P. Kantek

Curso de Cálculo Numérico

UFPR
2022

Curso de Cálculo Numérico

P. Kantek

28 de janeiro de 2022

O homem só ensina bem o que para ele tem poesia. Rabindranath Tagore (poeta indiano ganhador do prêmio Nobel de literatura em 1913)

Sumário

1 Apresentação e Introdução **9**

2 Conversão de bases **11**

3 Números em computadores **19**
3.1 Inteiros . . . 19
3.2 Inteiros negativos . . . 20
3.3 Reais ou flutuantes . . . 20
3.3.1 Padrão 16 bits original . . . 21
3.4 Otimizações - padrão IEEE754 . . . 21
3.4.1 Polarização . . . 21
3.4.2 Desprezo de d_1 . . . 22
3.4.3 flexibilização da normalização na mantissa . . . 22
3.4.4 Outros formatos . . . 22

4 Erros **25**
4.1 Exemplo de e^x . . . 26
4.2 Truncamento e Arredondamento . . . 27
4.3 Propagação de erros . . . 29
4.4 Outros erros . . . 30

5 Ambientes de computação **33**
5.1 Como evoluiu este estudo . . . 33
5.2 FORTRAN . . . 34
5.3 C++ . . . 35
5.4 Python . . . 36
5.4.1 Winpython . . . 36
5.4.2 Ipython . . . 37
5.4.3 Python na WEB . . . 37
5.5 Maple . . . 37
5.6 MATLAB . . . 38
5.7 OCTAVE . . . 38
5.7.1 Freelab . . . 38
5.8 APL . . . 38
5.9 Em resumo . . . 40

6 Python **41**
6.1 Instalação e alo mundo . . . 42
6.2 Alo Mundo . . . 42
6.3 Variáveis, sua entrada e saída . . . 43
6.4 Atribuição . . . 43
6.5 Tipos . . . 45
6.5.1 Nomes . . . 47
6.6 Expressões aritméticas . . . 47
6.7 Saída de Dados . . . 48
6.8 Entrada de dados . . . 49
6.8.1 Um programa completo usando input e print . . . 50
6.9 Variáveis Lógicas . . . 50
6.9.1 Expressões Lógicas . . . 51
6.10 Comandos condicionais . . . 51
6.11 Repetições . . . 53
6.11.1 while . . . 53
6.11.2 break . . . 53
6.11.3 Repetição aninhada . . . 54
6.11.4 Continue . . . 54
6.11.5 For . . . 55
6.12 Listas . . . 55
6.12.1 Fatiamento . . . 57
6.12.2 Mais sobre condições . . . 57
6.13 Numpy . . . 58
6.14 Sympy . . . 64
6.14.1 Exercício . . . 68
6.15 Um exemplo de plotagem de uma função . . . 69
6.15.1 Resposta . . . 70
6.16 Matplotlib . . . 71

7 Matlab e Octave **83**
7.1 Ambiente do Octave . . . 83
7.2 Gráficos . . . 85
7.3 Programação . . . 87
7.4 Funções anônimas . . . 88

8 Zeros de funções **89**
8.1 Isolamento de raízes . . . 89
8.2 Bissecção . . . 91
8.2.1 Qualidades do método da bissecção . . . 93
8.2.2 Defeitos do método da bissecção . . . 94
8.3 Falsa Posição . . . 95
8.4 Iteração Linear . . . 96
8.5 Newton-Raphson . . . 99
8.5.1 Problemas no uso de Newton-Raphson . . . 102
8.6 Secante . . . 103
8.7 Misto . . . 103
8.8 Mandando o sympy calcular . . . 104

9 Sistemas Lineares **105**
9.1 Equações e Sistemas Lineares . . . 105
9.2 Método de Escalonamento de Gauss . . . 107
9.3 Pivoteamento Parcial . . . 109
9.4 Pivoteamento total . . . 110
9.5 Gauss-Jordan . . . 112
9.6 Fatoração LU . . . 113
9.6.1 Condições de existência . . . 114
9.6.2 Processo de obtenção de LU . . . 114
9.6.3 Determinante eficiente . . . 117
9.7 Usando a matriz inversa . . . 118
9.8 Gauss-Jacobi . . . 126
9.9 Gauss-Seidl . . . 130
9.9.1 Critério de convergência . . . 130
9.10 Relaxação . . . 132
9.10.1 Comparação entre os métodos . . . 133

10 Interpolação **143**
10.1 Polinomial . . . 143
10.2 Lagrange . . . 146
10.3 Newton . . . 149
10.4 Newton-Gregory . . . 153

11 Ajustes de curvas **163**
11.0.1 Regressão linear: mínimos quadrados . . . 163
11.0.2 Regressão múltipla . . . 166

12 Integração **169**
12.1 Integração Numérica . . . 169
12.2 Trapézio . . . 170
12.3 Simpson . . . 173
12.3.1 Uma lista de exercícios . . . 174

13 Bibliografia **179**

O autor, pelo autor

Meu nome é Pedro Luis Kantek Garcia Navarro, conhecido como Kantek, ou Pedro Kantek. Nasci em Curitiba há mais de 60 anos. Sou portanto brasileiro, curitibano e coxa-branca com muito orgulho, mas sendo filho de espanhóis (meus 7 irmãos nasceram lá), tenho também a nacionalidade espanhola. Aprendi a falar em *castellano* era o que se falava na minha casa, o português é meu segundo idioma, só visto na escola. Estudei no Colégio Bom Jesus e quando chegou a hora de escolher a profissão, lá por 1972, fui para a engenharia civil, mas sem muita convicção. As alternativas: medicina (arghhh!) ou direito (arghhh! arghhh!).

