

Manual de Cálculo Numérico Aplicado

Prof. Doutor Rafael Gabler Gontijo

19 de agosto de 2025



SOLUÇÕES EM COMPUTAÇÃO CIENTÍFICA

Quando pensamos na palavra ecossistema, imaginamos uma estrutura viva, interdependente, onde diferentes formas de existência sujeitas a diferentes níveis de consciência interagem entre si obedecendo uma hierarquia circular, colaborando, cada qual dentro de suas potencialidades, de forma dinâmica a serviço da manutenção do sistema. Um ecossistema é, por definição, uma estrutura bem arquitetada, em que cooperação, hierarquia e mutualismo coexistem — permitindo que cada agente evolua a partir da conexão com os demais. É exatamente com esse espírito que nasce a L2C – Soluções em Computação Científica.

A L2C é um ecossistema científico, tecnológico e educacional, voltado à construção de uma rede colaborativa entre estudantes, engenheiros, cientistas, programadores e clientes do setor produtivo que precisam de soluções computacionais para problemas de Engenharia. Essa rede tem como propósito não apenas entregar soluções baseadas em simulação computacional, mas promover um processo de emancipação intelectual de todos os seus agentes, com base na colaboração, na transparência e no rigor científico.

Nesse cenário, um dos pilares centrais e diferenciais da L2C é a adoção consciente da filosofia do código aberto, não apenas como um recurso técnico, mas como uma postura ética diante da construção e circulação do conhecimento gerado dentro desse ecossistema.

Essa filosofia tem raízes históricas que remontam ao início da era da computação, quando os softwares desenvolvidos eram concebidos para rodar em máquinas específicas. Naquele tempo, os programas só podiam ser executados nas máquinas para as quais eram projetados. Não havia ainda portabilidade de software entre diferentes máquinas. O cliente, geralmente uma empresa, universidade, órgão governamental ou centro de pesquisa, comprava uma máquina cara e os programas vinham juntos. Não fazia sentido existir na época uma indústria de software separada da indústria de hardware. Os códigos associados aos programas utilizados nessas máquinas circulavam de forma natural e colaborativa entre comunidades de programadores que aprendiam e evoluíam juntos.

Com o avanço da indústria da computação veio a portabilidade de software entre diferentes plataformas. Com a portabilidade surge então a possibilidade de vender o software como um produto separado do hardware, marcando o início da indústria de software proprietário e do modelo de código fechado sob o qual a maioria de nós aprendeu a interagir com computadores. Nesse novo paradigma, o conhecimento computacional passa a ser encapsulado e vendido como ativo restrito e com isso o usuário de computador vai se distanciando cada vez mais do desenvolvedor. Esse modelo populariza o uso de computadores por usuários domésticos, mas ao mesmo tempo torna os usuários reféns de arquiteturas de software engessadas, definidas por um grupo pequeno de programadores.

A L2C visa resgatar, atualizar e reposicionar esse espírito colaborativo original, inserindo-o em um novo contexto mais amadurecido, técnico e estruturado, voltado ao universo da modelagem matemática e simulação computacional de problemas práticos de Engenharia.

A L2C não se baseia apenas na entrega de soluções computacionais para problemas de engenharia. Além disso, a empresa busca atuar como ponte, estabelecendo conexões entre clientes e colaboradores dentro de um ecossistema de desenvolvimento e comparti-

lhamento de soluções computacionais baseadas em código aberto para problemas práticos utilizando modelagem matemática e construção de algoritmos autorais juntamente com implementações em cima de códigos consagrados pela comunidade científica, baseados na filosofia open-source.

Nossa premissa fundamental é a de que a ciência precisa ser transparente, auditável e compartilhável para cumprir seu papel social. Nesse sentido, as linhas de atuação da empresa se baseiam no tripé: consultoria, desenvolvimento de software como serviço e capacitação e treinamento. Todas pautadas no princípio do código aberto como caminho ético de construção de um ecossistema saudável.

O ambiente de capacitação proposto pela empresa é baseado na oferta de cursos presenciais e remotos em diferentes formatos e sobre os mais diversos assuntos dentro do universo da computação científica. Criamos também cursos sob medida para aplicações específicas de acordo com a necessidade de clientes do setor tecnológico-industrial. Oferecemos, portanto, não apenas uma solução computacional, mas um caminho de transformação técnica, organizacional e intelectual — para que nossos clientes evoluam juntos com a ferramenta que utilizam.

Perfil do fundador

A L2C foi fundada pelo Prof. Rafael Gabler Gontijo, Engenheiro Mecânico (CREA 14931/D-DF), Mestre e Doutor em Ciências Mecânicas, cientista e professor de Engenharia Mecânica com mais de vinte anos de experiência em simulação computacional, modelagem numérica e pesquisa científica em Mecânica dos Fluidos, Eletromagnetismo e Transferência de Calor. Ao longo da sua trajetória acadêmica e profissional, sempre atuou na interface entre ciência, tecnologia e educação, buscando transformar conhecimento científico em soluções acessíveis e eficientes.

Ao longo de sua carreira como cientista, coordenou diversos projetos de pesquisa, publicou dezenas de artigos científicos em revistas internacionais de alto fator de impacto em sua área de pesquisa, e formou Mestres e Doutores. Lecionou disciplinas em cursos de graduação e pós-graduação tanto na UnB quanto na Unicamp, consolidando sua atuação como pesquisador e educador. É também bolsista de produtividade em pesquisa do CNPq, reconhecimento concedido a pesquisadores com produção científica destacada no Brasil.

Apaixonado por ensino, inovação e pelo potencial do software livre, Rafael também é programador e criador de conteúdos educacionais voltados à formação de estudantes e profissionais em áreas técnicas, mantendo um canal no YouTube onde disponibiliza cursos inteiros de graduação e pós-graduação de maneira gratuita. O canal conta com milhares de inscritos. A L2C nasce como uma extensão natural desse percurso: uma empresa comprometida com a excelência técnica, o compartilhamento de saberes e o desenvolvimento de soluções científicas aliadas à emancipação intelectual de seus clientes e colaboradores.

Sumário

| | | |
|----------|---|-----------|
| 1 | Introdução, história, generalidades e motivação | 6 |
| 1.1 | O que veremos nesse curso? | 6 |
| 1.2 | Um pouco de história | 7 |
| 1.3 | Como funciona um computador? | 13 |
| 1.3.1 | Portas lógicas e álgebra Booleana | 16 |
| 1.3.2 | CPU e memória | 16 |
| 1.3.3 | O formato IEEE 754 | 17 |
| 1.3.4 | Um pouco sobre compiladores | 19 |
| 1.4 | Por que precisamos de métodos numéricos? | 20 |
| 1.5 | A importância da validação de uma solução numérica | 24 |
| 1.6 | Tarefa inicial para aquecer os motores | 25 |
| 2 | Raízes de equações | 27 |
| 2.1 | O método da biseção | 27 |
| 2.2 | O método da falsa posição | 29 |
| 2.2.1 | Exercícios: biseção e falsa posição | 29 |
| 2.3 | O método da falsa posição modificado | 30 |
| 2.4 | O método de Newton-Raphson | 31 |
| 2.5 | O método da secante e da secante modificada | 32 |
| 2.6 | Raízes múltiplas | 33 |
| 2.6.1 | Exercício: Método de Newton-Raphson para raízes múltiplas | 34 |
| 2.7 | Métodos especiais para polinômios e aplicáveis à raízes complexas | 34 |
| 2.7.1 | O método de Müller | 35 |
| 2.7.2 | Programa para casa: método de Müller × método da secante | 37 |
| 2.7.3 | O método de Bairstow | 38 |
| 2.7.4 | Proposta de programa: método de Bairstow | 41 |
| 2.8 | O método de Newton-Raphson aplicado a sistemas não lineares | 44 |
| 2.8.1 | Exemplo de aplicação: solução de uma EDP não linear | 46 |
| 3 | Sistemas de equações lineares | 57 |
| 3.1 | Tipos especiais de matrizes quadradas | 57 |
| 3.2 | O método gráfico | 59 |
| 3.3 | Determinantes e a regra de Cramer | 60 |
| 3.4 | A eliminação de variáveis | 62 |
| 3.5 | A eliminação Gaussiana ingênuia | 63 |
| 3.5.1 | Contando o número de operações do algoritmo de eliminação Gaussiana | 65 |
| 3.5.2 | Tarefa proposta: contando operações | 68 |

| | | |
|----------|---|------------|
| 3.6 | Por que eliminação Gaussiana ingênuia? | 69 |
| 3.7 | Decomposição L.U | 73 |
| 3.7.1 | Fórmulas fechadas para a decomposição de Crout | 76 |
| 3.8 | Refletindo sobre o método de Gauss-Jordan e a decomposição de Doolittle | 76 |
| 3.9 | Motivação para o estudo de inversão matricial | 77 |
| 3.10 | Decomposição L.U e inversão matricial | 79 |
| 3.11 | Métodos para matrizes especiais | 80 |
| 3.11.1 | Sistemas envolvendo matrizes de banda | 81 |
| 3.11.2 | Sistemas envolvendo matrizes simétricas | 82 |
| 3.12 | O método de Gauss-Seidel | 83 |
| 3.12.1 | A convergência do método de Gauss-Seidel | 84 |
| 3.13 | Alguns exercícios para treinar | 87 |
| 4 | Otimização | 88 |
| 4.1 | Perspectiva histórica | 89 |
| 4.2 | Motivação inicial: otimização e aprendizado de máquina | 90 |
| 4.3 | Como formular um problema de otimização? | 95 |
| 4.4 | Razão áurea e otimização 1D | 97 |
| 4.4.1 | Exemplo de otimização 1D pelo método da razão áurea | 99 |
| 4.5 | O método da interpolação quadrática | 100 |
| 4.6 | O método de Newton | 101 |
| 4.7 | Otimização multidimensional sem restrições | 102 |
| 4.7.1 | Métodos gradientes | 103 |
| 4.7.2 | A matriz Hessiana: um critério para avaliação de pontos críticos | 105 |
| 4.7.3 | O método do aclive máximo | 108 |
| 4.7.4 | O método dos gradientes conjugados | 110 |
| 4.7.5 | Programa para casa | 113 |
| 4.7.6 | O método de Newton | 114 |
| 4.7.7 | O método de Levenberg-Marquardt | 115 |
| 4.8 | Otimização multidimensional com restrições | 116 |
| 4.8.1 | Como formular um problema de programação linear? | 117 |
| 4.8.2 | Solução pelo método gráfico | 118 |
| 4.8.3 | O algoritmo SIMPLEX | 120 |
| 4.8.4 | Para casa: algoritmo SIMPLEX | 126 |
| 4.8.5 | Programação linear com pacotes | 127 |
| 4.8.6 | Para casa: programação linear usando pacotes | 128 |
| 5 | Ajuste de curvas: regressões e interpolações | 129 |
| 5.1 | Regressões em pesquisa científica: um exemplo de cálculo numérico aplicado | 130 |
| 5.2 | Regressão por mínimos quadrados | 133 |
| 5.2.1 | Para casa: explorando regressões polinomiais e regressões múltiplas | 135 |
| 5.3 | Regressões baseadas na linearização de relações não-lineares | 136 |
| 5.3.1 | Para casa: ajustando dados por uma lei de potência | 138 |
| 5.4 | Interpolação: diferenças divididas de Newton | 139 |
| 5.4.1 | Forma geral dos polinômios interpoladores de Newton | 140 |
| 5.4.2 | Para casa: pensando sobre polinômios interpoladores de Newton | 141 |
| 5.5 | Interpolação por splines | 142 |
| 5.5.1 | Splines lineares | 142 |
| 5.5.2 | Splines quadráticos | 143 |

