Pensamento Computacional e Matemática

Carlos Albuquerque

O que nos ocorre quando se fala em pensamento computacional? Uma primeira reação pode ser pensar em informática. Mas qual a relação com a matemática? Vamos fazer um percurso através de alguns momentos da história da matemática para descobrirmos as múltiplas conexões entre a matemática e a ciência da computação. Vamos também fazer algumas experiências simples com pequenos programas em Python, que podem ser experimentados e modificados usando apenas um browser.

OS CÁLCULOS DE HERÃO

Herão de Alexandria (século I) é conhecido pela fórmula para calcular a área de um triângulo dados os seus lados a, b e c: $A=\sqrt{s(s-a)(s-b)(s-c)}$, onde $s=\frac{1}{2}(a+b+c)$. Herão descreveu também como fazer o cálculo de uma raiz quadrada, exemplificando com o número 720. Para se aproximar $\sqrt{720}$ começa-se por considerar um quadrado perfeito próximo: $729=27^2$. Calcula-se de seguida $\frac{720}{27}$ e a aproximação seguinte é $\frac{1}{2}(27+\frac{720}{27})=26\frac{5}{6}$. Se agora verificarmos qual o quadrado desta aproximação obtemos $\left(26\frac{5}{6}\right)^2=720\frac{1}{36}$ que é uma aproximação muito melhor do que a obtida com 27. Depois:

> Se quisermos fazer a diferença menor do que 1/36, em vez de 729 tomaremos o número agora obtido, 720 1/36, e pelo mesmo método encontraremos uma aproximação com uma diferença muito menor que 1/36. (Katz, 1992, p. 151)

Não temos aqui uma regra geral, mas é fácil inferir um processo para obter aproximações da raiz quadrada de qualquer número positivo: se tivermos uma aproximação x de $\sqrt{a}$ então $\frac{1}{2}\left(x+\frac{a}{x}\right)$ é a aproximação seguinte, que deverá ser melhor. A qualidade da aproximação, que pode funcionar como critério para parar, é aqui avaliada pela diferença entre a e x^2. Também não temos uma demonstração de que o processo continuará a produzir aproximações cada vez melhores.

O exemplo de Herão é típico de uma abordagem da matemática através de exemplos. Katz (1992, p. 7-8) afirma:

> Na maioria dos documentos disponíveis de matemática antiga, o autor descreve o problema a ser resolvido e continua usando uma regra, explícita ou implícita, que fornece a solução. Há pouca preocupação nos documentos sobre como foi descoberta a regra, porque funciona a regra ou quais são as limitações da regra.

O MÁXIMO DIVISOR COMUM EM EUCLIDES

Embora anterior no tempo, Euclides registou uma abordagem mais precisa para determinar o máximo divisor comum de dois números. Cerca de 300 a.C., no livro VII dos Elementos, encontram-se duas proposições que envolvem uma mesma regra: uma versão para decidir se dois números são primos entre si e outra para determinar o máximo divisor comum de dois números não primos entre si.

No enunciado da proposição 1 podemos ler:

> Dados dois números diferentes, e sendo o menor continuamente subtraído, de cada vez, do maior, se o que resta nunca medir o [número] que o precede, até que reste a unidade, então os números originais são primos entre si. (Fitzpatrick, 2008, p. 195)

Podemos dar como exemplo o par (35,48). Subtraindo o menor ao maior ficamos com (35,13). De seguida obtemos (22,13) e ainda (9,13). Continuando fica (9,4), depois (5,4) e finalmente chegamos a (1,4). O aparecimento do 1 assegura que 35 e 48 são primos entre si.

A proposição 2 segue um procedimento semelhante, mas supõe que os números não são primos entre si e pára quando um dos números divide o outro, sendo o menor dos dois o máximo divisor comum.

Estas regras têm pressupostos e critérios de paragem diferentes, mas cada uma está definida de modo geral, sem ser apenas com exemplos, e é dada uma demonstração de que faz o que se pretende.

Uma versão moderna do processo de Euclides, que supõe que está definida a divisão inteira, é um algoritmo que é tipicamente estudado em cursos introdutórios de computação.

ALGORITMOS

A palavra algoritmo é atualmente muito usada, mas tem uma origem matemática que vale a pena rever. O sistema de numeração posicional decimal que usamos, com origem na Índia, chega à Europa através dos árabes de um autor que viveu no atual Uzbequistão, na região de Khwarezm. Por isso o nome deste autor incluía a terminação al-Khwarizmi. O nome de al-Khwarizmi ficou associado ao sistema de numeração e às regras para as operações aritméticas neste sistema, que por vezes se chamava *algorismo*[1]. Essencialmente é o sistema de aritmética

[1] A palavra "algorismo" (do latim medieval "algorismus") foi usada na Europa e é traduzida em inglês por "algorism" e em francês por "algorisme". Contudo, os dicionários portugueses não registam esta palavra.

escrita que hoje usamos, com regras bem definidas para as operações aritméticas. O termo algoritmo também foi usado para indicar o sistema de aritmética, mas evoluiu depois para significar mais geralmente um conjunto de regras para resolver um dado problema.

A TABELA DE COPÉRNICO

Muitos matemáticos ao longo da história fizeram ou dirigiram cálculos numéricos em grande escala. No século XVI Copérnico escreve a sua obra "As revoluções dos orbes celestes". É um tratado técnico de astronomia, mas havia necessidade de ter uma tabela trigonométrica.