Durante a copa do mundo de futebol de 1974 na Alemanha, ao folhear a Gazeta do Povo, achei um pequeno anúncio sobre um estágio na área de processamento de dados (os nomes informática e computação ainda não existiam). Lá fui eu para um estágio na CELEPAR, que acabou mais de 35 anos depois. Na CELEPAR fui de tudo: programador, analista, suporte a BD (banco de dados), suporte a TP (teleprocessamento), coordenador de auto-serviço, coordenador de atendimento, ... Minha carreira lá terminou na área de governo eletrônico. Desde cedo encasquetei que uma boa maneira de me obrigar a continuar estudando a vida toda era virar professor. Comecei essa desafiante carreira em 1976, dando aula num lugar chamado UUTT, que não existe mais. Passei por FAE, PUC e cheguei às Faculdades Positivo em 1988. Na década de 80, virei instrutor itinerante de uma empresa de nome CTIS de Brasília, e dei um monte de cursos por este Brasil afora (Manaus, Recife, Brasília, Rio, São Paulo, Fortaleza, Floripa, ...). Em 90, resolvi voltar a estudar e fui fazer o mestrado em informática industrial no CEFET. Ainda peguei a última leva dos professores franceses que iniciaram o curso. Em 93 virei mestre, e a minha dissertação foi publicada em livro pela editora Campus (Downsizing de sistemas de Informação. Rio de Janeiro: Campus, 1994. 240p, ISBN:85-7001-926-2). O primeiro cheque dos direitos autorais me manteve um mês em Nova Iorque, estudando inglês. Aliás, foi o quarto livro de minha carreira de escritor, antes já havia 3 outros (MS WORD - Guia do Usuário Brasileiro. Rio de Janeiro: Campus, 1987. 250p, ISBN:85-7001-507-0, Centro de Informações. Rio de Janeiro: Campus, 1985. 103p, ISBN:85-7001-383-3 e APL - Uma linguagem de programação. Curitiba. CELEPAR, 1982. 222p). Depois vieram outros. Terminando o mestrado, rapidamente para não perder o fôlego, engatei o doutorado em engenharia elétrica. Ele se iniciou em 1994 na UFSC em Florianópolis. Só terminou em 2000, foram 6 anos inesquecíveis, até porque nesse meio tive que aprender o francês - mais um mês em Paris aprendendo-o. Finalmente virei engenheiro, 25 anos depois de ter iniciado a engenharia civil. Esqueci de dizer que no meio do curso de Civil desistí (cá pra nós o assunto era meio chato...) em favor de algo muito mais emocionante: matemática. Nessa época ainda não havia cursos superiores de informática. Formei-me em matemática na PUC/Pr em 1981. Em 2003, habilitei-me a avaliador de cursos para o MEC. Para minha surpresa, fui selecionado e virei delegado do INEP (Instituto Nacional de Pesquisas Educacionais) do Governo Brasileiro.

O autor, há muitos anos, (mas muitos mesmo) quando ainda havia cabelo...

De novo, visitei lugares deste Brasilzão que sequer imaginava existirem (por exemplo, Rondonópolis, Luiziânia, Rio Grande, entre outros), sempre avaliando os cursos na área de informática: sistemas de informação, engenharia e ciência da computação. Atualmente estou licenciado da PUC. Já fiz um bocado de coisas na vida, mas acho que um dos meus sonhos é um dia ser professor de matemática para crianças: tentar despertá-las para este mundo fantástico, do qual – lastimavelmente – boa parte delas nunca chega sequer perto ao longo de sua vida. Em 2018, comecei a dar aulas na UFPR.

Neste livro, há muito código fonte. Para saber onde implementar, atente para as cores:

Python, Numpy e Matplotlib

Python+Sympy

APL

Matlab ou Octave

C++ ou C

Fortran

Maple

o resto

Faço minhas as palavras de Chapra [Cha13] que diz:

> Embora meu principal interesse seja capacitar estudantes proporcionando-lhes uma boa introdução à resolução de problemas numéricos, eu tenho o objetivo complementar de tornar essa introdução estimulante e prazerosa. Acredito que estudantes motivados, que gostam de engenharia e ciência, resolução de problemas, matemática e também programação, acabarão por se tornar melhores profissionais.

Capítulo 1

Apresentação e Introdução

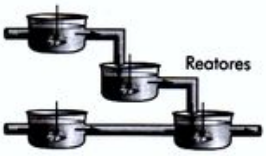
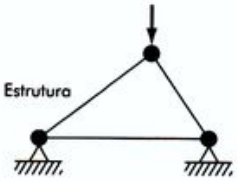
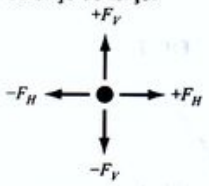
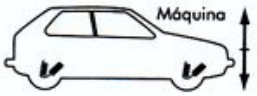
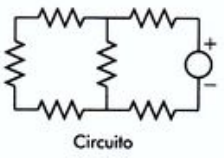
A maior indústria do século XX foi o petróleo. Na virada do século do 20 para o 21, já havia perdido o bastão para a indústria da informática. Hoje, em pleno século 21, a maior empresa do mundo é a Google, seguida pela Microsoft, Facebook, ... A Shell e a Exxon (Esso no Brasil) estão lá atrás na corrida. E não param de andar para trás: a escassez de petróleo e a poluição do planeta são duas espadas de Dâmocles sobre elas e suas parceiras.

A migração do mundo real para o mundo virtual, parece em pleno andamento: nada de novo sob o sol. Platão há uns 2400 anos, já se perguntava: *o universo existe ou é apenas uma manifestação dos meus sentidos ?* Que boa pergunta. É uma tendência inexorável. O uber pulverizou o valor das licenças de taxi. O airbnb reorganiza o negócio hoteleiro. O whatsapp detona com a indústria da telefonia, e o estrago só não é maior por que afinal, a comunicação via tecnologia da informação precisa se dar em algum tipo de estrato (camada) de comunicação.

Seja como for, o objetivo aqui é apresentar ao engenheiro os métodos, as idéias e as ferramentas que permitem resolver os problemas da profissão. As soluções são convertidas em (muitas!) operações aritméticas e lógicas simples. Como os computadores são bons nisso, a idéia subjacente é usá-los extensivamente. Nos tempos pré-computacionais tais métodos já existiam, mas a limitação no cálculo limitava a abrangência e a usabilidade deles. Hoje, a computação abundante nos presenteia com uma *varinha mágica* espetacular. A questão agora é só aprender a usá-la. Uma segunda etapa nessa escalada, é a disponibilidade de pacotes, softwares, linguagens que facilitam ainda mais esse uso. Essa é também uma prioridade em nosso curso. Numa passada citam-se: Matlab e Octave, Python e seus pacotes Numpy, Sympy, Matplotlib. C++ e APL, só para citar alguns.