| | | |
|----------|---|------------|
| 5.5.3 | Splines cúbicos | 146 |
| 6 | Integração numérica | 147 |
| 6.1 | Motivação prática para o estudo de integração numérica | 149 |
| 6.2 | Fórmulas de integração de Newton-Cotes | 154 |
| 6.2.1 | A regra do trapézio | 155 |
| 6.2.2 | Regras de Simpson | 156 |
| 6.2.3 | Para treinar um pouco em casa | 157 |
| 6.3 | Integrais múltiplas | 158 |
| 6.3.1 | Programa para escrever em casa e praticar | 159 |
| 7 | Equações diferenciais ordinárias | 161 |
| 7.1 | Classificação de equações diferenciais ordinárias | 162 |
| 7.2 | Problemas de valor inicial × valor de contorno | 163 |
| 7.3 | Erros de truncamento em métodos de passo único | 167 |
| 7.3.1 | O método de Heun | 169 |
| 7.4 | Métodos de Runge-Kutta | 170 |
| 7.4.1 | Runge-Kutta de ordem 2 | 171 |
| 7.4.2 | Métodos de Runge-Kutta de ordem 3 e 4 | 172 |
| 7.4.3 | Runge-Kutta com passo adaptativo | 174 |
| 7.5 | Sistemas de EDOs acopladas | 176 |
| 8 | Equações diferenciais parciais | 180 |
| 8.1 | EDPs elípticas, parabólicas e hiperbólicas | 182 |
| 8.1.1 | Equações elípticas | 183 |
| 8.1.2 | Equações parabólicas | 185 |
| 8.1.3 | Equações hiperbólicas | 186 |
| 8.1.4 | Navier-Stokes: uma equação difícil de classificar | 187 |
| 8.2 | A equação de diferença de Laplace | 188 |
| 8.3 | Condições de contorno | 192 |
| 8.4 | Fronteiras irregulares | 196 |
| 8.4.1 | Tente fazer em casa | 197 |
| 8.5 | A abordagem por volumes finitos | 198 |
| 8.5.1 | Forma geral para representação de equações de balanço | 201 |
| 8.5.2 | Conceitos fundamentais em volumes finitos | 201 |
| 8.5.3 | Esquemas de discretização dos termos em volumes finitos | 201 |
| 8.5.4 | Exemplos de aplicação de conceitos usando OpenFOAM | 201 |
| 9 | Referências bibliográficas | 202 |

Capítulo 1

Introdução, história, generalidades e motivação

1.1 O que veremos nesse curso?

Métodos numéricos consistem em construções lógicas voltadas à solução de equações simbólicas utilizando técnicas aproximativas. A existência desses métodos precede a existência física de computadores. Na verdade, sempre que transformamos a solução de um problema matemático numa sequência de passos que vão convergindo para a solução proposta estamos aplicando um método numérico. É claro que a existência de computadores potencializa o alcance e a aplicabilidade de um determinado esquema numérico voltado à solução de um determinado tipo de equação matemática.

Como equações matemáticas podem ser enunciadas de diferentes maneiras, é natural que criemos categorias de problemas que podem ser resolvidos por meio da aplicação de esquemas numéricos de solução. De um modo geral, podemos dividir os problemas de interesse no campo dos métodos numéricos nas seguintes classes:

1. Determinação de zeros de funções: encontrar \mathbf{x} que satisfaça $f(\mathbf{x}) = 0$;
2. Determinação de soluções de sistemas de equações lineares: encontrar um vetor $\{\mathbf{x}\}$ que satisfaça $[\mathbf{A}] \cdot \{\mathbf{x}\} = \{\mathbf{b}\}$;
3. Determinação de pontos ótimos (otimização): encontrar um vetor $\{\mathbf{x}\}$ que satisfaça $\min[\mathbf{A}](\{\mathbf{x}\})$ sujeito ou não à restrições;
4. Ajustes de curvas: determinar funções que melhor aproximam distribuições de dados experimentais;
5. Integração numérica: obter valores numéricos associados ao resultado de integrais definidas aplicadas à domínios uni ou multidimensionais;
6. Soluções de equações diferenciais ordinárias: resolver x que satisfaça $\mathcal{D}\{(x)\} = 0$, em que $\mathcal{D}\{(x)\}$ é um operador diferencial;
7. Soluções de equações diferenciais parciais: obter um vetor \mathbf{x} que satisfaça $\mathcal{D}\{(\mathbf{x})\} = 0$, em que $\mathcal{D}\{(\mathbf{x})\}$ é um operador diferencial aplicado a um vetor multidimensional $\mathbf{x} = (x_1, x_2, \dots, x_n)$;

Essas categorias de divisão de problemas numéricos são úteis para organizar a informação e nos possibilitam encadear assuntos que vão ganhando complexidade com a caminhada e se complementando numa visão sistêmica. Essa visão é importante na formação de um numericista, que deve ser capaz de compreender como um método é concebido, construído, implementado, testado e validado. Nesse sentido, essa apostila será estruturada na ordem de apresentação de métodos numéricos de acordo com a lista acima.

1.2 Um pouco de história

Quando pensamos em métodos numéricos, rapidamente associamos esta palavra a números rodando na tela de um computador (terminal), sendo calculados por um processador composto por conjuntos de circuitos transistorizados utilizando álgebra booleana e muitas camadas de abstração para obter soluções aproximadas de problemas de interesse de cientistas, físicos e engenheiros. Mas, na verdade, se fôssemos contar a história dos eventos que deram origem a essa possibilidade moderna, precisaríamos voltar a um tempo bem distante.

É sabido que grande parte da história da matemática foi empreendida na busca de fórmulas fechadas para soluções (raízes) de polinômios de ordem geral. Todo estudante pré-universitário está familiarizado com a fórmula quadrática, aprendemos essa fórmula ainda no ensino de base. Mas soluções fechadas para polinômios de ordem maior que dois demandam construções axiomáticas e lógicas cada vez mais elaboradas. E é assim que a matemática se constrói e se expande. Geralmente começamos com problemas simples e vamos nos perguntando como certas estruturas se comportariam em contextos mais gerais. Essa busca por soluções de polinômios de ordem cada vez mais alta é interessante, no sentido de nos ensinar um pouco como funciona a história da própria matemática.

No caso dos polinômios, essas estruturas aparecerão muitas vezes neste curso o que pode ser entendido como um prenúncio de uma forte conexão entre o desenvolvimento da álgebra, do cálculo e dos métodos numéricos como área de conhecimento.

Em termos históricos, os Babilônios, por exemplo, já tinham conhecimento de métodos algébricos para solução de equações quadráticas do tipo $ax^2 + bx + c = 0$ desde cerca de 1800 a.C. A famosa tábua de argila *Plimpton 322* já continha sistemas para solução deste tipo de problema matemático, ver figura (1.1).

O papiro de Rhind, encontrado no Egito, sugere também que apesar dos Egípcios não terem conhecimentos de teorias gerais para soluções de polinômios, eles já resolviam equações lineares.

Apesar desses registros históricos de esquemas algébricos para soluções de certos tipos de equações por parte de Babilônios e Egípcios, é só no período clássico que esquemas matemáticos formais mais elaborados começam a surgir e se desenvolver na Grécia antiga. Aproximadamente 300 a.C., Euclides escreve um tratado geométrico clássico, chamado *Os elementos*, onde desenvolve métodos geométricos equivalentes aos métodos algébricos conhecidos para a solução de equações quadráticas. Essa junção da álgebra com a geometria amplia a compreensão de estruturas matemáticas subjacentes à diferentes tipos de sistemas de equações e estende o desenvolvimento da matemática como linguagem descritiva dos fenômenos do mundo material. O impacto que a obra de Euclides teve no mundo clássico foi tamanho que dizem que Platão mandou escrever na porta de sua Academia: *Quem não é geômetra não entre!*

Também na Grécia, aproximadamente 250 a.C., Diofanto escreve *Arithmetica*, onde introduz uma forma de notação algébrica rudimentar e estuda equações polinomiais in-

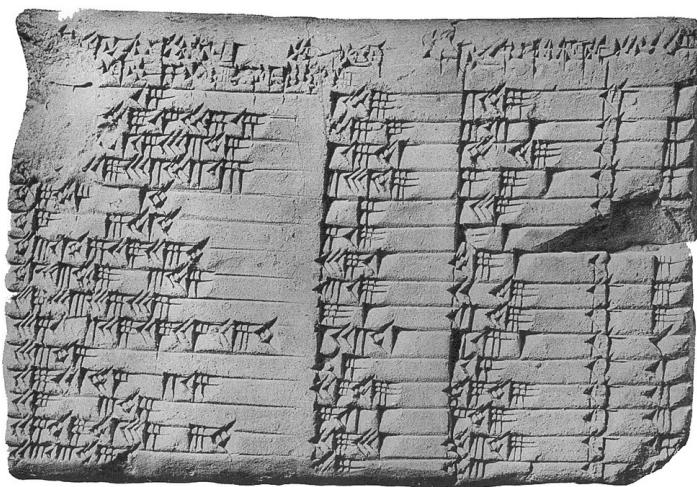


Figura 1.1: Tábua de argila Plimpton 322, usada por Babilônios cerca de 1800 a.C. para resolver equações quadráticas homogêneas

determinadas. É interessante pensarmos no aspecto do ganho de eficiência devido ao uso de notações mais práticas. O uso de geometria para deduzir teoremas matemáticos e fórmulas gerais para solução de problemas matemáticos envolve muitos triângulos, círculos e linhas que se cruzam em relações complexas que demandam tempo de quem desenvolve esses cálculos e ficam restritas ao que se pode desenhar em duas dimensões com papel e lápis. O ganho de eficiência no uso da álgebra somada à notação simbólica como ferramenta dedutiva de verdades e ideias matemáticas é um legado deixado por várias pessoas ao longo de séculos de desenvolvimento. Na época de Diófanto, o polinômio $x^3 + 13x^2 + 5x$ era escrito como $K^y\alpha\Delta^y\iota\Upsilon\zeta\varepsilon$. Imagine como a matemática produzida por seres humanos não estaria atualmente se tivéssemos que trabalhar com uma notação dessas até hoje?