Copérnico descreve a tabela que está no seu livro:

> Construímos uma Tabela na escala ascendente de $\frac{1}{6}$ e com três colunas: na primeira estão os graus ou partes de um arco e sextas partes de um grau; a segunda tem o valor numérico da metade da corda correspondente ao dobro do arco; a terceira contém as diferenças destes valores numéricos relativos a cada grau. (2014, p. 71)

A tabela é precedida no capítulo XII por seis teoremas geométricos e um problema, "seguindo Ptolomeu de perto" (2014, p. 64), que servem de fundamentação para os cálculos. Copérnico observa também:

> (...) depois que foi aceite o uso de algarismos hindus. Estes algarismos são superiores a quaisquer outros, gregos ou latinos, devido ao seu manejo extremamente simples, sendo muito vantajosos e próprios para toda a espécie de cálculos. (2014, pp. 63-64)

Não só os teoremas indicam um meio de construir a tabela, como estas observações sugerem que Copérnico terá efetivamente estado envolvido nos cálculos, a ponto de ser sensível às vantagens da aritmética escrita.

A TABELA DE PEDRO NUNES

Se no mar tivermos uma bússola ou outro meio de determinar o Norte, podemos seguir mantendo sempre um ângulo constante com o Norte. Será que a trajetória que seguimos é um grande círculo? Ainda no século XVI Pedro Nunes respondeu a esta questão pela negativa e estudou a curva que se descreve na superfície de uma esfera quando um ponto se move numa trajetória que faz sempre um ângulo constante com o Norte. Chamamos a esta curva linha de rumo ou loxodrómia (Queiró, 2002; Rodrigues, 2017; Loxodrómias e Espirais, 2012).

Nunes foi mais longe e descreveu como calcular pontos de cada curva, de modo aproximado. Sugeriu uma tabela para o efeito, mas o seu interesse pelos cálculos concretos era bem menor do que o de Copérnico, pelo que se limitou a apresentar uma tabela em branco e observou:

> Segue-se a disposição da tábua, repartida em sete partes, e, seguindo as precedentes demonstrações, os moços aplicados acharão os números que se devem escrever nela, estendendo-os quanto lhes aprouver. (Nunes, 2008, p. 482)

Tal como Pedro Nunes houve muitos matemáticos que sentiram a distância entre conceber os processos de cálculo e produzir os resultados. O caráter repetitivo e previsível dos cálculos sugeriu a procura de soluções mecânicas, no sentido literal. Entre os precursores das máquinas mecânicas para cálculo aritmético encontramos Pascal e Leibniz. (Williams, 1997)

O MÉTODO DE NEWTON

Um dos métodos computacionais atualmente mais conhecidos para aproximar um zero de uma função é o chamado método de Newton. Vamos ver qual a origem desta atribuição.

Numa das suas obras, Newton (2004) pretende aproximar a solução da equação $y^3-2y-5=0$. Dado que tem já uma primeira aproximação, que é 2, "cuja diferença do verdadeiro valor da raiz é inferior à sua décima parte" (2004, p. 6), supõe que a solução é da forma $y=2+p$. Trata-se agora de determinar p. Para tal, podemos fazer a substituição na equação original e, após alguns cálculos, resulta para p a equação $p^3+6p^2+10p-1=0$. Neste ponto parece que não se ganhou nada, mas Newton faz uma observação fundamental: "e desprezando p^3+6p^2 em virtude da sua pequenez, restará $10p-1=0$, ou $p=0{,}1$ que é muito próximo do valor de p." (2004, p. 6)

A hipótese de que p é um número pequeno, cujas potências superiores a 1 são negligenciáveis, é decisivo para se simplificar a equação. Newton de seguida aplica o mesmo método à equação completa de p, escrevendo a sua solução como $p=0{,}1+q$. A equação para q fica então $(0{,}1+q)^3+6(0{,}1+q)^2+10(0{,}1+q)-1=0$, que é equivalente a $q^3+6{,}3\,q^2+11{,}23\,q+0{,}061=0$. Omitindo de novo os termos de grau superior a 1, obtém-se $11{,}23\,q+0{,}061=0$ cuja solução é aproximadamente -0,0054. O passo seguinte seria fazer $q=-0{,}0054+r$.

Note-se que as aproximações da raiz foram sendo sucessivamente $x_0=2$, $x_1=2{,}1$ e $x_2=2{,}0946$.

Este processo de Newton foi desenvolvido por diversos matemáticos, começando por J. Raphson. Progressivamente percebeu-se que o método podia ser generalizado a funções diferenciáveis. Entre aqueles que deram contributos para a formulação atual do método podemos encontrar Lagrange e Cauchy.

Atualmente este método expressa-se da seguinte forma: consideramos a função definida por $f(y)=y^3-2y-5$, cuja derivada é $f'(y)=3y^2-2$. Sendo $x_0=2$, define-se por recorrência a sucessão do método de Newton:

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}.$$

Este método tem uma interpretação geométrica interessante e os termos da sucessão coincidem com os que se obtêm aplicando o procedimento descrito por Newton (Gonçalves, 2014; Sousa, 2016). Prova-se também que, neste caso, esta sucessão converge efetivamente para a solução da equação.

Quando fazemos $f(y)=y^2-a$, cuja derivada é $f'(y)=2y$, obtemos:

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}=x_n-\frac{x_n^2-a}{2x_n}=\frac{1}{2}\left(x_n+\frac{a}{x_n}\right).$$

Isto é, o método de Herão é exatamente o método de Newton aplicado à resolução da equação x^2-a=0. Prova-se que o método de Herão converge para $\sqrt{a}$, qualquer que seja o valor inicial $x_0>0$, embora o número de iterações necessário até obter uma aproximação suficientemente boa (de acordo com um determinado critério) dependa do valor inicial usado.