Só para citar um exemplo simples, do que se disse, pensemos nas leis de conservação. Seu uso implica em sistemas lineares, alguns com milhares de equações e de incógnitas. Vejamos sua omnipresença na física e na engenharia: (figura extraída de [Cha13])

Dispositivos e tipos de balanços que são comumente usados nas quatro áreas principais da engenharia. Para cada caso, a lei de conservação na qual o balanço é baseado está especificada.

Área	Dispositivo	Princípio organizacional	Expressão matemática
Engenharia química		Conservação da massa	Balanço de massa:  Ao longo de uma unidade de período de tempo Δmassa = entradas – saídas
Engenharia civil		Conservação do momento	Balanço de força:  Em cada nó Σ Forças horizontais $(F_H) = 0$ Σ Forças verticais $(F_V) = 0$
Engenharia mecânica		Conservação do momento	Balanço de força: Força para cima $x = 0$ Força para baixo $m\frac{d^2x}{dt^2}$ = força para baixo – força para cima
Engenharia elétrica		Conservação da carga	Balanço de corrente:  Em cada nó Σ corrente $(i) = 0$
		Conservação da energia	Balanço de tensão:  Em torno de cada laço Σ fem's – Σ queda de tensão nos resistores = 0 $\Sigma\,\xi - \Sigma\,iR = 0$

Capítulo 2

Conversão de bases

A aritmética tal como a conhecemos se desenvolveu a partir dos últimos 10.000 anos. Nesse aspecto, a matemática é privilegiada. Estivéssemos falando de ciênciais sociais ou mesmo de tecnologia aplicada, estaríamos tratando do último modismo, ou da última invenção. Mas, a matemática é eterna. Pitágoras e Euclides continuam tão verdadeiros e frescos como eram há 2400 anos. Aliás, se algum dia, descobrirmos uma civilização extraterrena, com biologia completamente distinta, seres com 7 braços e 3 pernas que vivem a bilhões de anos luz da Terra, pode-se garantir: o Teorema de Pitágoras lá continua valendo: $a^2 + b^2 = c^2$ ainda que eventualmente tenha outro nome e outros símbolos, talvez o Pitágoras deles seja um camarão inteligente.

A primeira grande invenção na aritmética foi o número zero (devido a matemáticos indianos). Com ele criou-se um sistema de notação posicional. A base escolhida foi a decimal, muito provavelmente devido a termos 10 dedos, já que estes são os primeiros e muito efetivos instrumentos de contagem de que dispõe o ser humano.

Eis os fatos sabidos até aqui: nosso sistema de numeração é decimal, (base=10), posicional e usa os dígitos arábicos 0,1,...9. Uma curiosidade: embora diversos povos do planeta usem distintos alfabetos para a escrita, (cirílico, katakana, hangul, árabe, grego,...) todos os principais parecem concordar com os dígitos arábicos para escrever números. Assim o número 1003 é 1003 em qualquer rincão culto da Terra.

A mecânica da contagem é usar os 9 primeiros dígitos, começando no um e ao chegar ao 10, que é quando se completa uma base, e se acabam os dígitos unitários, escreve-se o um (o primeiro) seguido de um zero.

Aqui, a primeira novidade. Nenhuma generalidade se perde (e se ganha espaço) se a origem das contagens for o zero e não o um. Dentro da ciência da computação, em muitos momentos a origem da contagem será o zero. Fiquemos espertos.

Quando, os engenheiros ingleses construiram o primeiro computador de que se tem notícia, seguindo os planos teóricos do matemático Alan Turing estiveram diante de uma decisão tecnológica: como representar uma quantidade dentro de uma máquina ?

A primeira hipótese foi fazer uma analogia com um circuito elétrico e variar alguma grandeza física (elétrica) proporcionalmente à quantidade em estudo. Aliás alguns computadores analógicos foram assim construídos. Imprecisão, dificuldades imensas na programação, e baixa generalidade foram fatais a este projeto. O caminho estava em outro lugar: em computadores digitais. Aliás,

esta tendência que começou na década de 40, está hoje entre nós na digitalização de imagens, sons, filmes, livros, raios-x, relógios etc etc.

Ao construir uma variável digital (ao invés de uma analógica) é possível controlar sua exatidão, garantir ausência de erros e obter a precisão tão grande quanto se queira. A pergunta agora é: que tipo de variável ? A tentação é criar uma variável decimal (com 10 estados digitais). Mais fácil de falar do que fazer. Imaginem um circuito que a 0V indica 0, 1V indica 1, ... até 9V indica 9. Basta cair um raio, haver uma interferência e um 7 vira 9 fácil, fácil. A opção foi radical: variáveis binárias, contendo apenas dois valores, e completamente opostos um do outro. Digamos que 0 é +5V e 1 é −5V. Ao cair um raio, um 5V pode virar 4V ou até 3V, mas o circuito é suficiente "esperto" para não se confundir: 3V está muito mais perto de 5V do que de −5V.

Assim, chega-se à aritmética binária: a base de todos os computadores digitais (e mais genericamente de toda a eletrônica digital). Toda a circuitaria de um computador está construída assim. Deixando de lado os detalhes sórdidos (como 5V e −5V) vamos passar a usar a aritmética binária: apenas dois dígitos 0 e 1.

Continuam valendo TODAS as regras da aritmética: adição, produto, subtração, divisão, logaritmo, etc etc.

Acompanhe no exemplo: 0+0=0, 0+1=1, 1+0=1 e 1+1=0 e vai um ou seja, 1+1=10. Resta um problema: o tamanho dos números: Por exemplo, o número $55786803_{(10)}$ é igual a $1101010011001111010011\ 0011_{(2)}$. Um número em base 2 contém em média $log10 \div log2 = 3,32$ algarismos a mais do em base 10. Nada que assuste, principalmente tendo em vista a aumento explosivo dos limites máximos de memória de qualquer circuito atual.

