3(8n+5)^2}\right.\\
\left.+\frac{1}{2^3(8n+6)^2}-\frac{1}{2^4(8n+7)^2}+\frac{1}{2(8n+1)^2}\right)\\
-2\sum_{n=0}^{\infty}\frac{1}{2^{12n}}\left(\frac{1}{2^4(8n+2)^2}+\frac{1}{2^6(8n+3)^2}-\frac{1}{2^9(8n+5)^2}\right.\\
\left.-\frac{1}{2^{10}(8n+6)^2}-\frac{1}{2^{12}(8n+7)^2}+\frac{1}{2^3(8n+1)^2}\right)
\end{aligned}
$$

A partir disto desenvolvemos um algoritmo semelhante ao primeiro, entretanto agora formaremos dois vetores para representar os somatórios. Vejamos o código do cálculo iterativo a seguir.

```
#retorna as n primeiras iteracoes da constante de catalan utilizando um
    metodo mais rapido
def catalan(n):
   seqA = np.zeros(n)
   seqB = np.zeros(n)
   i = 0
   seqA[0] = 1.0/(2**(4*i)) * (-1.0/(2 * (8 * i + 2)**2) + 1.0/(4 * (8 *
       i + 3)**2) - 1.0/(8 * (8 * i + 5)**2) + 1.0/(8 * (8 * i + 6)**2)
       - 1.0/(16 * (8 * i + 7)**2) + 1.0/(2 * (8 * i + 1) ** 2))
   seqB[0] = 1.0/(2**(12*i)) * (1.0/(16 * (8 * i + 2)**2) + 1.0/(64 * (8
       * i + 3) ** 2) - 1.0/(512 * (8 * i + 5)**2) - 1.0/(1024 * (8 * i
       + 6)**2) - 1.0/(4096 * (8 * i + 7)**2) + 1.0/(8 * (8 * i + 1)**2))
   for i in xrange(1,n):
      seqA[i] = seqA[i-1] + (1.0/(2**(4*i)) * (-1.0/(2 * (8 * i + 2)**2)
          + 1.0/(4 * (8 * i + 3)**2) - 1.0/(8 * (8 * i + 5)**2) + 1.0/(8
          * (8 * i + 6)**2) - 1.0/(16 * (8 * i + 7)**2) + 1.0/(2 * (8 *
          i + 1) ** 2)))
      seqB[i] = seqB[i-1] + (1.0/(2**(12*i)) * (1.0/(16 * (8 * i +
          2)**2) + 1.0/(64 * (8 * i + 3) ** 2) - 1.0/(512 * (8 * i +
          5)**2) - 1.0/(1024 * (8 * i + 6)**2) - 1.0/(4096 * (8 * i +
          7)**2) + 1.0/(8 * (8 * i + 1)**2)))
   seqA = seqA * 3
   seqB = seqB * -2
   seqTot = seqA + seqB
```

Table 2: Resultados do segundo método com precisão 15

Iteração	**Resultado**	**Erro Relativo (em %)**
1	0.915421715561225	5.937762503874647 x 10^{-2}
2	0.915956718835822	9.689601283201474 x 10^{-4}
3	0.915965344753118	2.723072817760773 x 10^{-5}
4	0.915965585361346	9.624677291018147 x 10^{-7}
5	0.915965593823505	3.861652231175100 x 10^{-8}
6	0.915965594161837	1.679348124675072 x 10^{-9}
7	0.915965594176512	7.719733676448046 x 10^{-12}
8	0.915965594177185	3.696842159391826 x 10^{-12}
9	0.915965594177217	1.818119094782865 x 10^{-13}
10	0.915965594177219	1.212079396521910 x 10^{-14}

```
return seqTot
```

Nota-se como primeira diferença a perda da simplicidade do código, entretanto, isto é proveniente de um aumento da complexidade do cálculo estipulado por Broadhurst em relação ao utilizado na primeira solução, no entanto as etapas lógicas são análogas: criamos um vetor numérico para cada somatório, atribuímos os valores da primeira iteração de cada um deles em sua respectiva posição e damos início ao cálculo iterativo ao longo do vetor; ao final disto, aplicamos a linearidade do somatório nos vetores ao multiplicá-los pelas constantes adjacentes e somamos os dois vetores. O resultado disto é um único vetor com cada posição contendo o valor da iteração correspondente do método.

A tabela 2 apresenta o resultado das primeiras iterações deste método com precisão 15. Para referência, lembre-se que

$$G \approx 0.915965594177219015054603514932384110774$$

É fácil de ver a velocidade de convergência que este método possui a partir da redução substancial do erro relativo a cada iteração e da obtenção de 16 dígitos exatos em 10 iterações.

5 Conclusão

Fica evidente o ganho de performance da solução proposta por Broadhurst após a apresentação dos resultados em comparação ao método comum. Na primeira solução fez-se necessário o cálculo de 105 iterações para alcançar 5 dígitos exatos, enquanto na segunda, a primeira iteração já alcança 4. Utilizando o segundo método com precisão 15, necessitou apenas 10 iterações para se alcançar o valor

mais exato possível. Então mostra-se a eficiência deste método no cálculo da constante de Catalan, tornando necessário poucas iterações para alcançar uma aproximação satisfatória de G.

References

[1] Catalan's constant. http://www.numberworld.org/digits/Catalan. Acessado: 2016-10-23.

[2] Catalan's constant. http://mathworld.wolfram.com/CatalansConstant.html. Acessado: 2016-10-23.

[3] Numerical calculations with numpy. http://kestrel.nmt.edu/~raymond/software/python_notes/paper003.html. Acessado: 2016-10-23.

[4] D. J. Broadhurst. Polylogarithmic ladders, hypergeometric series and the ten millionth digits of $\zeta(3)$ and $\zeta(5)$. *Arxiv preprint math*, 1998.