A rapidez da convergência do método de Newton, quando se parte de um valor suficientemente próximo da raiz, e a possibilidade de se generalizar o método a espaços de mais do que uma dimensão, tornam este método especialmente importante, quer do ponto de vista teórico, quer do ponto de vista prático.

Temos também aqui um exemplo de abstração. O método de Herão começa por ser pensado para resolver um problema: calcular raízes quadradas. Newton propõe um método para aproximar zeros de polinómios, que inclui o método de Herão como um caso particular. Apresentámos depois uma formulação atual do método de Newton aplicável a funções reais de variável real, desde que deriváveis, o que engloba os polinómios. Mas o método pode ainda ser generalizado a funções complexas e a funções em espaços vetoriais de dimensão finita ou infinita. Em cada caso generalizamos o método a estruturas mais abstratas.

OS CÁLCULOS DE GAUSS

Quando se fazem medições há necessariamente erros. Determinar os parâmetros da órbita de um objeto no sistema solar, a partir de múltiplas observações com os inevitáveis erros, gera uma situação em que temos um sistema de equações sobredeterminado. Para obter um valor mais provável para os parâmetros, Gauss e Legendre propuseram, de forma independente, no início do século XIX, o método dos mínimos quadrados. Gauss terá aplicado o método na determinação da órbita do planeta Ceres, ainda antes de o publicar, e aplicou-o depois em trabalhos de geodesia. Para obter a solução dos problemas de mínimos quadrados era necessário resolver sistemas de equações lineares. Até então a resolução de sistemas lineares era um problema académico. Com as necessidades geradas pelos mínimos quadrados, passou a ser um problema economicamente importante e Gauss, que era um "calculador incorrigível" (Grcar, 2011a, p. 785), desenvolveu métodos práticos que foram adotados por quem fazia trabalhos de geodesia. Hoje chamamos eliminação de Gauss a um método de resolução de sistemas lineares que é fruto de muitas contribuições, ao longo de vários séculos, mas para o qual Gauss também contribuiu (Grcar, 2011a).

Na eliminação de Gauss a resolução do sistema é decomposta em duas partes. Na primeira parte o sistema é transformado, através de operações sobre linhas, num sistema equivalente, mas que permite uma resolução mais simples. Na segunda parte este sistema mais simples é resolvido, da última para a primeira equação.

SEBASTIÃO E SILVA

Em 1940, a revista Portugaliae Mathematica publica um artigo com o título "Da resolução numérica de equações algébricas" da autoria de um jovem matemático, José Sebastião e Silva. Neste artigo o autor discute a convergência de métodos numéricos, nomeadamente do método de Newton (que chama de método de Newton-Fourier ou da tangente) e de generalizações de métodos conhecidos (Sebastião e Silva, 1940; Sousa, 2016). No ano seguinte Sebastião e Silva publica na mesma revista um novo trabalho, desta vez em francês, com o título "Sur une méthode d'approximation semblable a celle de Graeffe" (Sebastião e Silva, 1941). Este artigo veio a ter repercussão internacional algumas décadas mais tarde, sendo o método de Sebastião e Silva generalizado por figuras importantes da análise numérica como Householder e Stewart (Householder & Stewart, 1971; Householder, 1971; McNamee & Pan, 2012).

Em 1947, Sebastião e Silva traduz do italiano e publica na Gazeta de Matemática, com o título "A máquina calculadora eletrónica" um artigo (Kennedy, 1947) que lhe tinha chegado através do Professor Mauro Picone, de Roma, e que estava num relatório do "United States Information Service". Trata-se de uma descrição do primeiro computador eletrónico digital programável, o ENIAC. O original deste artigo tinha sido publicado no The New York Times, em fevereiro de 1946, com o título "Electronic Computer Flashes Answers, May Speed Engineering" (Kennedy, 1946). Este artigo dá-nos uma perspetiva rica sobre o ambiente que envolveu o aparecimento dos computadores eletrónicos digitais programáveis nos Estados Unidos.

Na nota que junta ao artigo, Sebastião e Silva apresenta também algumas observações interessantes:

> Abundam assim, na Análise clássica, as *elegantes* demonstrações de existência, que bem magra satisfação podem dar a quem não queira permanecer eternamente numa atitude contemplativa.
>
> Todavia a máquina calculadora eletrónica vem modificar enormemente este estado de coisas, multiplicando por mil, segundo se diz, as possibilidades humanas de efetuar cálculos numéricos. Deste modo, muitos procedimentos considerados até hoje como inexequíveis passam a ter viabilidade — o que vem abrir à Ciência e à Técnica perspetivas inimagináveis. (...)
>
> Uma última conclusão nos parece lícito tirar daqui: a necessidade premente de arejar os nossos métodos e programas do ensino, tornando-os adequados ao espírito da época. (Sebastião e Silva, 1947, p. 4)

MATEMÁTICOS E COMPUTADORES ELETRÓNICOS

Entre 1935 e 1945 houve diversas experiências na construção de computadores digitais eletrónicos e com o esforço de guerra havia grandes necessidades de cálculo. Cada nova peça de artilharia precisava, para ser usada, de tabelas de tiro que implicavam cálculos extensos e demorados. Isto é realçado no artigo de Kennedy (1947). Este artigo destaca justamente Mauchly e Eckert pelo seu papel na conceção do ENIAC. Entretanto um dos mais notáveis matemáticos do século XX, John von Neumann, soube da construção do ENIAC e envolveu-se nas discussões sobre

um novo computador que estava a ser planeado, o EDVAC. Von Neumann redigiu um relatório sobre os planos do EDVAC (von Neumann, 1993), que teve uma enorme circulação, e desde então deu-se o nome de arquitetura de von Neumann a um tipo de organização de um computador eletrónico digital programável em que, entre outras características, o programa é mantido na mesma memória que os dados e pode alterar-se a si próprio. (Williams, 1997). Atualmente quase todos os computadores de utilização geral têm este tipo de arquitetura.