Por tudo isso, a unidade básica de registro (chamado, algo incongruentemente de memória) em um computador é o BIT (BInary digiT), que como vimos pode conter 0 ou 1. Usa-se abreviar o BIT por um "b" minúsculo. Devido à pobreza desse elemento individual, os bits são agrupados em unidades maiores, chamadas BYTES e compostas por 8 bits. A França insiste em chamar isto de octeto, mas sem muito sucesso. Usa-se abreviar o BYTE por um "B" maiúsculo. Deve-se este conceito ao engenheiro americano Claude Shanon, autor da assim chamada *Teoria da Informação*. Se, em um bit só podem ser armazenados dois valores, em um byte, podem-se guardar $2^8 = 256$ estados, devidamente numerados de 0 até 255.

Sistema Hexadecimal Os dígitos binários são tudo o que o computador consegue manusear. Mas, vimos que se torna cansativo e sujeito a erros para o ser humano, manipular cadeias enormes de zeros e uns. A solução para isso, está em usar um segundo sistema, o hexadecimal. A palavra hexadecimal tem a ver com a base=16. Ressalte-se que ele é uma modificação do sistema binário (que insisto: é o único que o computeador entende), criada exclusivamente para facilitar a vida dos humanos que têm que pensar em binário. Essa simplificação consiste em trocar 4 dígitos binários por 1 único dígito hexadecimal. Em um sistema de base 16, vai-se precisar de 16 dígitos. Na falta de melhor opção, vão-se usar os 10 dígitos numéricos (0..9) e depois 6 letras começando no A (A..F).

Antes de mostrar as tabelas, vamos fazer uma contagem:

em decimal 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...

em binário 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100 ,1101, 1110, 1111, 10000, 10001, 10010, 10011, 10100, ...

em hexadecimal 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, ...

Agora, uma tabela de conversão:

decimal	binário	hexad.
0	0000.0000	00
1	0000.0001	01
2	0000.0010	02
9	0000.1001	09
10	0000.1010	0A
15	0000.1111	0F
16	0001.0000	10
255	1111.1111	FF

Perceba-se que dentro de um byte (8 bits) é costume pôr um ponto separando as duas metades do byte. mais ou menos o mesmo que escrever 34.567 ao invés de 34567. Facilita a leitura.

Exercício Construa a tabela acima para os 256 primeiros valores. Comece em 0 e termine em 255.

Conversão de base 2 para base 10 Multiplique cada dígito pelas potências crescente de 2, começando à esquerda com 2^0 e some o resultado. Acompanhe no exemplo:

Quanto é $110100_{(2)} =?_{(10)}$ Faça-se: $0 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 + 1 \times 2^5 = 0 + 0 + 4 + 0 + 16 + 32 = 52 = 52_{(10)}$.

Conversão de base 16 para base 10 A mesma coisa: Multiplique cada dígito pelas potências crescente de 16, começando à esquerda com 16^0 e some o resultado. Acompanhe no exemplo:

Quanto é $02CA_{(16)} =?_{(10)}$ Faça-se: $A \times 16^0 + C \times 16^1 + 2 \times 16^2 + 0 \times 16^3 = 10 \times 16^0 + 12 \times 16^1 + 2 \times 16^2 + 0 \times 16^3 = 10 + 192 + 512 + 0 = 714_{(10)}$.

Conversão de base 10 para base 2 A regra é através da apropriação dos sucessivos restos da divisão inteira do número por 2. Acompanhe no exemplo:

Quanto é $52_{(10)} =?_{(2)}$. Divida-se 52 por 2. O resultado é 26 e o resto é 0. Depois $26 \div 2 = 13$ e o resto é 0. Depois $13 \div 2 = 6$ e o resto é 1. Depois $6 \div 2 = 3$, resto 0. $3 \div 2 = 1$, resto 1. $1 \div 2 = 0$, resto 1. Tomando os restos de trás para a frente, obtem-se o número binário: 110100, que é o número buscado.

Conversão de base 10 para base 16 A regra é a mesma: a apropriação dos sucessivos restos da divisão inteira do número por 16. Acompanhe no exemplo:

Quanto é $714_{(10)} =?_{(16)}$. Divida-se 714 por 16. Dá 44, com resto 10. Depois, $44 \div 16 = 2$, resto 12. Depois, $2 \div 16 = 0$, resto 2. Tomando os restos fica 2.12.10 ou mais propriamente 2CA, que é o número buscado. Para a realização deste tipo de raciocínio, convém decorar a seguinte tabela de potências crescentes de 2 (e de 16)

2^0	1		
2^1	2	1 b	
2^2	4		
2^3	8		
2^4	16		
2^5	32		
2^6	64		
2^7	128		
2^8	256	1 B	10^0 B
2^9	512		
2^{10}	1024	1 KB	$\approx 10^3$
2^{12}	4096		
2^{16}	65536		
2^{20}	1048576	1 MB	$\approx 10^6$
2^{30}	giga	1 GB	$\approx 10^9$
2^{40}	tera	1 TB	$\approx 10^{12}$
2^{50}	peta	1 PB	$\approx 10^{15}$
2^{60}	exa	1 EB	$\approx 10^{18}$
2^{70}	zetta	1 ZB	$\approx 10^{21}$
2^{80}	yota	1 YB	$\approx 10^{24}$

O sistema de numeração é a base da operação de contagem, que vem a ser a origem de toda a matemática (a rainha das ciências, a propósito). Os sistemas de numeração posicionais – a grande contribuição dos indianos e depois dos árabes –, tem como característica a escolha de um número fixo, denominado BASE, que nos sistemas mencionados vale 10. Depois disso, se necessitam esse mesmo número de símbolos distintos, começando sempre pelo ZERO. Assim, tem-se 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9 como símbolos para escrever números.

A interpretação de um número com vários dígitos, embora seja feita de maneira quase automática pelo nosso cérebro, passa por algumas operações:

⋆ a separação dos dígitos. Assim, o número 346, é analisado como formado por 3, 4 e depois 6.

⋆ a identificação da base: neste caso, 10.