Após esses avanços no período Grego, temos a dominação da Grécia pelo império Romano e a perseguição à parte de elementos da cultura grega. A queima da Biblioteca de Alexandria é um exemplo simbólico de uma transição cultural que marca o fim do período clássico e o início do que muitos historiadores denominam de Idade Média. Esse movimento gera uma fuga do espírito Grego para regiões do Oriente. Nessa época existem registros de métodos utilizados para soluções de quadráticas por Brahmagupta na Índia em 628 d.C. e Al-Khwarizmi no mundo Árabe em 830 d.C. Já na China, no século XII, Qin Jiushao em seu tratado matemático em nove capítulos descreve a regra de Cramer para solução de sistemas lineares, muito antes da formulação ocidental e introduz técnicas para lidar com matrizes de coeficientes. Veremos neste curso que sistemas de equações lineares são blocos básicos de construção de diversos métodos numéricos voltados às mais distintas finalidades. É interessante notar que as ideias associadas à construção desses métodos circulam entre nós muito antes do surgimento da álgebra booleana e dos circuitos transistorizados que dominam o mundo moderno.

Com o fim da Idade Média e início do período histórico conhecido como renasença, temos um florescimento do conhecimento matemático e científico na Europa ocidental. No século XVI, Tartaglia e Cardano descobrem fórmulas para resolver equações cúbicas do tipo $ax^3 + bx^2 + cx + d = 0$. Por volta de 1540, Ludovico Ferrari, aluno de Cardano, resolve equações de quarto grau do tipo: $ax^4 + bx^3 + cx^2 + dx + e = 0$. Esses avanços mostram um rápido processo de evolução de ideias matemáticas, estimulado por uma revolução cultural importante na história da humanidade, na qual o ser humano empreende um processo de

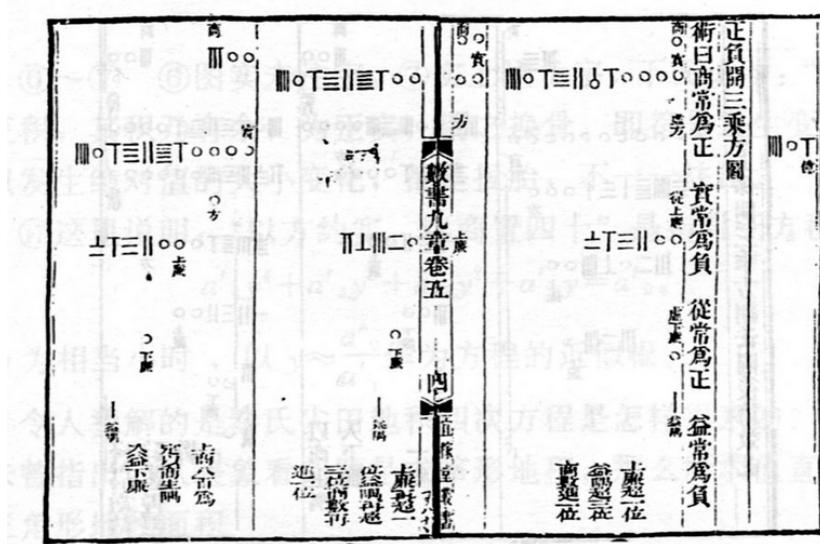


Figura 1.2: Trecho da obra de Qin Jiushao, um tratado matemático em nove capítulos produzido na China no século XII

redescoberta de seu lugar no mundo em algumas partes da Terra.

Em 1665, Newton desenvolve métodos semelhantes ao que hoje conhecemos por série de Taylor para expandir funções contínuas em termos de séries infinitas, porém não chega a publicar formalmente seus resultados nesse tópico, pois devia estar muito ocupado inventando o cálculo diferencial, formulando as leis do movimento e revolucionando a física para sempre. Em 1683, Leibniz desenvolve o conceito de determinantes, fundamental no campo da álgebra linear e na solução de sistemas lineares. Já no ano de 1715, Brook Taylor publica a fórmula da série de Taylor em seu artigo *Methodus Incrementorum Directa et Inversa*. No ano de 1801, Gauss, na Alemanha, desenvolve o método de eliminação Gaussiana, um procedimento sistemático para a solução de sistemas lineares, utilizado em muitos algoritmos modernos voltados por exemplo à simulação de escoamentos no campo da Dinâmica dos Fluidos Computacional.

Todas essas ideias que trouxemos até aqui dizem respeito ao campo da matemática. Ainda não tínhamos computadores na época de Gauss. Mas as ideias por trás do desenvolvimento dessas máquinas começaram a circular mais ou menos por volta dessa mesma época. O ano de 1804 é especialmente importante na história da computação. Nesse ano, um francês chamado Joseph-Marie Jacquard se viu com um problema que precisava ser resolvido. Jacquard era um empresário do setor têxtil e ganhava um bom dinheiro vendendo tecidos estampados na França. Porém, para bordar um determinado padrão no tecido, um funcionário da fábrica precisava manualmente trocar as linhas do tear conforme o mesmo ia bordando o tecido. Jacquard percebeu que ele poderia automatizar o processo mudando a mecânica de sua máquina e incorporando a ela um sistema de cartões perfurados previamente confeccionado que iria girar junto com o rolo do tear e permitir que apenas certas agulhas passassem por certos buracos ao longo do tempo. Com isso ele inventa o primeiro tear automatizado. Essa ideia de automatizar um processo mecânico por meio de uma programação prévia é o embrião de todo computador moderno. Qualquer semelhança entre os cartões perfurados da máquina de Jacquard e os cartões perfurados dos primeiros programas em FORTRAN do final dos anos 1950 não é mera coincidência.

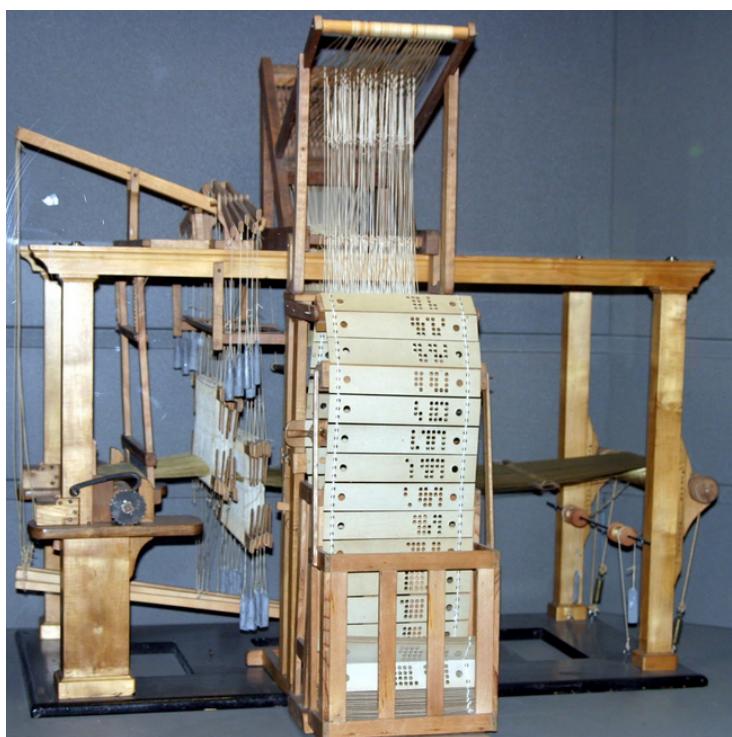


Figura 1.3: Réplica do modelo de tear automatizado desenvolvido por Jacquard. Pode não parecer um computador moderno, mas é quando o conceito de programas armazenados surge pela primeira vez.

Nessa época, na França, um outro personagem muito importante na história da computação entra em cena. Seu nome é Charles Babbage. Em 1822, Babbage constrói a máquina de diferenças, um dispositivo de computação mecânica programável de uso geral, voltado à tabulação de logaritmos e funções trigonométricas usando aproximações polinomiais baseadas em diferenças finitas. Sua máquina era voltada à aplicações diversas, incluindo o cálculo de trajetórias balísticas que durante uma eventual guerra dependia da consulta à grandes tabelas no meio do campo de batalha. A máquina de Babbage era uma engenhoca mecânica cheia de engrenagens, eixos, cilindros e manivelas que se organizavam num movimento sincronizado de contagem mecânica: uma verdadeira obra-prima de engenharia, mas muito complexa e de difícil construção, ver figura (1.4). Tão difícil, que em 1834, quando Babbage concebe o projeto de uma máquina de uso mais geral, que ele apelidou de máquina analítica, ninguém conseguiu construí-la e ela não chegou a sair do papel. Mas seu projeto e sua lógica incluíam uma série de características usadas hoje em computadores modernos. Essa máquina hipotética foi projetada para armazenar informações (memória) e executar sequências específicas de instruções por meio de placas perfuradas (programas).