Outro matemático incontornável na criação e aplicação dos computadores é Alan Turing. Em 1937, Turing publicou um artigo na revista Proceedings of the London Mathematical Society, em que apresenta uma máquina teórica (máquina de Turing), para resolver problemas teóricos, mas cujo modelo de computação ainda hoje é usado para estudar o que se pode esperar que um computador possa calcular. Durante a Segunda Guerra Mundial, Turing trabalhou na criptanálise das cifras alemãs e participou no desenvolvimento de computadores eletrónicos.

Em 1950 Turing propôs um teste (que ficou com o seu nome) para avaliar a capacidade de uma máquina programada imitar a inteligência humana. Uma pessoa comunicaria com a máquina sem saber se dialogava com uma máquina ou com uma pessoa. Se a pessoa ficasse convencida de que tinha dialogado com outra pessoa, a máquina passaria no teste.

No desenvolvimento inicial dos computadores eletrónicos programáveis encontramos assim, nos EUA e no Reino Unido, dois dos mais notáveis matemáticos do século XX.

OS COMPUTADORES PARA O CÁLCULO CIENTÍFICO E AS CIÊNCIAS DA COMPUTAÇÃO

Uma das motivações para a construção de computadores eletrónicos digitais e uma das primeiras aplicações foi o cálculo científico. Em 1947, von Neumann, em colaboração com Goldstine, publica no Bulletin of the American Mathematical Society, um artigo com o título "Numerical Inverting of Matrices of High Order", que foi considerado "o primeiro artigo moderno em análise numérica" (Grcar, 2011b).

Ao longo dos anos 50 e 60 do século XX os matemáticos são considerados especialmente aptos para trabalhar com os computadores, mas começa a questionar-se o que distingue a matemática das ciências associadas ao desenvolvimento e utilização dos computadores.

Georges Forsythe foi um matemático que se dedicou à análise numérica e à informática e que desempenhou um papel significativo para o estabelecimento da Ciência da Computação como uma disciplina reconhecida (Knuth, 1972).

Dirigindo-se a matemáticos, Forsythe (1968, p. 454) começava com uma questão: "O que é a ciência da computação, afinal?" Na época as universidades americanas estavam a criar departamentos de informática e era preciso explicar o que distinguia esta nova área. Neste artigo Forsythe escreve:

> As aquisições mais valiosas numa educação científica ou técnica são as ferramentas mentais de uso geral que continuam úteis ao longo da vida. Considero a linguagem natural e a matemática as mais importantes dessas ferramentas, sendo a ciência da computação a terceira. A matemática que ensinais chega a uma aplicação efetiva em grande medida através da computação digital, pelo que vós e os vossos estudantes precisais de saber alguma ciência da computação. A aprendizagem da matemática e da ciência da computação em conjunto tem vantagens pedagógicas, pois os conceitos básicos de cada uma reforçam a aprendizagem da outra (e.g., os conceitos de função em matemática e procedure em Algol 60). (1968, p. 456-457)

Donald Knuth concluiu um doutoramento em matemática no California Institute of Technology em 1963, mas foi paralelamente trabalhando com computadores e aprofundando o seu interesse pela computação. Dedicou-se ao estudo dos algoritmos e das linguagens de programação, sendo o autor de uma obra de referência em vários volumes: "The Art of Computer Programming". Estes trabalhos de Knuth valeram-lhe ser galardoado em 1974 com o prémio Turing da Association for Computing Machinery, um prémio com um prestígio na área da computação análogo aos prémios Nobel noutras áreas. Knuth é conhecido também por ter criado o sistema TeX (no qual se baseia o LaTeX) para composição tipográfica com fórmulas.

Em 1974, Knuth referia o valor pedagógico de uma aproximação algorítmica, também na matemática: "Na verdade, uma pessoa não entende realmente uma coisa até ser capaz de a ensinar a um computador, i.e., expressá-la como um algoritmo." (1974, p. 327) e dá um exemplo: "Os linguistas pensavam que compreendiam as línguas, até que tentaram explicá-las aos computadores; rapidamente descobriram o quanto faltava ainda aprender." (1974, p. 327)

MATEMÁTICA E COMPUTAÇÃO

O prémio Turing de 1972 foi atribuído a E. Dijkstra. Em 1974 este autor publicava um artigo com o título "A programação como uma disciplina de natureza matemática" (Dijkstra, 1974), onde apresenta três características do trabalho em matemática: asserções muito precisas, por comparação com outras áreas, asserções aplicáveis a grandes classes de instâncias, e possibilidade de asserções com um grau de confiança muito elevado. Dijkstra encontra estas características no trabalho do programador e por isso o classifica como de natureza matemática. Além disso, discute os esforços que já então se faziam para a demonstração formal da correção de programas.