⋆ A multiplicação – da direita para a esquerda – de potências crescentes da base, começando com o expoente ZERO. Então:

  – 6 é multiplicado pela base (10) elevada ao expoente 0. Dá $6 \times 10^0 = 6 \times 1 = 6$.
  – 4 é multiplicado pela base (10) elevada ao expoente 1. Dá $4 \times 10^1 = 4 \times 10 = 40$.
  – 3 é multiplicado pela base (10) elevada ao expoente 2. Dá $3 \times 10^2 = 3 \times 100 = 300$.

⋆ As parcelas achadas são somadas. No caso $300 + 40 + 6 = 346$.

⋆ Aqui fica claro a importância do zero. Ele é usado para reservar posições não usadas, a fim de que o número fique univocamente determinada.

⋆ Cada dígito no número passa a ter dois valores: o absoluto e o relativo. Como exemplo, em 346, o 3 tem valor absoluto=3 e valor relativo=300.

No mundo eletrônico-digital que nos cerca (celular, relógio, GPS, tevê, som no carro, computador, I-pxd, cartão bancário, etc, etc, etc), o sistema de

numeração básico e sempre presente é o de base=2. As razões são físico-históricas e têm a ver com confiabilidade, segurança e facilidade. Fazendo a analogia com o visto no sistema decimal, tem-se aqui:

⋆ BASE = 2

⋆ Dois (e apenas dois) dígitos: 0 e 1.

Uma operação simples de contagem neste sistema agora é: 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, 10000 e assim por diante. Nenhuma heresia foi cometida, todas as regras da numeração foram seguidas.

Faça as seguintes conversões:

De base 2	para base 10
101	
1000	
111	
1001	
101010	
110001	

Deve-se estudar e treinar o caminho oposto: dado um número decimal, digamos 18, como obter o equivalente binário ? A regra aponta para divisões sucessivas pela base, até que o quociente e o resto sejam zero.

⋆ Pega-se o número a transformar (18) e divide-se-o pela base (2) Fica: $18 \div 2 = 9$ e resto 0.

⋆ Divide-se o quociente obtido (9) pela base (2). Fica: $9 \div 2 = 4$ e resto 1.

⋆ Idem e fica: $4 \div 2 = 2$ e resto 0.

⋆ Idem e fica: $2 \div 2 = 1$ e resto 0.

⋆ Idem e fica: $1 \div 2 = 0$ e resto 1.

⋆ O número binário é obtido recolhendo-se os restos de baixo para cima: 1, 0, 0, 1, 0.

⋆ Finalmente, tem-se $18_{10} = 10010_2$.

Note que, sempre que necessário, convenciona-se escrever a base à direita do número na forma de um subscrito. Faça as seguintes conversões:

De base 10	para base 2
6	
9	
11	
15	
31	
33	

Números fracionários também podem ser convertidos. Basta extender a regra de conversão à direita da vírgula decimal. Veja um exemplo: O número decimal 10,5 pode ser entendido como $1\times10^1+0\times10^0+5\times10^{-1} = 10+0+0,5 = 10,5$.

Suponha-se agora o número binário 101,101. Quanto ele vale em decimal? $1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3} = 4+1+0,5+0,125 = 5,625_{10}$.

Para sua orientação, eis uma tabela das potências negativas de 2.

2^{-1}	0,5
2^{-2}	0,25
2^{-3}	0,125
2^{-4}	0,0625
2^{-5}	0,03125
2^{-6}	0,015625
2^{-7}	0,0078125
2^{-8}	0,00390625

Acompanhe o processo de converter um número de base 10 fracionário em seu equivalente binário. Por exemplo, o número $8,375_{10}$ em binário. Tudo começa pela parte inteira, que é convertida como já se sabe. Então $8_{10} = 1000_2$. A parte fracionária (0,375) é convertida sendo multiplicada sucessivamente por 2.

Então: $0,375 \times 2 = 0,750$. O 0 inteiro é o primeiro dígito binário após a virgula, e o processo recomeça.

Então, $0,750 \times 2 = 1,5$. O 1 inteiro é o segundo dígito e o processo recomeça com 0,5.

Então $0,5 \times 2 = 1,00$. O 1 inteiro é o terceiro dígito e o processo termina já que a parte fracionária é zero.

Ao final, tem-se que $0,375_{10} = 0,011_2$ e o número completo fica $8,375_{10} = 1000,011_2$.

Pode ocorrer, ao longo deste processo da parte fracionária reaparecer com o mesmo valor. Este fato indica uma dízima periódica e continuando com o cálculo vai-se produzir um número infinito. Tal fato deve ser indicado por meio de reticências, parando-se a seguir o cálculo. Para ver isto na prática, faça a conversão de $4,8_{10} =?_2$.

Outra observação importante é que este processo normalmente gera números muito grandes. O leitor pode abortar o processo tão logo uma precisão aceitável seja alcançada.

Base octal. É a numeração de base 8, muito usada em eletrônica, sobretudo em circuitos com palavras de 6 bits. Usa os números 0, 1, 2, 3, 4, 5, 6 e 7. Acompanhe a contagem neste sistema: 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, ..., 77, 100, 101, ...

Para converter de base octal para base decimal, multiplica-se cada dígito pelas potências crescentes da base 8. Por exemplo, $144_9 =?_{10}$. Tem-se $4 \times 8^0 + 4 \times 8^1 + 1 \times 8^2 = 4 + 32 + 64 = 100_{10}$.

Exercícios: 1. converta o número 14_8 em decimal; 2. Converta o número 67_8 em decimal; 3. Idem para 153_8 e para 1552_8. 4. Por que o número 15874_8 não pode ser um número octal?

A conversão de octal para binário é simples: Cada dígito octal é traduzido em 3 bits, incluindo-se os zeros à esquerda, se necessário.

Exercícios: Converta para binário os seguintes octais: 1. 34_8, 2. 536_8 e 3. 44675_8.

A conversão de binário para octal exige a separação do número binário em grupos de 3 bits, eventualmente completando-se zeros à esquerda. Cada grupo de 3 bits gera um número octal.

A conversão de decimal para octal pressupõe a divisão sucessiva por 8, obtendo-se os restos até que o último quociente seja zero. Por exemplo, seja converter 92_{10} para octal. Divide-se 92 por 8, obtendo-se 11 e resto 4. Daí divide-se 11 por 8 obtendo-se 1 e resto 3. Finalmente, divide-se 1 por 8,

obtendo-se 0 e resto 1. A conversão é 134_8. Um processo alternativo aparentemente mais trabalhoso, mas que tem lá seu charme é converter o número a binário antes e a octal depois.