As ideias por traz da máquina analítica de Babbage circularam pela Europa e em 1843, a matemática Ada Lovelace, amiga de Charles Babbage, propõe uma sequência de instruções de como o modelo da máquina analítica poderia calcular a sequência de Bernoulli. Essa sequência de instruções é conhecida como o primeiro programa de computador da história. É só uma pena que Ada Lovelace não pode chegar a compilar esse programa, pois a máquina para tal compilação ainda não existia. Quatro anos depois, em 1847, George Boole escreve *The mathematical analysis of logic*, no qual apresenta ao mundo a álgebra Booleana, uma forma de lógica fundamental por trás do funcionamento de todos

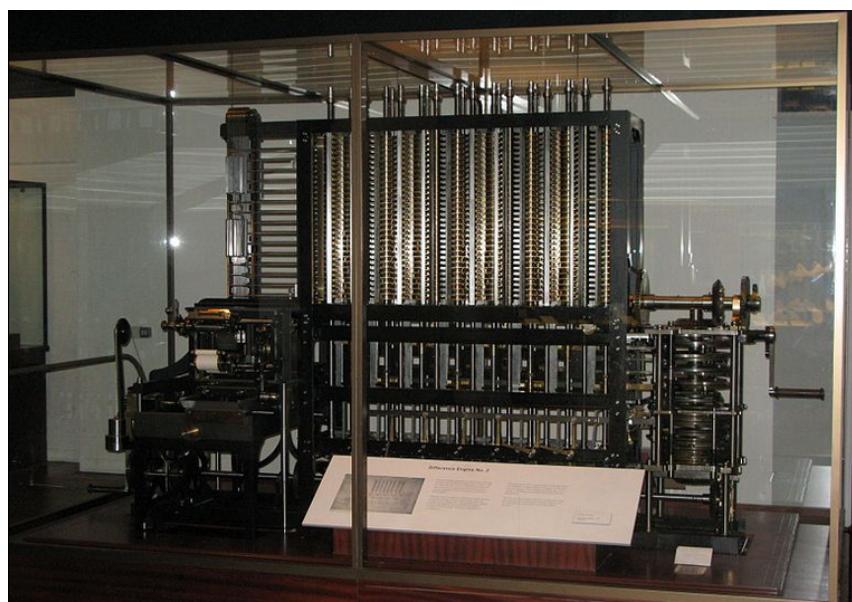


Figura 1.4: Réplica do modelo da máquina de diferenças de Babbage, construída com base no projeto original do próprio autor e exposta no museu de Ciências de Londres

os nossos computadores modernos.

Percebe-se que nessa época, o terreno estava fértil para o surgimento de máquinas de computação mais modernas. Mas faltava ainda uma oportunidade que justificasse o uso dessas máquinas em maior escala. E essa oportunidade surgiu, como muitas vezes, na forma de um problema. No final do século XIX, os Estados Unidos precisavam fazer um censo para avaliar dados demográficos de uma população em rápida expansão. Antigamente, os métodos de contagem do censo eram manuais e muito lentos. A estimativa na época era de que a contagem do censo demoraria cerca de 13 anos, período em que a população provavelmente já teria mudado significativamente de tal sorte que aquela informação já não teria mais relevância. Diante desse problema, no ano de 1889 um criativo engenheiro chamado Herman Hollerith inventa uma máquina eletromecânica de contagem que diminuiu esse tempo para cerca de 2 anos, resolvendo o problema do censo. Em 1911, Hollerith se junta a outros fabricantes de máquinas de cálculo voltadas à aplicações comerciais e funda a International Business Machines Corporation (IBM).

Em 1936, Alan Turing publica um artigo no qual propõe um modelo abstrato de um computador, conhecido como a máquina de Turing. É interessante ver como o interesse pelo aspecto teórico da computação passa a fomentar o desenvolvimento de máquinas reais cada vez melhores voltadas à finalidades de cálculo computacional. Até esse momento as máquinas eram mais mecânicas do que elétricas, mas a partir da segunda guerra esses dispositivos passam a incorporar cada vez mais elementos elétricos em seus projetos de engenharia.

Em 1944, a IBM constrói o Mark 1, a primeira calculadora eletromecânica automática que operava com base em relés, interruptores, eixos rotativos, engrenagens e embreagens. Essa máquina pesava cerca de 4.500 kg, possuía 3.500 relés, 800 km de fiação e cerca de 3 milhões de conexões. Imagine dar manutenção numa máquina como essa. Um problema dessas máquinas é que elas eram lentas. Principalmente devido ao tempo de abertura dos relés, que são dispositivos eletromecânicos. Um relé pode ser pensado como uma válvula que está aberta ou fechada e pode mudar de estado em função de uma programação externa. A mudança no estado de um relé é feita com base na passagem ou não de

corrente elétrica por um eletroímã que por efeito magnético abre ou fecha um contato. Esse processo é um processo lento quanto comparado com outros dispositivos mais modernos como os transistores. Na verdade, o aumento da velocidade de processamento dessas máquinas foi em grande parte estimulado pelo desenvolvimento tecnológico de dispositivos de abertura e fechamento de circuitos mais rápidos.



Figura 1.5: Mark 1: a primeira calculadora eletromecânica baseada em relés, feita pela IBM

Em 1946, por exemplo, o computador integrador numérico eletrônico, o ENIAC, entra em funcionamento. Esse dispositivo foi o primeiro computador digital eletrônico de grande escala e foi utilizado principalmente em aplicações militares voltados à cálculos balísticos. Essa máquina podia realizar 5000 operações por segundo graças às suas 17468 válvulas termiônicas, dispositivos de abertura e fechamento mais rápidos que relés e altamente sensíveis. Quem possui um amplificador de guitarra valvulado sabe do que estamos falando. O ENIAC ocupava uma sala de $150\ m^2$ e pesava cerca de 30 toneladas.



Figura 1.6: O ENIAC: o primeiro supercomputador famoso do mundo

O desenvolvimento de novos computadores vem acompanhado por uma questão interessante: como programar essas novas máquinas? Até o momento a programação de uma máquina envolvia basicamente a conexão de plugues em conectores fêmeas para fechar diferentes circuitos em grandes painéis. Cada máquina tinha seu desenho de engenharia e sua arquitetura de painel. Não existiam linguagens universais de programação. Até que em meados dos anos 1950 surgem as primeiras linguagens como Assembly e FORTRAN, sigla em inglês para IBM Mathematical Formula Translation System, criada em 1954 e lançada oficialmente em 1957. O FORTRAN é um marco na história da programação científica, uma vez que possuía fórmulas matemáticas internas prontas, como funções exponenciais, logarítmicas e demais funções transcendentais. Por esse motivo é usado até hoje por físicos, engenheiros e cientistas.

De lá para cá muita coisa mudou. Novas linguagens surgiram, veio a portabilidade de software entre diferentes máquinas e com ela uma indústria de software separada da indústria de hardware, novos bilionários se fizeram no vale do silício, diferentes sistemas operacionais novos surgiram, regras rígidas de proteção de direitos autorais em softwares proprietários passaram a operar, muitos códigos deixaram de ser abertos e passaram a ser vendidos para usuários que não precisavam ser desenvolvedores e novas filosofias e formas de utilização do poder de cálculo computacional para diversas finalidades foram surgindo e se aperfeiçoando. O setor acadêmico passou a usar computadores para escrever simuladores e aumentar nossa capacidade de entendimento e predição do comportamento de sistemas físicos. A engenharia passou a usar recursos computacionais e criou a sua própria sigla para isso: CAE (computer assisted engineering). Áreas de projeto passaram a usar CAD (computer assisted drawing) para representar graficamente seus designs. A dinâmica dos fluidos passou a utilizar CFD (computational fluid dynamics) para simular escoamentos de interesse científico e industrial. Engenheiros de estruturas passaram a dispor da técnica de elementos finitos (FEM - finite element method) para dimensionamento de esforços e deformações mecânicas em geometrias cada vez mais complexas com o auxílio de computadores. E agora assistentes de inteligência artificial nos auxiliam nas nossas dúvidas, cálculos, códigos e projetos. Mas apesar de todas essas revoluções, os fundamentos para a compreensão, elaboração, construção, implementação, validação e utilização de todas essas possibilidades continuam os mesmos. Por isso é fundamental aprendermos os fundamentos que nos levam do cálculo à simulação computacional caso queiramos ser numericistas confiantes, criativos e eficazes no uso de todas essas ferramentas aplicada à solução de problemas do mundo real.

1.3 Como funciona um computador?

Antes de mergulharmos no estudo de algoritmos e métodos computacionais, é fundamental entendermos como funciona um computador, ainda que de forma simplificada. O objetivo aqui não é fornecer uma visão aprofundada de todas as camadas de abstração necessárias à compreensão de como um computador opera no nível do que se espera de um cientista da computação por exemplo. A ideia é dar uma visão de Engenharia de como um computador funciona, para que Engenheiros interessados em utilizá-lo como ferramenta de solução de problemas práticos possam interagir com um computador a partir de um ponto em que certas camadas de abstração tenham sido expostas e compreendidas num nível mínimo. Essa compreensão nos oferece uma noção mais concreta sobre como as instruções que escrevemos em um código de alto nível são transformadas em operações físicas realizadas sobre a matéria, mais especificamente, sobre cargas elétricas transitando por milhões (ou

bilhões) de pequenos interruptores microscópicos.

Bom, um computador é, essencialmente, uma máquina de manipular bits. A palavra bit é uma abreviação de binary digit (dígito binário). Um bit representa a unidade mínima de informação em um computador e pode assumir apenas dois valores: 0 ou 1. O que chamamos aqui de zero é a associação a uma ausência de tensão elétrica cruzando um determinado circuito. Já o outro estado, o 1 é a associação direta à presença de tensão elétrica passando pelo mesmo tipo de circuito.

No jargão técnico da ciência da computação, temos também a ideia do byte, que nada mais é do que um conjunto de 8 bits. Isso significa que um byte pode representar combinações diferentes de valores binários — ou seja, pode representar inteiros de 0 a 255 (em geral), caracteres de texto (como no padrão ASCII), ou qualquer outro dado codificável nesse espaço. Existem portanto 2^8 formas diferentes de organizarmos 8 sequências de dígitos binários. Por isso o número 256 aparece com tanto frequência no mundo da computação. Um exemplo simples: o número binário 01100001 corresponde tanto ao número decimal 97 quanto à letra "a" no padrão ASCII.

Para operarmos um computador precisamos de entradas e saídas. As entradas, como teclados, mouses, câmeras e microfones, enviam informações que devem ser processadas dentro dessa estrutura que chamamos de computador e geram consequências que se manifestam ao usuário por meio de saídas, como monitores, impressoras e caixas de som, como ilustrada na figura (1.7).