O prémio Abel de 2021, atribuído a László Lovász e a Avi Wigderson, "celebra a união da matemática com a ciência da computação", pois foi atribuído

> pelas suas contribuições fundamentais para a ciência da computação teórica e matemática discreta, e pelo seu papel de liderança em transformá-las em áreas centrais da matemática moderna (Castelvecchi, 2021)

Num trabalho de que Wigderson é coautor, dedicado à Teoria da Computação, pode ler-se:

> Algoritmos eficientes quase só são descobertos se existir uma estrutura matemática tangível. Esta conexão já beneficiou o progresso matemático em muitas áreas, como Teoria de Números, Álgebra, Teoria de Grupos e Combinatória, onde por um lado existia a necessidade de algoritmos eficientes e, por outro lado, a sua procura gerou resultados estruturais interessantes por si próprios. (Goldreich & Wigderson, 2009)

O PENSAMENTO COMPUTACIONAL

Seymour Papert tinha dois doutoramentos em matemática quando em 1963 começou a trabalhar no MIT. Aí desenvolveu trabalhos em várias áreas, incluindo a inteligência artificial e os processos de aprendizagem das crianças, e criou a linguagem LOGO (Brittanica). Em 1980, Papert usa a expressão "pensamento procedimental" ("procedural thinking" no original) e escreve:

> Neste livro tenho defendido claramente que o pensamento procedimental é uma ferramenta intelectual poderosa e até sugeri fazer uma analogia de si mesmo com um computador, como uma estratégia para o efeito. (1980, p. 155)

Papert também usa uma vez a expressão "pensamento computacional" (1980, p. 182), mas sem lhe dar destaque. "Computational thinking" não aparece no índice remissivo, enquanto, por exemplo, "Combinatorial thinking", aparece (1980, p. 225) e aparecem, entre outras referências, "Thinking, combinatorial" e "Thinking, procedural" (1980, p. 229).

Em 2006, Jeannette Wing dá um novo destaque ao termo "pensamento computacional", tentando defini-lo e realçando a importância de generalizar as formas de pensar associadas à ciência da computação. Desde então tem aumentado a consciência da necessidade de enriquecer o ensino com o que as ciências da computação podem oferecer, mas não há uma definição estável de pensamento computacional.

Um dos participantes nos debates sobre a caracterização do pensamento computacional foi A. Aho, prémio Turing em 2020, que em 2011 destaca o papel dos modelos de computação: "As abstrações matemáticas chamadas modelos de computação estão no coração da computação e do pensamento computacional." (Aho, 2011)

Sobre o pensamento computacional podem ser encontradas várias perspetivas em Ramos & Espadeiro (2014). Beecher (2017) discute detalhadamente as características do pensamento computacional e, depois de um capítulo de iniciação ao Python, concretiza os princípios num contexto de desenvolvimento de software. Uma perspetiva mais histórica pode ser encontrada em Tedre & Denning (2016) e em Denning & Tedre (2019). Um artigo também recente, com uma panorâmica interessante sobre o pensamento computacional, é Nardelli (2019).

Não temos aqui o objetivo de explorar as várias definições de pensamento computacional, mas há alguns aspetos que são recorrentes nas várias definições e que podemos identificar na matemática e/ou nos exemplos históricos que referimos antes. Alguns destes exemplos permitem o aprofundamento do pensamento computacional num contexto perfeitamente natural para a matemática.

ALGORITMOS E PENSAMENTO LÓGICO

Já vimos que, de modo informal, os algoritmos sempre estiveram presentes na matemática.

Nos anos 30 do século XX, a propósito do estudo de questões dos fundamentos da matemática, os matemáticos sentiam a necessidade de pensar de forma rigorosa o que poderia ser efetivamente calculado (Chabert et al., 1994). O artigo de Turing (1937) vem nesta linha, começando com a frase: "Os números "computáveis" podem ser resumidamente descritos como os números reais cujas expressões decimais são calculáveis por meios finitos." (1937, p. 230) Na mesma página, refere: "Os números computáveis não incluem, no entanto, todos os números definíveis, e é dado um exemplo de um número definível que não é computável." Aqui a noção de algoritmo pode ser definida de forma rigorosa, associada a um modelo de máquina computacional.

Os algoritmos incluem naturalmente testes lógicos e uma evolução diferenciada em função da avaliação de condições lógicas, pelo que é essencial o uso do cálculo lógico.

ABSTRAÇÃO

Na modelação matemática está presente a abstração quando se escolhem apenas os aspetos da realidade que são essenciais para o problema em estudo, ignorando-se os restantes. Para Copérnico o objetivo era descrever as posições dos astros, determinadas em termos numéricos, sem haver necessidade de discutir a realidade física. Outro exemplo de abstração é a questão das pontes de Konigsberg, que Euler reduziu aos grafos.

Falamos também de abstração em matemática quando estudamos, em simultâneo, diferentes objetos que têm certas propriedades em comum. Quando se estudam as propriedades dos grupos em geral, os resultados aplicam-se aos números reais com a adição, às matrizes reais 2×2 invertíveis com o produto matricial ou às isometrias do plano com a operação de composição. Já vimos, no caso do método de Newton, a generalização de métodos a contextos progressivamente mais abstratos.