Capítulo 3

Números em computadores

Qualquer máquina digital (computadores, celulares, fornos de microondas, impressoras, I-pads, automóveis, aviões, brinquedos, máquinas fotográficas, geladeiras, televisores, máquinas de dinheiro,...) precisa manusear e armazenar números. Embora lá nos primórdios do século XX tenha havido a tentação de representar números por medidas analógicas, rapidamente a qualidade da representação digital se impôs.

Neste ponto estamos nos afastando um pouco da matemática e nos aproximando da engenharia e da tecnologia. E precisamos fazer este movimento, para entender porque muitas coisas são como são dentro das máquinas.

Todas elas usam lógica binária e portanto usam bits (0 ou 1) internamente. Em geral, para guardar um número, todas trabalham com quantidades fixas de bits, o que ao fim e ao cabo implica em uma quantidade finita de números representáveis. Todos os demais números são obtidos por proximidade a algum número desses que é representável, gerando aqui um erro de representação. Por exemplo, 2, seu quadrado e o número 4 podem ser representados em um computador usando Python. Já, a raiz quadrada de 5 não pode (e quando obtida é forçosamente arredondada) daí:

```
>>> 2**2 == 4
True
>>> ((5**0.5)**2) == 5
False
>>> '%2.24f'%(((5**0.5)**2)-5)
'0.000000000000000888178420'
```

3.1 Inteiros

Os inteiros podem ser corretamente representados, limitados ao tamanho do **registro** assim entendido como a unidade de bits que são alocados ao armazenamento de um número. Supondo n bits, pode-se guardar até o número $2^n - 1$. Por exemplo, se $n = 4$ pode-se guardar até o 15. Não acredite em mim, faça as representações para se convencer. Se $n = 8$, guarda-se até o 255. Se $n = 16$ o limite agora é 65535, e assim por diante. Note-se que esta representação é para inteiros sem sinal (ou positivos). Se, se necessitar guardar números negativos também, perde-se um bit (o mais à esquerda) para o sinal e portanto o maior número passa a ser $2^{n-1} - 1$ e o menor, que lá no caso

sem sinal era o zero, agora é -2^{n-1}. Por exemplo, se $n = 8$ o intervalo de representação é $-127..0..127$. Convenciona-se usar o bit 0 para positivo e o bit 1 para negativo e daqui:

$$\begin{aligned}
11111111 &= -(2^6 + ... + 2^1 + 2^0) = -127 \\
&... \\
10000001 &= \text{-1} \\
10000000 &= \text{-0 (?)} \\
00000000 &= 0 \\
&... \\
01111111 &= 2^6 + 2^5 + ... + 2^0 = 127
\end{aligned}$$

As memórias têm crescido muito nos últimos anos, e os tamanhos agora são bem generosos. Por exemplo, em C++, inteiros podem ser de 8 ou 16 bits (já visto antes) mas também de 32 bits (limite de 4.294.967.296) ou de 64 bits (limite de 18.446.744.073.709.551.616) ou a metade disso se houver sinal. A menos de algum detalhe obscuro, todas as plataformas de computadores compartilham estes valores.

3.2 Inteiros negativos

Os negativos inteiros são representados como complemento para 2. Esta operação naturalmente levará o primeiro bit a ser 1 (positivos têm primeira posição igual a zero). Mas a representação do número é um pouco diferente do que seria esperado. A regra é simples: Troque zeros por uns e vice-versa. Some 1. Pronto. Eis um exemplo: Suponha o número $(1011)_2$ como expressão de um negativo (1^o bit=1). Trocando uns-zeros fica (0100). Somando 1 fica (0101) que corresponde a 5. Como o primeiro bit era 1, então -5. Acompanhe na tabela de 4 bits:

Binário	Hexad	Decimal	Compl. 2
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	8	-8
1001	9	9	-7
1010	A	10	-6
1011	B	11	-5
1100	C	12	-4
1101	D	13	-3
1110	E	14	-2
1111	F	15	-1

3.3 Reais ou flutuantes

A dificuldade inicial é como representar o intervalo $-\infty..\infty$ em um número limitado e fixo de bits. Agora os bits disponíveis terão que ser distribuídos em

diversos componentes do número a representar, a saber

$$\pm d_1 d_2 ... d_n \times \beta^{\pm e_1 e_2 ... e_m}$$

Considerando que a base β é fixa na arquitetura, ela não precisa ser guardada a cada número. Restam os dígitos $d_1 d_2 d_n$ junto com o seu sinal, além dos dígitos $e_1 e_2 ... e_m$ também com o seu sinal. O conjunto $d_1 d_2 ... d_n$ é conhecido como mantissa do número e está normalizado entre 0 e 1. Em outras palavras, o número 127.345 seria representado por 0.127345×10^3.

3.3.1 Padrão 16 bits original

Esta representação clássica nem é mais usada, mas vai-se começar com ela por razões didáticas. Eis como funcionava um sistema em ponto flutuante, com base binária ($\beta = 2$), com $t = 10$ digitos binários na mantissa e expoentes limitados entre $I = -15$ e $S = +15$, lembrando que $(15)_{10} = (1111)_2$ e portanto o registro é

s_1	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	s_2	e_1	e_2	e_3	e_4

s_1 sinal do número (da mantissa). $0 \rightarrow$ positivo e $1 \rightarrow$ negativo

$d_1..d_{10}$ mantissa com 10 dígitos binários significativos

s_2 sinal do expoente: $0 \rightarrow +$ e $1 \rightarrow -$

$e_1..e_4$ expoente

Nessa representação os números são armazenados no formato normalizado (entre 0 e 1) logo, $d_1 \neq 0$ e portanto o número é

$$v = (-1)^{\pm s_1} (0.f)_2 \ 2^{\pm exp}$$

Nesse esquema, o menor valor representável é `0.1000000000.11111` e este valor é $+(0.1)_2 \times 2^{-15} = (2^{-1} \times 2^{-15})_{10} = (2^{-16})_{10} = 0.0000152587890625$.