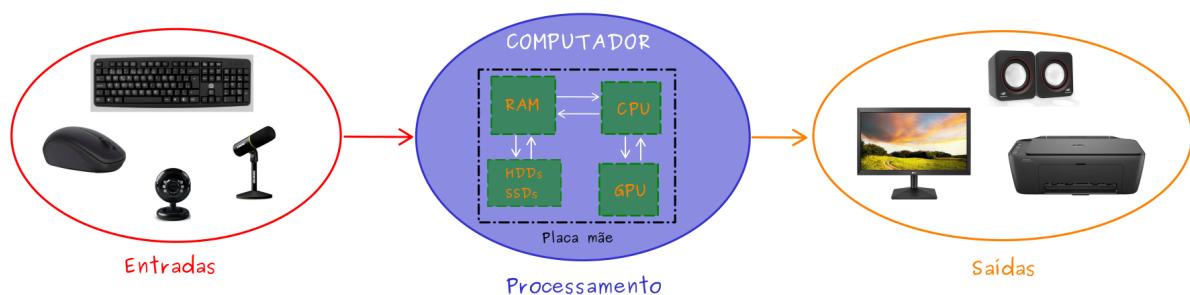


Figura 1.7: Relações entre um computador e eventuais entradas e saídas

Toda a informação processada — texto, imagem, som, equações, variáveis — é representada usando dois símbolos apenas: 0 e 1. Essa escolha não é uma escolha estética, mas sim prática: os computadores são feitos de componentes eletrônicos que podem ter dois estados físicos bem definidos, como presença ou ausência de tensão elétrica. Esses dois estados são chamados de níveis lógicos, e formam a base do sistema binário utilizado pelos computadores.

Podemos pensar em um computador como uma *máquina que processa informações seguindo instruções, combinando componentes de hardware e software para realizar tarefas*. Essa definição é muito interessante como ponto de partida para entendermos questões centrais sobre como um computador funciona. A primeira palavra importante nessa definição é a palavra *informação*. Para nós, seres humanos, uma informação é basicamente algo que tenha algum tipo de significado. Podemos pensar em imagens, sons, textos, vídeos, etc. Para um computador, uma informação é uma sequência binária que passa a representar algo quando interpretada. Para que um computador atenda nossas necessidades, precisamos dizer a ele o que queremos. Em outras palavras, precisamos transmitir a esta espécie de cérebro eletrônico informações que ele seja capaz de compreender a fim de processá-las e nos retornar informações que nós sejamos capazes de entender. Quem

define o que um computador irá fazer no fundo é um ser humano por meio da escrita de um programa. Por mais que hoje em dia tenhamos modelos de inteligência artificial capazes de escrever programas, é importante lembrarmos que esses modelos também são no fundo programas.

Podemos entender um programa como uma longa sequência binária, adequada à compreensão da máquina, gerada a partir de um texto escrito de um jeito bem específico, que nós chamamos de código. Muitas vezes chamamos o código de programa, mas no fundo o programa é um binário que foi gerado a partir do código. Esse código é escrito numa certa linguagem que ao ser interpretada ou compilada (falaremos sobre compiladores logo mais) resulta numa sequência de instruções binárias que serão repassadas à unidade central de processamento. Essas instruções binárias, por sua vez vão abrir e fechar circuitos elétricos por meio de portas lógicas formadas por associações de transistores que irão processar essas instruções gerando como resultado (saída) um outro binário. Esse binário de saída é então decodificado por meio de programas próprios para isso, por exemplo um editor de texto que irá nos mostrar a saída do nosso programa numa linguagem mais parecida com a que nós entendemos. A figura (1.8) ilustra esse fluxo de informação.

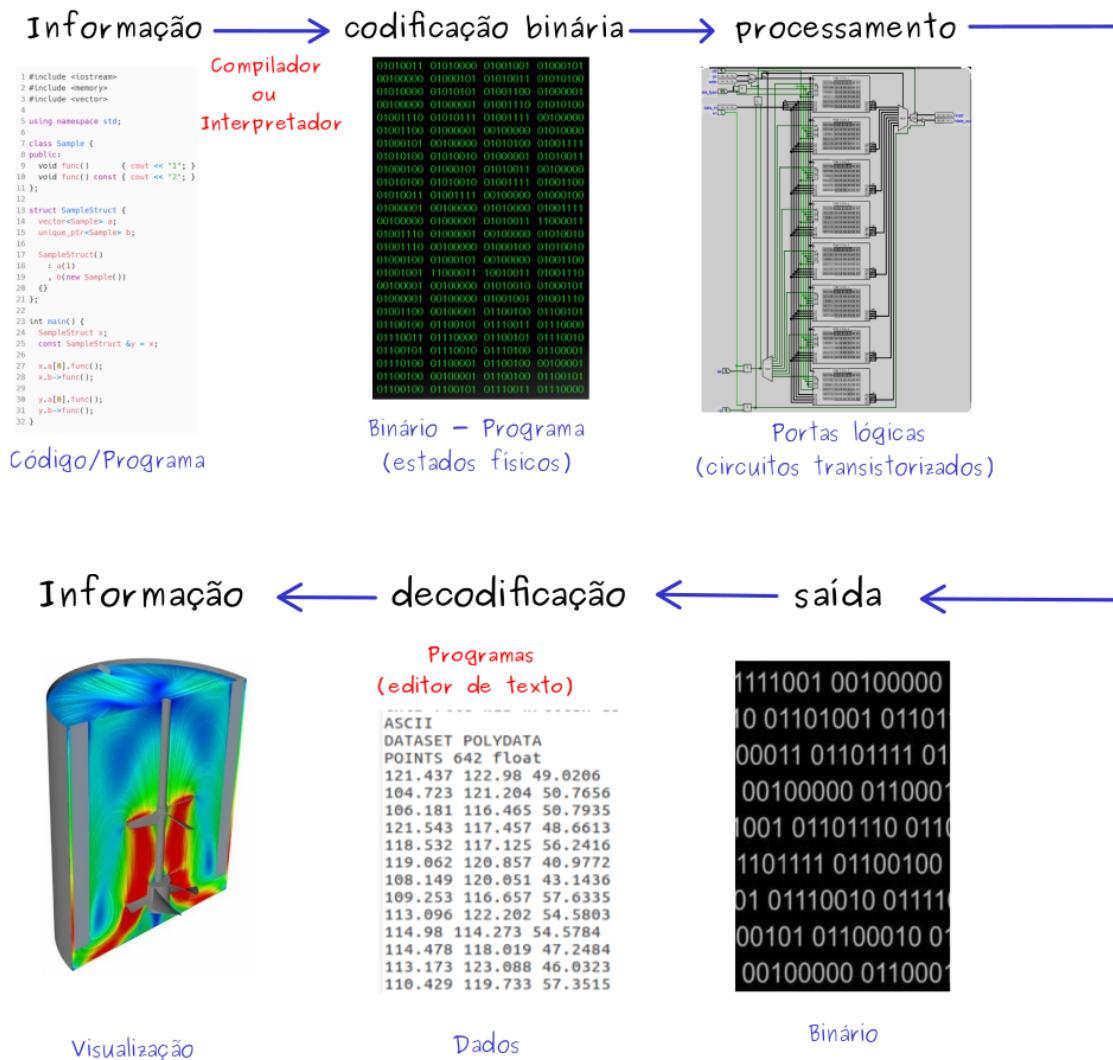


Figura 1.8: Do código à visualização: fluxo típico da informação numa simulação computacional

1.3.1 Portas lógicas e álgebra Booleana

Aqui surge já uma questão importante: como é que um conjunto de 0s e 1s pode fazer cálculos complexos? Para que isso seja possível precisamos recorrer a uma área da matemática, conhecida como álgebra booleana, um sistema lógico que opera com valores verdadeiros (1) e falsos (0), e define regras de operação como AND, OR, NOT, entre outras para realizar toda espécie de conta matemática. As primeiras calculadoras eletrônicas, baseadas em relés, como o Mark I, já faziam seus cálculos com base no uso de circuitos elétricos e álgebra booleana.

Essas operações são fisicamente implementadas usando portas lógicas, que nada mais são do que pequenos circuitos eletrônicos compostos por transistores. O transistor é como um interruptor: pode permitir ou bloquear o fluxo de corrente elétrica, dependendo de um sinal de controle. O princípio básico de funcionamento de um transistor se baseia num entendimento a nível molecular da ciência de semi-condutores, que permite abrirmos e fecharmos válvulas minúsculas de corrente elétrica de forma ultra-rápida. A vantagem de um transistor em relação aos antigos relés e às válvulas termiônicas é gigantesca. Transistores são rápidos, pequenos, silenciosos e demandam pouco trabalho de manutenção.

As portas lógicas utilizadas para realização de operações matemáticas usando numeração binária e álgebra booleana são compostas por associações em série ou paralelo de transistores ligados em mini-circuitos. Mas o que são essas portas? Uma porta AND, por exemplo, só libera um "1" de saída se ambas as entradas forem "1". Uma porta OR libera "1" se pelo menos uma das entradas for "1". Uma porta NOT inverte o valor: transforma "1" em "0" e vice-versa.

Combinando essas portas, podemos construir circuitos mais complexos capazes de somar, subtrair, comparar e controlar o fluxo de informações.

1.3.2 CPU e memória

O componente físico de um computador que controla todo esse fluxo de corrente elétrica em circuitos transistorizados representados pela conexão de diferentes portas lógicas nessa pequena cidade miniaturizada que é um computador é chamado de CPU. A CPU (Central Processing Unit) é o “cérebro” do computador. É nela que são realizadas operações aritméticas, lógicas e de controle. A CPU funciona de maneira cíclica, baseada em uma sequência de etapas conhecidas como ciclo de máquina:

1. Busca (fetch): a CPU busca uma instrução da memória;
2. Decodificação (decode): interpreta qual operação deve ser realizada;
3. Execução (execute): realiza a operação, usando seus registradores internos e a unidade lógica-aritmética (ULA).
4. Armazenamento (write-back): grava o resultado de volta em um registrador ou na memória principal.

Essas instruções são codificadas em linguagem de máquina, composta por sequências de bits que a CPU é capaz de interpretar diretamente. Mas para um computador funcionar, precisamos também de uma estrutura que seja capaz de memorizar informações, dados, processos a fim de manter um fluxo temporal contínuo que use os recursos da CPU para realizar tarefas sequencialmente e em paralelo. Em outras palavras, precisamos também de memória. E existem dois tipos principais de memória. A primeira

delas, chamamos de RAM. A memória RAM é onde os dados e instruções ficam temporariamente armazenados enquanto um programa está em execução. Ela pode ser lida e escrita rapidamente, mas é volátil — ou seja, perde seu conteúdo quando o computador é desligado.