DECOMPOSIÇÃO E RECONHECIMENTO DE PADRÕES

Dividir para reinar é uma estratégia conhecida para todo o tipo de situações. Decompor um problema em partes é uma forma não só de resolver de cada vez um problema mais simples como permite verificar cuidadosamente se cada parte é corretamente resolvida, restando depois assegurar a integração correta das partes. Esta abordagem foi detalhada por Descartes no seu Discurso do Método (Descartes, 2007).

Na matemática esta abordagem surge naturalmente com a estrutura demonstrativa. Partindo de um pequeno número de axiomas é possível ir demonstrando progressivamente resultados mais complexos, de modo que na solução de determinados problemas usamos com confiança teoremas poderosos, que exprimem resultados muito elaborados. Definir o método de Newton usando a derivada de uma função incorpora toda a

análise real que está por trás do estudo do cálculo diferencial, sem ser necessário rever e demonstrar tudo de novo.

Quando se trata de resolver problemas com um computador esta estratégia é tão necessária que as linguagens de programação incluem naturalmente estruturas para facilitar a decomposição. Uma das mais simples é a noção de função numa linguagem, pois permite separar o funcionamento interno de uma função e verificar cuidadosamente a sua correção, sem interferir com os programas que podem vir a usar essa função.

Esta decomposição conjuga-se naturalmente com a deteção de situações em que há um padrão: certas "peças" mais pequenas são úteis em várias situações diferentes.

DEPURAÇÃO

Um dos aspetos que pode ser desmotivador numa primeira abordagem à programação é a quantidade de erros que se produzem. Desde a escrita inicial das primeiras linhas de código até à finalização de um projeto, os erros e as respetivas mensagens, geradas automaticamente, são inevitáveis.

Quando se está a começar e se está habituado a uma perspetiva da matemática em que os erros não devem ter lugar, estar a receber uma mensagem automática com relatos de erros a cada minuto pode ser profundamente desmotivador. Mas é importante saber-se que isso é absolutamente normal. Os erros de sintaxe, frequentes quando se está a começar ou quando se muda entre duas linguagens (mesmo que o programador tenha prática com as duas), vão desaparecendo com a persistência. Mas os erros de semântica, em que a máquina faz exatamente o que lhe ordenámos, mas não o que realmente queríamos, fazem parte do processo. Esta característica das máquinas, que consiste em fazerem exatamente o que lhes é ordenado, parece irritante ao início, mas é precisamente o que as torna úteis por serem muito fiáveis (Dijkstra, 1974).

Entretanto, desde que existem computadores programáveis, houve necessidade de rever os programas para detetar e remover todo o tipo de erros. Esta atividade tomou um nome em inglês, "debugging", tendo-se adotado em português a palavra depuração. Mesmo quando escrevemos pequenos programas, o mais normal é que haja erros e convém sempre fazer alguns testes para verificar se os resultados obtidos estão de acordo com as nossas intenções.

Apesar de desagradáveis, os erros podem ser ultrapassados com persistência e eventualmente com a cooperação de alguém mais experiente. O acesso à net permite colocar qualquer questão num motor de busca e na maior parte das vezes encontram-se respostas às nossas questões ou pelo menos informação com pistas que nos permite novas abordagens.

Como em muitas atividades humanas, as dificuldades do percurso aumentam a satisfação depois de atingido o sucesso.

PARA PASSAR À PRÁTICA

Para passarmos a uma perspetiva mais prática, sugerimos aqui alguns programas muito simples, que permitem experimentar o Python usando apenas um browser. Há sites online que correm programas em Python, como https://www.programiz.com/python-programming/online-compiler/ que usámos para este artigo. Neste caso é suficiente ter um telemóvel ou computador com um *browser*..

O MÉTODO DE NEWTON

O algoritmo de Herão para a raiz quadrada pode ser implementado pelo seguinte programa Python, que calcula seis termos da sucessão, além do inicial:

```
a=720
x=27
print(x)
for k in range(6):
    x=(x+a/x)/2
    print(x)
```

No compilador *online Programiz* apaga-se o código que inicialmente aparece do lado esquerdo e copia-se o programa. Clicando no botão Run, obtém-se do lado direito o resultado:

```
27
26.833333333333336
26.832815734989648
26.832815729997478
26.832815729997478
26.832815729997478
26.832815729997478
```

Convém notar aqui que em Python os espaços no início das linhas são usados para caracterizar o âmbito das instruções, pelo que devem ser cuidadosamente verificados. Se numa das linhas usarmos um espaço a mais, como no exemplo:

```
a=720
x=27
print(x)
for k in range(6):
  x=(x+a/x)/2
   print(x)
```

em vez dos resultados obtemos a mensagem:

```
File "<string>", line 6
    print(x)
    ^
IndentationError: unexpected indent
```

Para calcular a raiz quadrada de outro número positivo, atribui-se esse número à variável a, na primeira linha. A aproximação inicial, atribuída a x na segunda linha, pode ser qualquer número positivo. Uma possibilidade é partir do valor 1. Por exemplo, o programa:

```
a=750
x=1
print(x)
for k in range(10):
    x=(x+a/x)/2
    print(x)
```

produz na janela de saída:

```
1
375.5
188.74866844207722
96.36110320380936
52.07216333664015
33.23762613989639
27.901207258439086
27.390882271161686
27.38612828788248
27.38612787525831
27.386127875258303
```

Considerando agora o método de Newton em geral, se pretendermos apenas um cálculo específico, podemos ser bastante económicos nas linhas de código envolvidas. Consideremos, por exemplo, a equação que Newton usou como exemplo.