Já o maior valor é `0.1111111111.01111` Que vem a ser $(32736)_{10} \approx 2^{15}$.

Finalmente o zero é `0.0000000000.11111`. Lembrando que este é o único número não normalizado, pois sua mantissa é toda zero. Todo número menor que o menor valor representável acarreta *underflow* enquanto todo número maior que o maior representável acarreta *overflow*.

3.4 Otimizações - padrão IEEE754

Algumas características do exposto foram alteradas para ampliar a faixa de representação e organizar a representação. Primeiro, a mantissa e o expoente têm seus locais invertidos.

3.4.1 Polarização

A idéia é inventar um valor fixo e somá-lo a todos os expoentes, transformando um número de 4 bits mais sinal em um número de 5 bits. Neste caso (16 bits) o valor de polarização é 15 e agora passa-se de $-15..15$ a $0..31$. Isto é possível porque usando o sinal, tinham-se duas representações para o zero ($+0e-0$) e isto é desnecessário. Agora o número é visto como

$$v = (-1)^s (0.f)_2 \ e^{e-15}$$

3.4.2 Desprezo de d_1

Não se armazena o d_1 que como se viu vale sempre 1. Agora a mantissa representada passa a ter os dígitos $d_2, d_3, ...d_{t+1}$ e a posição normalizada do ponto passa a ser à direita de $d_1 = 1$ ou seja

$$v = (-1)^s (1.f)_2 \ 2^{e-15}$$

3.4.3 flexibilização da normalização na mantissa

O explicado acima tenderia a levantar o valor da menor representação possível. Para evitar isso, quando $e = 00000$ a normalização do $d_1 = 1$ é eliminada. Agora o menor valor possível é `0.00000.0000000001` que é igual a `0.0000000596046448`. O padrão ficou assim:

s	e (5d)	f (10d)
1	2..6	7..16

O número aqui armazenado obedece a algumas regras:
se $0 < e < 31$ então $v = (-1)^s \ 2^{e-15} \ (1.f)_2$
se $e = 0$ e $f \neq 0$ então $v = (-1)^s \ 2^{-14}(0.f)$
se $e = 0$ e $f = 0$ então este número é zero
se $e = 31 = 2^5 - 1$ então v pertence à região de overflow.

3.4.4 Outros formatos

Esta especificação foi levada a outros formatos, a saber

A	B	C	D
formato	exp	exp_{10}	mant
binary16	5 dig	31	10 d
binary32	8 dig	255	23 d
binary64	11 dig	2047	52 d
binary128	15 dig	32767	112 d

Exemplo Ache os valores negativos (por complemento a 2) das variáveis

```
1 1 0 1 1 0 1 1 0 1 1 0 0 1 0 1
1 1 1 0 1 0 1 0 1 1 1 1 0 1 0 0
```

São eles: -9371 e -5388

Agora, ache os flutuantes, com 7 casas decimais, de

```
1 1 0 0 1 1 0 0 0 1 0 0 0 0 1 0
0 1 1 0 0 0 0 1 1 1 0 0 1 0 1 0
0 1 1 0 1 0 0 0 0 1 0 1 1 0 0 1
1 0 1 0 1 1 1 1 1 0 0 0 1 0 1 0
```

As respostas são -17.0312500, 741.0000000, 2226.0000000 e -0.1177978

Exemplos em Octave

O Octave possui algumas funções nativas relacionadas a representação interna de números (que lembrando são efetuadas em modo 64 bits)

```
>> format long
>> realmax
ans =   1.797693134862316e+308
```

```
>> realmin
ans =   2.225073858507201e-308
>> eps
ans =    2.220446049250313e-16
```

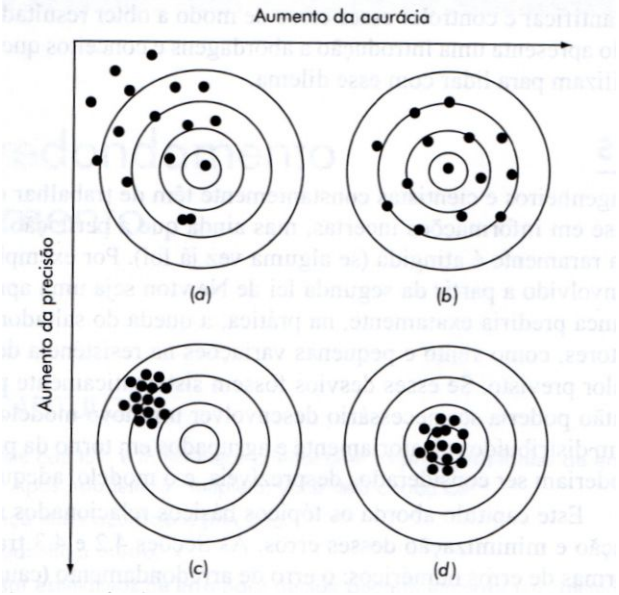
Vale a lembrança de que números guardados em computadores possuem uma característica interessante: o intervalo de registro é fixo em números inteiros (daí a sua característica de *contabilidade*) e possuem intervalos proporcionais à magnitude do número representando em representações flutuantes. Por exemplo, a diferença entre 0.7 e 0.8 é de 0.1. Já a diferença entre 0.7×10^4 e 0.8×10^4 é de 0.1×10^4 que vem a ser de 1000 vezes a unidade. Como se diz em [Jus18], *números não estão igualmente espaçados no eixo real. Números pequenos estão bem próximos, enquanto números com expoentes grandes estão bem distantes (pág. 22)*. Tenha isto em mente quando operar com estes números.

Capítulo 4

Erros

Tomamos decisões com base em dados levantados no mundo real. Uma característica que estes dados sempre tem é a sua incerteza. Ainda que a perfeição seja uma meta louvável, ela raramente é atingida (se é que alguma vez o foi). Este capítulo aborda os temas relacionados à identificação, quantificação e minimização de erros.