A CPU acessa a RAM por meio de um barramento de dados e um barramento de endereços, que funcionam como canais de comunicação entre os componentes. Nesse sentido, um importante *insight* para o numericista é que memória RAM importa principalmente quando precisamos fazer várias coisas ao mesmo tempo. Além disso, é importante entender que a CPU precisará dialogar com a memória RAM para poder realizar cálculos associados à execução de um programa. Esse diálogo demora um certo tempo e depende da velocidade da memória RAM, que possui não só uma capacidade de armazenamento de informações, mas também uma velocidade de leitura e escrita associada.

Um outro tipo de memória importante, está associada a dispositivos de armazenamento como HDs, SSDs ou pendrives. Essas formas de memória guardam informações de forma não volátil, permitindo que dados sejam preservados mesmo sem energia elétrica. Além disso, no âmbito da CPU, existe uma outra forma de memória ultrarrápida chamada de memória cache e que possui papel semelhante à RAM, porém com capacidade de armazenamento de informações menor e mais velocidade de comunicação com a CPU. Dependendo da linguagem de programação que você escolher usar, um entendimento desses conceitos vai te ajudar a estruturar um código mais ou menos eficiente em termos de processamento.

1.3.3 O formato IEEE 754

Uma outra pergunta central aqui é: como um computador lida com números reais, uma vez que se baseia em um sistema binário? Como ele lida com abstrações como zero, números negativos, irracionais e números dessas naturezas estranhas ao universo binários dos circuitos transistorizados? Afinal de contas, para realizarmos tarefas de interesse de Engenharia utilizando métodos numéricos, precisamos que o computador consiga representar e manipular números de todas essas categorias *estranhas*.

Bom, números inteiros positivos não representam grandes desafios em termos de formatação, uma vez que podem ser representados diretamente em binário. O número 5 em binário é dado diretamente pela sua versão binária $101 = 1 \times (2^2) + 0 \times (2^1) + 1 \times (2^0) = 4 + 0 + 1 = 5$. Já o número 12 em binário é dado por 1100. Para representar números negativos, usa-se geralmente o método do complemento de dois, que permite que operações de soma e subtração sejam feitas de forma eficiente em hardware. Esse método é bem interessante e em certo sentido genial. Sabemos por exemplo que $5 - 5 = 0$, de tal sorte que o número -5 é aquele que somado ao número positivo $+5$ gera como resultado o número 0. Como representamos então o número -5 em binário? O algoritmo é simples: invertemos do número positivo todos os bits, ou seja, trocamos zeros por uns e vice-versa e em seguida somamos 1. Lembre que o número 5 pode ser representado em binário por 101, mas aqui usaremos 4 dígitos binários, ou seja 0101. Isso é útil nesse exemplo e alinha-se à forma que um computador é programado para operar. Apesar da unidade mínima ser o bit, um computador é projetado para operar em cima de agrupamentos de bits na forma de bites. Quantidades ímpares de bits não fazem muito sentido na lógica de um computador e aqui usaremos um exemplo com 4 bits para transmitir a forma de representação de números negativos por meio do complemento de 2.

Se invertermos a representação do número 5, representado em 4 bits, ou seja, 0101,

obtemos 1010, somando 1 a esse número, obtemos $1010 + 0001 = 1011$. Note que se somarmos $1011 + 0101 = 10000$. Nessa soma, quando somamos $1 + 1 = 2$, o resultado por casa é atribuído como zero e subimos o número um (carry). Ao final, ignoramos o quinto dígito 1 e os 4 bits resultantes são todos nulos. Ou seja, o método funciona. É claro que os maiores números positivos ou os menores números negativos passíveis de serem representados por esse método vão depender do número de bits utilizados para representá-los.

Já números fracionários e ponto flutuante são mais desafiadores, pois fogem do que um computador é capaz de representar diretamente e precisam de certas convenções para serem representados e utilizados em manipulações matemáticas por meio das associações de portas lógicas dentro da nossa unidade lógica aritmética. Números como 3.14 ou -0.001 precisam de um formato mais geral: o ponto flutuante, que segue a norma IEEE 754. O IEEE é um importante instituto de Engenheiros Eletricistas e Eletrônicos e convencionou uma forma de lidar com esses números dentro do universo binário de um computador. A ideia desse padrão é similar à notação científica. No padrão IEEE 754, o formato de ponto flutuante expressa um número decimal genérico como:

$$\text{Número} = \text{Sinal} \times \text{Mantissa} \times 2^{\text{Expoente}}$$

O sinal será atribuído como 0 para representar um número positivo e 1 para representar um número negativo. A quantidade de bits necessários para representação do sinal é um. Ou seja, um único dígito binário é suficiente para representação do sinal de um número. Já a mantissa, está associada à parte significativa do número e é utilizada para representar a componente fracionária do mesmo. Mostraremos como ela é construída. Considere representar por exemplo o número -5.75 usando o formato IEEE 754. Já vimos que o primeiro dígito define o sinal do número. Aqui temos o dígito 1 para representar o fato de que estamos lidando com um número negativo. Para construção da mantissa, o processo funciona da seguinte forma: começamos pela parte fracionária, ou seja, 0.75. Pegamos esse número e multiplicamos uma vez por 2 $\rightarrow 0.75 \times 2 = 1.5$. O resultado é separado como $01.5 = 1 + 0.5$. Já temos um dígito binário no processo, o número 1, mas ainda temos casas decimais que sobraram, no caso 0.5. Multiplicamos novamente o resultado por 2 $\rightarrow 0.5 \times 2 = 1$. Agora, temos um outro dígito binário, novamente o 1 e nenhuma sobra decimal. Até aqui nossa parte fracionária original 0.75 virou .11. Mas lembrem-se que nosso número é -5.75. O número 5 já vimos que é 101 em binário. O decimal 0.75 até aqui é dado por .11, de tal sorte que $5.75 = 101.11$. Mas na notação IEEE 754 nós representamos esse número de maneira diferente. O procedimento aqui consiste em mover o ponto duas casas para trás, justamente o número de multiplicações sucessivas por 2 que tivemos que fazer para eliminar as sobras decimais do número 0.75. Por esse motivo, atribuímos a expressão *ponto flutuante* a essa forma de representação. Com esse deslocamento do ponto, temos que: $5.75 = 1.0111 \times 2^2$. Aqui o expoente 2 nos auxilia a reconstruir o número desejado usando apenas dígitos binários. Mas na notação IEEE 754 ainda temos mais elementos para serem trabalhados.

A mantissa é geralmente expressa utilizando 23 dígitos binários. Essa é uma quantidade capaz de gerar representações nesse formato de números originais com 7 a 8 dígitos decimais, o que fornece uma precisão razoável e compatível com a maior parte das aplicações em Engenharia. Dessa forma, nossa mantissa que em 4 bits é dada por 0111, quando representada em 23 bits se torna 0111000000000000000000000. Já o expoente, é geralmente representado utilizando 8 bits. Mas aqui entra um truque interessante. Para a representação do expoente utilizamos um artifício matemático chamado de *bias*, que no

nosso contexto quer dizer *excesso*. Aqui definiremos bias=127. De tal sorte, que nosso expoente verdadeiro 2 será transformado em $127 + 2 = 129$. Esse número, 129, chamamos de *expoente armazenado*. Para recuperarmos o valor original do expoente, fazemos expoente original = expoente armazenado – bias. Esse truque é feito para que possamos expressar expoentes que variam entre -126 à $+127$ utilizando 8 bits para armazenar o valor do expoente. Dessa forma, o expoente do nosso exemplo 2 se torna 129, que em binário (utilizando 8 bits) é dado por 10000001. Finalmente, nosso número -5.75 é armazenado no formato IEEE 754, como 11000000101110000000000000000000.

Uma característica importante desse formato é que certos números jamais conseguirão ser expressos de maneira exata, o que nos leva a erros de arredondamento inevitáveis. Esses erros, por sua vez, poderão ser minimizados aumentando o número de casas decimais consideradas durante a execução do programa. A maioria das linguagens nos permite definir categorias de números com quantidade de casas decimais pré-definidas para cada variável utilizada durante a execução do programa. Ao mesmo tempo que mais casas decimais significam menores erros de arredondamento, o custo a ser pago é mais espaço de armazenamento necessário (memória).

1.3.4 Um pouco sobre compiladores

Um outro conceito muito importante para estudantes de métodos numéricos é a ideia de compilador. Esse conceito, que antigamente era mandatório para quem escrevia códigos de computadores, passou a ser acessório na era moderna em que linguagens como Python, que não demanda a passagem explícita por um compilador para execução de códigos por parte do usuário, passaram a ser mais populares que as antigas e robustas linguagens compiláveis como FORTRAN e C++. Mas quem trabalha com computação de alto desempenho sabe da importância de um bom compilador. O compilador é o programa que faz a ponte entre o código e o hardware.

Quando escrevemos um programa em uma linguagem compilável como C++ ou FORTRAN, não estamos falando diretamente com a CPU. O que escrevemos precisa ser traduzido para linguagem de máquina. Essa é a função do compilador. O compilador transforma o código-fonte em um conjunto de instruções binárias específicas para o processador em uso. Compiladores otimizados são fundamentais para garantir eficiência e performance em aplicações numéricas, especialmente em métodos que exigem milhões de operações por segundo. Dependendo da física que precisa ser resolvida e da resolução espaço-temporal demandada pelo fenômeno físico, precisaremos dar muita atenção não só à linguagem na qual escreveremos nossos programas, como também ao compilador utilizado para geração do programa em si. Além disso, a arquitetura na qual o processador é estruturado também é um fator importante e a adequação do compilador que será usado para a criação de um programa (executável) otimizado para determinada arquitetura é um fator relevante na escolha do ambiente no qual um certo programa será desenvolvido e estruturado.

Todo esse funcionamento interno, das portas lógicas ao ponto flutuante, molda os limites e as possibilidades dos algoritmos que desenvolveremos. Um método numérico que “funciona no papel” pode falhar no computador por conta de erros de arredondamento, cancelamento numérico ou limitações de precisão. Entender como um computador funciona, ainda que no nível superficial descrito aqui, nos ajuda a programar melhor, interpretar erros com mais consciência e projetar soluções numéricas mais robustas.

1.4 Por que precisamos de métodos numéricos?

Antes de entrarmos na apresentação dos métodos que serão vistos nesse curso, precisamos nos fazer uma pergunta fundamental: *por que precisamos de métodos numéricos?* E para tentar responder essa questão, vamos começar com um problema simples que será usado como um exemplo didático. Este problema consiste no estudo do processo de sedimentação de uma pequena esfera de raio a , com massa específica ρ_s que se desloca em um fluido de viscosidade η sob a ação da gravidade \mathbf{g} . A figura (1.9) ilustra esse esquemático.