Um programa simples seria

```
x=2
print(x)
for k in range(6):
    x=x-(x**3-2*x-5)/(3*x**2-2)
    print(x)
```

com o resultado

```
2
2.1
2.094568121104185
2.094551481698199
2.0945514815423265
2.0945514815423265
2.0945514815423265
```

Neste ponto podemos fazer várias generalizações. Podemos definir antecipadamente como funções Python a função f e a sua derivada, a que chamaremos df. Isto permite que, se usarmos várias vezes a função f, não tenhamos que reescrever a sua expressão (o que, além de moroso, aumenta a probabilidade de erro, seja por esquecimento de mudar alguma ocorrência seja por erro ao escrever ou copiar a expressão).

Também o critério de paragem, nos nossos programas anteriores, é pouco flexível. Escolhemos um número fixo de iterações, independentemente do desenrolar dos cálculos. Ora uma das vantagens da programação é podermos escolher o que fazer em função da avaliação dos resultados e da combinação lógica de diversas avaliações.

Vamos então adotar como critério de paragem a ocorrência de pelo menos uma de duas situações: a diferença entre um valor calculado e o anterior é inferior a uma tolerância dada (tol) ou ultrapassamos um número máximo de iterações que consideramos aceitável (maxit). Para controlarmos o número de iterações precisamos de ter um contador (n) e para avaliar a diferença entre duas iterações é necessário guardar a iteração anterior numa variável auxiliar (x0) enquanto a nova iteração fica noutra variável (x1).

```
def f(x):
    return x**3-2*x-5

def df(x):
    return 3*x**2-2

tol=1e-10
maxit=100

x0=2
print(x0)
x1=x0-f(x0)/df(x0)
print(x1)
n=1
while abs(x1-x0)>=tol and n<=maxit:
    x0=x1
    x1=x1-f(x1)/df(x1)
    print(x1)
    n=n+1
```

A vantagem deste programa é que podemos facilmente identificar o que mudar para aplicarmos o método de Newton em novas situações. Este processo de generalização poderia ser levado mais longe, criando uma nova função Python que receberia como argumentos todos os dados (função e sua derivada, tolerância, máximo de iterações e valor inicial) e devolveria apenas a última iterada calculada e o número de iterações efetuadas. Uma variante seria a função devolver apenas uma lista com todas as iteradas calculadas.

CONCLUSÃO

O cálculo efetivo, primeiro por humanos e depois por máquinas, esteve sempre presente na matemática. Muitos dos melhores matemáticos calcularam ou desenvolveram métodos de cálculo. Foi em grande parte a necessidade de cálculo que levou ao desenvolvimento dos computadores, que foram inicialmente pensados a partir da matemática. Com o desenvolvimento dos computadores e o alargamento das suas aplicações, a informática autonomizou-se a partir da matemática e da engenharia eletrotécnica.

Já desde os anos 60 do século passado era evidente o interesse dos computadores para o ensino/aprendizagem da matemática, mas as dificuldades no acesso aos computadores não permitiam o seu uso generalizado. Atualmente qualquer dispositivo de acesso à Internet permite que se executem programas em Python e existem ambientes sofisticados gratuitos e de fácil instalação para desenvolvimento de programas em Python nos computadores pessoais.

Por outro lado, quando hoje se fala de pensamento computacional, são muitas as sobreposições com o pensamento matemático. Destacámos aqui alguns aspetos comuns: pensamento algorítmico e lógico, abstração, decomposição e depuração, que engloba a prevenção, deteção e correção de erros.

A execução efetiva dos programas é determinante para que seja interessante o seu uso generalizado na matemática e o desenvolvimento do próprio pensamento computacional não pode prescindir da componente de programação e execução por

uma máquina (Denning, Tedre & Yongpradit, 2017; Nardelli, 2019). Esta componente traz alguns desafios práticos, mas é possível ultrapassá-los tendo presente que se trata de dificuldades normais, comuns a todos os que trabalham com computadores.

O desenvolvimento do pensamento computacional permite uma visão mais profunda do papel da matemática na sociedade atual e abre portas para áreas em desenvolvimento na matemática.

Referências

Aho, A. V. (2011). Ubiquity symposium: Computation and computational thinking. Ubiquity, 2011(January). http://ubiquity.acm.org/article.cfm?id=1922682

Beecher, K. (2017). Computational Thinking - A beginner's guide to problem-solving and programming. BCS.

Brittanica. Seymour Papert. https://www.britannica.com/biography/Seymour-Papert

Castelvecchi, D. (2021). Abel prize celebrates union of mathematics and computer science. Nature, https://www.nature.com/articles/d41586-021-00694-9

Chabert, J.-L., Barbin, É., Guillemot, M., Michel-Pajus, A., Borowczyk, J., Djebbar, A., & Martzloff, J.-C. (1994). Histoire d'algorithmes – Du caillou à la pouce. Belin.

Copérnico, N. (2014). As revoluções dos orbes celestes. Fundação Calouste Gulbenkian. https://gulbenkian.pt/publication/as-revolucoes-dos-orbes-celestes/

Denning, P. J., & Tedre, M. (2019). Computational Thinking. The MIT Press.

Denning, P. J., Tedre, M. & Yongpradit, P. (2017). Misconceptions About Computer Science. Communications of the ACM, 60(3), 31-33.

Descartes, R. (2007). Discurso do método. Intratext. http://www.intratext.com/IXT/POR0305/_P2.HTM

Dijkstra, E. W. (1974). Programming as a Discipline of Mathematical Nature. The American Mathematical Monthly, 81(6), 608-612. https://www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD361.html

Fitzpatrick, R. (2008). Euclid's Elements of Geometry. https://farside.ph.utexas.edu/books/Euclid/Elements.pdf

Forsythe, G. E. (1968). What to do Till the Computer Scientist Comes. The American Mathematical Monthly, 75(5), 454-462, https://stacks.stanford.edu/file/druid:rz632tc5340/rz632tc5340.pdf

Goldreich, O. & Wigderson, A. (2009). The Theory of Computing: A Scientific Perspective. https://www.wisdom.weizmann.ac.il/~oded/PDF/toc-sp2.pdf

Gonçalves, R. A. (2014). Utilização de métodos numéricos na resolução de equações e perspetivas de integração curricular no Ensino Secundário. Universidade do Minho. http://repositorium.sdum.uminho.pt/handle/1822/34723

Grcar, J. (2011a). Mathematicians of Gaussian Elimination. Notices of the AMS, 58(6), 782-792. https://www.ams.org/notices/201106/rtx110600782p.pdf

Grcar, J. (2011b). John von Neumann's Analysis of Gaussian Elimination and the Origins of Modern Numerical Analysis. SIAM Review, 53(4), 607-682.

Householder, A. S. & Stewart, G. W. (1971). The numerical factorization of a polynomial. SIAM Review, 13(1), 38-46.

Householder, A. S. (1971). Generalizations of an Algorithm of Sebastião e Silva. Numerische Mathematik, 16, 375–382.

Katz, V. (1992). A History of Mathematics. HarperCollins.

Kennedy Jr., T. R. (1946). Electronic Computer Flashes Answers, May Speed Engineering. The New York Times, 15/2/1946. https://www.computerhistory.org/revolution/birth-of-the-computer/4/78/323

Kennedy Jr., T. R. (1947). A máquina calculadora electrónica. Gazeta de Matemática, 32, 1-3. https://gazeta.spm.pt/getArtigo?gid=1111

Knuth, D. E. (1972). George Forsythe and the Development of Computer Science. Communications of the ACM, 15(8), 721-727. http://coweb.cc.gatech.edu/guzdial/uploads/46/knuth-on-Forsythe-1972-CACM.pdf

Knuth, D. E. (1974). Computer Science and its Relation to Mathematics. The American Mathematical Monthly, 81, 323-343, https://www.cs.ou.edu/~diochnos/about/KnuthCSMaths.pdf

Loxodrómias e Espirais (2012). http://formas-formulas.fc.ul.pt/interactive/loxo/pt/index_pt.html

McNamee, J. M. & Pan, V. Y. (2012). Efficient polynomial root-refiners: A survey and new record efficiency estimates. Computers and Mathematics with Applications, 63, 239-254.

Nardelli, E. (2019). Do We Really Need Computational Thinking? Communications of the ACM, 62(2), 32-35.

Newton, I. (2004). O método das fluxões e das séries infinitas. Associação de Professores de Matemática e Editorial Prometeu.

Nunes, P. (2008). Obras – Vol. IV. Fundação Calouste Gulbenkian.

Papert, S. (1980). Mindstorms - Children, Computers, and Powerful Ideas. Basic Books.

Queiró, J. F. (2002). Pedro Nunes e as Linhas de Rumo. Gazeta de Matemática, 143, 42-47. https://gazeta.spm.pt/getArtigo?gid=56

Ramos, J. L. & Espadeiro, R. G. (2014). Os futuros professores e os professores do futuro. Os desafios da introdução ao pensamento computacional na escola, no currículo e na aprendizagem. Educação, Formação & Tecnologias, 7(2), 4-25. https://dialnet.unirioja.es/servlet/articulo?codigo=5112361

Rodrigues, J. F. (2017). Loxodrómias e Espirais – I. Revista de Ciência Elementar, 5(2):022. https://rce.casadasciencias.org/rceapp/art/2017/022/

Sebastião e Silva, J. (1940). Da resolução numérica de equações algébricas. Portugaliae Mathematica, 1, 303-332. https://purl.pt/404

Sebastião e Silva, J. (1941). Sur une méthode d'approximation semblable a celle de Graeffe. Portugaliae Mathematica, 2, 271–279. Republicado em Sebastião e Silva, J. (1985). Obras de José Sebastião e Silva, I, INIC, 77-85.

Sebastião e Silva, J. (1947). Nota. Gazeta de Matemática, 32, 3-4. https://gazeta.spm.pt/getArtigo?gid=1111

Sousa, L. (2016). Sobre um método proposto por Sebastião e Silva para o cálculo de raízes de uma equação algébrica. Universidade do Minho. http://repositorium.sdum.uminho.pt/handle/1822/45650

Tedre, M. & Denning, P. J. (2016). The Long Quest for Computational Thinking. Proceedings of the 16th Koli Calling Conference on Computing Education Research, 120-129.

Turing, A. M. (1937). On Computable Numbers, with an Application to the Entscheidungsproblem. Proceedings of the London Mathematical Society, s2-42(1), 230–265. https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf

von Neumann, J. (1993). First Draft of a Report on the EDVAC. IEEE Annals of the History of Computing, 15(4), 27-75. http://web.mit.edu/STS.035/www/PDFs/edvac.pdf

Williams, M. R. (1997). A History of Computing Technology. 2nd ed. IEEE Computer Society Press.

Wing, J. (2006). Computational Thinking. Communications of the ACM, 49(3), 33-35.

Carlos Albuquerque
Ciências - ULisboa