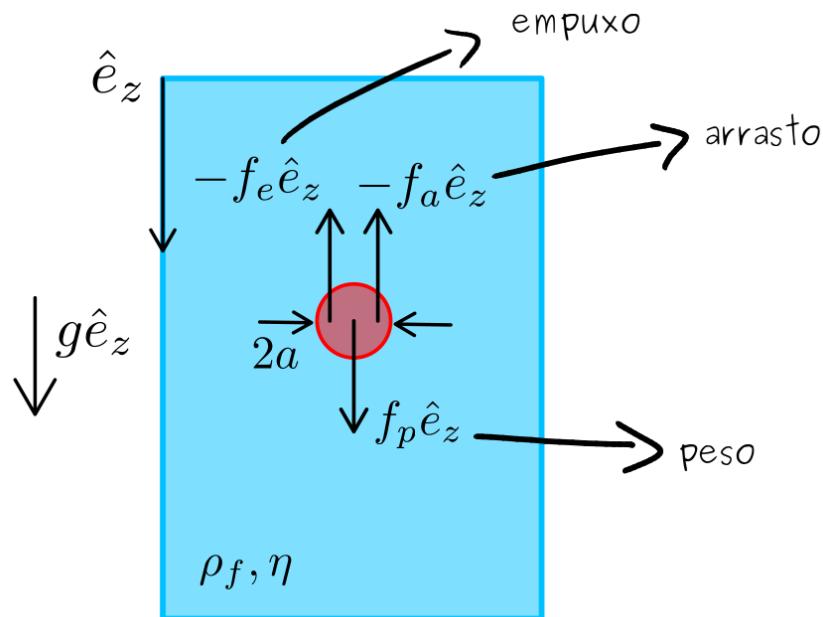


Figura 1.9: Esquemático do problema que iremos resolver nessa atividade.

Para esse problema, a esfera tem seu movimento descrito pela segunda lei de Newton:

$$m_s \frac{d\mathbf{v}}{dt} = \sum \mathbf{f}, \quad (1.1)$$

em que m_s é a massa da esfera, $\mathbf{v} = v_x \hat{e}_x + v_y \hat{e}_y + v_z \hat{e}_z$ é o vetor que representa a velocidade instantânea da esfera em cada direção do espaço, t é o tempo e \mathbf{f} representam as forças que atuam sobre a esfera. Na nossa modelagem consideraremos que a esfera encontra-se sujeita a uma força de arrasto \mathbf{f}_d proveniente da interação com o fluido base e ao empuxo líquido $\mathbf{f}_g = v_p(\rho_s - \rho_f)\mathbf{g}$. Aqui $v_p = 4\pi a^3/3$ é o volume da esfera, ρ_f é a densidade do fluido e $\mathbf{g} = -g \hat{e}_z$ é o vetor aceleração da gravidade.

A força de arrasto devido à interação da superfície da esfera com o fluido ao seu redor demanda uma modelagem um pouco mais complicada. Dependendo da velocidade da esfera, o regime no qual o escoamento induzido pelo seu movimento se estabelece pode levar a diferentes expressões. Quando a combinação de parâmetros dada por $Re_a = \rho_f |v| a / \eta$ é grande, o escoamento induzido pelo deslocamento da esfera no fluido estará sujeito à perturbações transitórias que se manifestam sob a forma de múltiplos vórtices (pequenos redemoinhos) que se organizam na forma de uma cascata dissipativa da energia que a esfera transfere ao fluido. Esse é um limite assintótico conhecido como *escoamento turbulento*. Nesse contexto, a força de arrasto que age sob a esfera depende fortemente de sua geometria de tal sorte que pequenas perturbações geométricas, como por exemplo o

aumento da rugosidade do material da esfera, afetam a topologia do escoamento induzido na região da esteira. Nesse limite temos um arrasto quadrático não-linear em que $|\mathbf{f}_d| \sim a^2$. Já no contexto em que $Re_a \ll 1$, o escoamento induzido pelo deslocamento da esfera é dominado pelos efeitos viscosos e se comporta na forma de lâminas de fluido bem comportadas que escoam de forma paralela sem interpenetração. Este regime é conhecido como *laminar*, onde a lei de Stokes, uma expressão analítica, pode ser usada para determinar uma expressão para a força de arrasto \mathbf{f}_d dada por:

$$\mathbf{f}_d = -6\pi\eta a\mathbf{v}, \quad (1.2)$$

Substituindo a equação (2.55) em (2.56), considerando o movimento unidimensional (apenas na direção z) e reorganizando os termos, podemos escrever a seguinte equação que descreve a evolução (relaxação) da velocidade da esfera:

$$\frac{dv_z}{dt} = -\zeta v_z + \beta, \quad (1.3)$$

em que $\zeta = 9\eta a^2/(2\rho_s)$ e $\beta = \Delta\rho/\rho_s$. Utilizando a seguinte lei de conversão entre quantidades dimensionais e não dimensionais:

$$v_z^* = \frac{v_z}{U_s} \quad \text{e} \quad t^* = tU_s/a, \quad (1.4)$$

em que as quantidades * denotam grandezas não-dimensionais correspondentes e U_s representa a velocidade terminal de uma única partícula sedimentando em baixo Reynolds (velocidade de Stokes), podemos reescrever a equação (2.57) em sua forma não dimensional como:

$$St \frac{dv_z^*}{dt^*} = 1 - v_z^*, \quad (1.5)$$

em que $St = mU_s/(6\pi\eta a)$ representa o número de Stokes, um parâmetro adimensional que mede a razão entre a escala de tempo de relaxação da partícula e uma escala de tempo convectiva equivalente ao tempo que uma esfera em velocidade de Stokes demora para sedimentar o próprio raio. A equação (2.58) possui solução exata pelo método dos fatores integrantes. Para $v_z^*(0) = 0$, a solução exata é dada por:

$$v_z^*(t) = 1 - e^{-t/St}. \quad (1.6)$$

Nesse caso, não precisamos de uma solução numérica para modelar esse problema, pois o mesmo é passível de uma solução exata utilizando apenas papel e lápis. A solução exata ou analítica, é o padrão ouro em ciência. Não se discute com a matemática. Quando um problema físico é corretamente modelado e recai numa equação passível de ser resolvida por métodos puramente matemáticos sem que precisemos recorrer à computadores, temos então uma Lei e isso é muito forte no campo científico. Entretanto, essa não é a realidade de grande parte dos problemas de engenharia, que acabam demandando soluções por métodos numéricos.

Para que fique mais claro como é fácil acrescentar complexidade à problemas simples, continuemos com nossa pequena esfera. Vamos assumir agora que o escoamento induzido pelo deslocamento da esfera ocorre ainda em regime de baixo Reynolds, porém não necessariamente $Re \ll 1$, suponha por exemplo que esse escoamento esteja sujeito a $Re \sim 1$. Nesse contexto, os padrões de movimento do fluido em torno da esfera levam à formação de pequenas estruturas circulares denominadas vórtices, como ilustrados na figura (1.10).

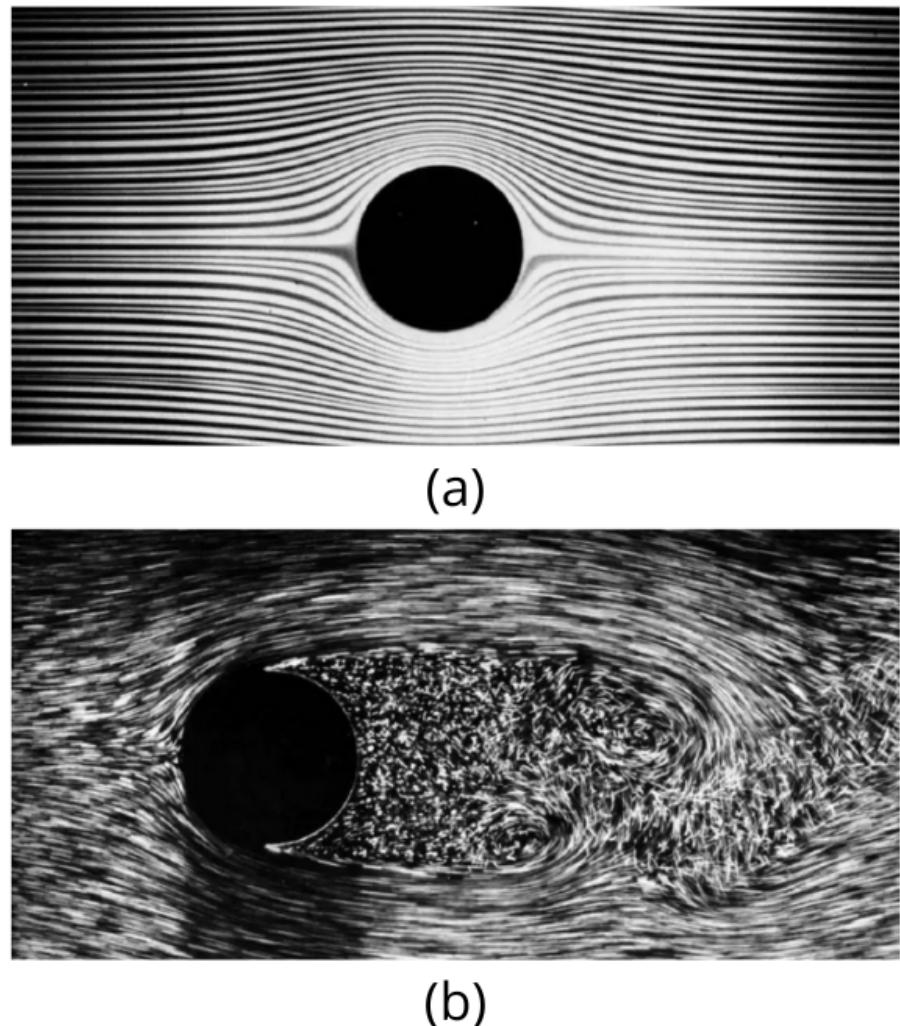


Figura 1.10: (a) escoamento em torno de um cilindro em $Re \rightarrow 0$ e (b) $Re \gg 1$

Esses vórtices interagem com os campos de velocidade e pressão do escoamento nas vizinhanças da esfera e alteram a forma pela qual a esfera percebe as forças viscosas associadas à interação com o fluido.

Nesse limite, podemos adicionar uma correção à força de arrasto de Stokes, proposta por Oseen, ainda restrita a regimes de Reynolds pequenos. Essa força adicional é dada por

$$\mathbf{f}_o = \frac{9}{4}\pi a^2 \rho_f \mathbf{v} |\mathbf{v}|. \quad (1.7)$$

Note que essa força possui agora uma dependência quadrática com relação à velocidade da esfera, diferentemente do cenário linear de um regime de Stokes. Adicionando essa nova força à equação que descreve o movimento da esfera, temos

$$m_s \frac{dv_z}{dt} = -6\pi\eta a v_z - \frac{9}{4}\pi a^2 \rho_f v_z^2 + \frac{4}{3}\pi a^3 \Delta \rho g. \quad (1.8)$$

Utilizando as mesmas escalas típicas do cenário anterior para realização do processo de adimensionalização da equação (1.8) obtemos a versão não-dimensional da nossa nova equação, dada por

$$St \frac{dv_z^*}{dt} = 1 - v_z^* - \frac{3}{8} Re v_z^{*2} \rightarrow Re \lesssim 1. \quad (1.9)$$

A equação (1.9) incorpora agora o número de Reynolds como um parâmetro físico a ser levado em consideração para descrever a dinâmica do nosso probleminha-exemplo. A equação (1.9) apesar de mais complicada que a equação (2.57) ainda possui solução analítica. Essa equação é uma equação de Riccati com coeficientes constantes e pode ser resolvida usando papel e lápis por meio de um processo de transformação de variáveis. Não vamos entrar nos detalhes aqui da solução exata, mas a expressão analítica para a evolução da velocidade da esfera ao longo do tempo é dada agora por

$$v_z(t) = v^* + \left[\frac{Q}{P} + \left(-\frac{1}{v^*} - \frac{Q}{P} \right) e^{-Pt} \right]^{-1} \quad (1.10)$$

onde:

$$\begin{aligned} v^* &= \frac{-1 + \sqrt{1 + \frac{3}{2} Re_s}}{\frac{3}{4} Re_s} \\ Q &= \frac{3}{8 St} Re_s \\ P &= -\frac{3}{4 St} Re_s \cdot v^* - \frac{1}{St} \end{aligned} \quad (1.11)$$

Note agora como a solução exata é dada por expressões muito mais elaboradas do que aquelas obtidas para o caso assintótico de $Re \rightarrow 0$. Ainda assim, papel, lápis e matemática dão conta do recado, de tal sorte que não precisamos apelar para métodos numéricos.

Vamos agora complicar um pouco mais as coisas e incorporar um mecanismo físico adicional. Em 1888, em seu tratado sobre hidrodinâmica, o físico Basset ao analizar as equações de Navier-Stokes, que descrevem o movimento do fluido ao redor da esfera, percebeu que manifestações transientes no fluido seriam responsáveis por um arrasto adicional percebido pela esfera. Esse arrasto transiente foi computado analiticamente por Basset para o caso em que os efeitos iniciais no fluido são desprezíveis e obteve uma expressão integral para essa força de arrasto \mathbf{f}_b dada por

$$\mathbf{f}_b(t) = 6a^2 \sqrt{\pi \rho_f \eta} \int_0^t \left(\frac{d\mathbf{v}}{dt} \right) \frac{1}{\sqrt{t-\chi}} d\chi, \quad (1.12)$$

em que χ representa uma variável de integração. Na verdade, a equação (1.12) estabelece a ideia de que para conhecermos essa força transiente devemos conhecer todo o histórico da aceleração da esfera até o momento em que desejamos saber o valor da força de Basset. A equação (1.12) possui agora uma integral de convolução e ao ser aplicada sobre a equação do movimento da esfera leva a uma equação íntegro-diferencial. Pronto, agora nossa matemática não dá mais conta do recado. Nesse ponto a saída é adotar um método numérico que resolva o problema. Esse problema específico é resolvido no artigo de Sobral et al. (2007) por meio da aplicação do algoritmo de Piccard, um método numérico próprio para tal finalidade.

É interessante notar que um problema tão simples quanto o movimento de uma única esfera em um fluido viscoso, ainda em regime de baixo Reynolds, dependendo das razões envolvidas entre as variáveis físicas do problema logo atinge um limite em que não é mais passível de ser abordado por meio de uma solução analítica. Nesse ponto fica claro por que soluções numéricas são necessárias.

1.5 A importância da validação de uma solução numérica

O probleminha da esfera explorado nessa seção é útil também para que percebamos as intersecções entre métodos analíticos, numéricos e experimentais na proposição de um racional robusto para abordar determinado cenário físico por meio de simulações numéricas.

Podemos sim utilizar uma solução numérica para aprendermos coisas novas sobre o nosso Universo sem necessariamente termos que recorrer a um experimento. Na verdade, quando passamos a utilizar computadores para estudar cenários físicos por meio de simulações computacionais, o método científico, como precursores como Francis Bacon e DaVinci, teve que incluir agora esse novo adendo: a simulação computacional como potencial substituto de um experimento físico. Muito do que a ciência moderna sabe sobre o comportamento de fenômenos fora do sistema solar vem de simulações numéricas. Quando queremos estudar o movimento de uma avalanche em Marte podemos fazer simulações computacionais de materiais granulares, validadas com base em experimentos na Terra e depois estendidas para outros planetas por meio da mudança de alguns parâmetros de entrada na simulação, como a gravidade e as propriedades dos grãos de Marte e pronto, temos uma simulação fidedigna desse comportamento sem precisarmos necessariamente instalarmos uma câmera em Marte para filmar essa avalanche hipotética.

A questão central aqui é que para tudo isso funcionar, precisamos de validação. E a grande pergunta é: como validar os resultados de uma simulação computacional? Uma opção óbvia seria por meio da comparação dos resultados de uma simulação com dados experimentais. Para termos uma ideia, quando começamos a desenvolver modelos de turbulência visando a modificação das equações de Navier-Stokes da mecânica dos fluidos para simularmos escoamentos turbulentos tipicamente industriais em um tempo computacional viável para as máquinas da época, tivemos que definir benchmarks (referências) a serem usados para fins de validação desses novos modelos que começaram a surgir. Um caso clássico que surgiu na época por exemplo foi o escoamento em torno de um degrau (expansão abrupta) que foi utilizado para verificação de como modelos matemáticos implementados com a finalidade de reduzir custos computacionais nesse tipo de simulação afetavam a captura de certos padrões de movimento do fluido sobre a expansão. A partir desse ponto muitos cientistas foram para o laboratório equipados com seus anemômetros para medir sinais turbulentos de velocidade em escoamentos sobre expansões abruptas para que esses dados pudessem ser utilizados como forma de validação de novos modelos de turbulência.

Uma questão interessante que surge é: e quando não é possível realizar um experimento? Ainda assim é possível validar uma simulação computacional? Para respondermos essa pergunta voltemos ao nosso problema da esfera. Nesse caso podemos por exemplo implementar um código computacional que resolva o problema para o caso mais simples de uma esfera sedimentando sob a ação de um arrasto de Stokes não-transiente no limite assintótico $Re \rightarrow 0$ utilizando por exemplo um esquema de Runge-Kutta (que veremos ao longo desse curso). Em seguida poderíamos utilizar a solução analítica por fatores integrantes para validar nossa implementação. Um próximo passo seria implementar no código o efeito por exemplo de um arrasto de Oseen para $Re \sim 1$ e em seguida validar essa implementação com base na solução exata da equação de Riccati aqui apresentada. Esses passos incrementais de validação vão dando confiança para o código que está sendo construído. Mesmo que não tenhamos uma solução analítica para o problema completo com o arrasto de Basset, poderíamos utilizar o mesmo código, porém com essa implemen-

tação adicional para estudar essa física. Teríamos bons motivos para crer que a solução numérica representaria o comportamento físico desse sistema para o caso completo, uma vez que a mesma estrutura de código foi capaz de recuperar limites assintóticos para os quais foi possível desenvolver uma solução analítica.

1.6 Tarefa inicial para aquecer os motores

Baseado nessa contextualização, sua tarefa consiste em escrever um programa de computador (FORTRAN, C++ ou Python) que resolva o problema de sedimentação de uma esfera em baixo Reynolds na sua forma adimensional utilizando o método de Runge-Kutta de quarta ordem clássico para realizar algumas análises. Esse método será visto em detalhes no capítulo dessa apostila referente ao estudo de esquemas numéricos voltados à solução de equações diferenciais ordinárias. Para realizar essa tarefa, apresentaremos aqui apenas a sequência de passos lógicos que constitui a aplicação do método para a solução de uma equação diferencial ordinária num contexto de solução de um problema de valor inicial.

Para uma equação diferencial ordinária do tipo

$$\frac{dy}{dt} = f(t, y),$$

sujeita à uma condição inicial $y(0) = y_0$, podemos construir a solução numérica para obtenção de $y(t)$ por meio das seguintes relações de recorrência:

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h$$

com

$$\begin{aligned} k_1 &= f(x_i, y_i), \quad k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right) \\ k_3 &= f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h\right), \quad k_4 = f(x_i + h, y_i + k_3 h), \end{aligned} \quad (1.13)$$

As análises a serem realizadas são definidas a seguir.

1. Para o caso de $Re \rightarrow 0$ compare a solução analítica com a solução exata para diferentes valores de St ;
2. Para um dado cenário varie o passo de tempo e mostre como o refinamento dessa quantidade afeta a qualidade da solução;
3. Para um pequeno efeito inercial no fluido ($Re \neq 0$) devemos adicionar uma força de arrasto quadrática ao movimento da esfera, de tal sorte que agora a equação governante (dimensional) do problema é dada por:

$$m_p \frac{dv_z}{dt} = -6\pi\eta a v_z - \frac{9}{4}\pi\rho_f a^2 v_z^2 + \frac{4\pi a^3}{3}\Delta\rho g. \quad (1.14)$$

Para esse cenário, adimensionalize a equação do movimento da partícula e mostre que a versão adimensional dessa equação possui além do número de Stokes uma dependência com o número de Reynolds de partícula Re_s baseado na velocidade de Stokes de uma partícula isolada, dado por:

$$Re_s = \frac{\rho_f U_s a}{\eta}; \quad (1.15)$$

4. Para este novo cenário, valide seu código com base na solução exata para o problema, que pode ser encontrada no artigo trabalhado nas aulas iniciais do curso [1];
5. Finalmente, plote o comportamento da solução numérica para diferentes valores de Re_s e mostre como a solução numérica se desvia do limite assintótico em que $Re \rightarrow 0$.