Começa-se definindo acurácia e precisão. A acurácia informa quão próximo do valor real está o valor calculado ou medido. A precisão informa quão próximos os valores calculados ou medidos estão uns dos outros. Uma ilustração gráfica pode ser feita com uma analogia com a prática de tiro ao alvo. Os buracos de bala em cada alvo representam as previsões de uma técnica numérica. A mosca no centro representa a verdade. A inacurácia (ou viés) é definida como um desvio sistemático da verdade. (A figura a seguir é extraída de [Cha13]).



Assim, tanto as figuras (a) como (c) estão centradas no quadrante esquerdo superior. Os dois casos são igualmente inacurados. As imprecisões (ou incertezas) referem-se à intensidade do espalhamento. Portanto as figuras (b) e (d) são igualmente exatas, já que centradas na mosca, mas (d) é mais precisa porque os tiros estão mais agrupados. A maneira de lidar com estes erros é considerar a posição do centro da distribuição de dados (medida do viés) e o

grau de dispersão dos dados (precisão).

Aqui o erro representa tanto a inacurácia quanto a imprecisão. Os erros numéricos são causados pelo uso de aproximações para representar operações e quantidades exatas. Pode-se formular como

$$\textit{valor verdadeiro} = \textit{aproximação} + \textit{erro}$$

Ou

$$\textit{erro} = \textit{valor verdadeiro - aproximação}$$

Um defeito desta formulação é que ela não leva em consideração a ordem de grandeza do que está sendo calculado ou medido. Por exemplo, um erro de um metro pode ser irrelevante em uma distância de centenas de quilômetros, mas certamente será inaceitável na largura de uma ponte. Corrige-se isto normalizando o erro com o valor verdadeiro e fica

$$\textit{erro relativo} = \textit{(valor verdadeiro - aproximação) / valor verdadeiro}$$

Uma dificuldade é que no mundo real dificilmente o valor real é conhecido. Aliás, se o for, toda esta discussão perde o sentido. Então, em lugar do valor verdadeiro, usa-se a melhor aproximação disponível.

Por exemplo, muitas técnicas numéricas usam uma abordagem iterativa para calcular a resposta. Ou seja a aproximação atual é feita com base em uma estimativa prévia. O processo é repetido para obter aproximações cada vez melhores (é o que se espera) e o erro aqui é estimado como a diferença entre as aproximações prévia e atual. Este procedimento é bastante semelhante ao conceito de limite na análise matemática.

O sinal do erro pode ser tanto positivo quanto negativo. Então o critério de parada pode ser $|erro - obtido| < erro - admitido$ usando-se a magnitude para cancelar o sinal do erro.

4.1 Exemplo de e^x

Acompanhe um exemplo: Na matemática algumas funções podem ser representadas por séries infinitas. Neste caso tem-se a série de Maclaurin para e^x, que é

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + ... + \frac{x^n}{n!}$$

e quantos mais termos são adicionados à série, mais próxima a soma fica do valor verdadeiro de e^x. Começando com a versão mais simples de $e^x = 1$ soma-se um termo de cada vez para obter $e^{0.5}$. Depois disso, calcula-se o erro observando-se que o valor verdadeiro é $e^{0.5} = 1.648721...$ Deve-se prosseguir até que o erro aproximado ε_a esteja dentro do critério especificado que é ε_s que garanta três algarismos significativos.

Primeiro, deve-se calcular o critério de erro de 3 algarismos significativos que é

$$\varepsilon_s = (0.5 \times 10^{2-3})\% = 0.05\%$$

Portanto, deve-se adicionar termos à série até que o erro caia abaixo deste nível.

O primeiro valor (1 termo) é $e^{0.5} = 1$. Como o erro considera 2 cálculos, precisa-se obter o segundo e $e^{0.5} = 1 + 0.5 = 1.5$ e o erro aproximado é

$$\varepsilon_a = \left|\frac{1.5 - 1}{1.5}\right| \times 100\% = 33.3\%$$

Neste caso em particular, como se conhece o valor verdadeiro, o erro real ε_r também pode ser calculado e ele é

$$\varepsilon_r = \left|\frac{1.648721 - 1.5}{1.648721}\right| \times 100\% = 9.02\%$$

Como ε_a não é menor do que ε_s deve-se prosseguir adicionando o novo termo $x^2/2!$ e repetindo os novos cálculos de erro até que $|\varepsilon_a| < \varepsilon_s$. O resultado obtido pode ser sistematizado em

termos	resultado	$\varepsilon_r(\%)$	$\varepsilon_a(\%)$
1	1	39.3	
2	1.5	9.02	33.3
3	1.625	1.44	7.69
4	1.645833333	0.175	1.27
5	1.648437500	0.0172	0.158
6	1.648697917	0.00142	0.0158

Depois que 6 termos são incluídos o erro aproximado cai abaixo de $\varepsilon_s = 0.05\%$ e os cálculos podem ser encerrados. Pode-se observar que em vez dos 3 algarismos corretos, eles o são até o quinto. Isto porque as equações são conservadoras – garantem que o resultado obtido é no mínimo tão satisfatório quanto se especifica.

A implementação computacional deste tipo de cálculo geralmente envolve um laço (*loop*) que se encerra por uma condição lógica. Acompanhe este cálculo de e^x usando uma função Python

```
# função e^x
import math
def eex(x):
  sol_ant=x**0/math.factorial(0)
  sol_atu=sol_ant+x**1/math.factorial(1)
  iteracao=2
  epsilon=0.00005
  while abs((sol_atu-sol_ant)/sol_atu)>epsilon:
    sol_ant=sol_atu
    sol_atu=sol_atu+(x**iteracao)/math.factorial(iteracao)
    iteracao=iteracao+1
  print('número de iteracoes',iteracao)
  return sol_atu
print('a resposta e = 'eex(0.5))
>>>
número de iteracoes 7
a resposta e = 1.6487196180555554
```

4.2 Truncamento e Arredondamento

A diferença conceitual entre estes dois tipos de erro é fácil perceber: Suponha que um determinado valor teve seu resultado calculado como 0.0002 e precisa ser guardado com uma mantissa de comprimento 3 ou 0.000|2. Aqui o erro de truncamento é de 0.0002 já que o valor será armazenado como 0.000. Não

há erro de arredondamento. Se houvesse arredondamento ele seria para baixo resultando 0.000 também. Este erro – de arredondamento –, apareceria se o cálculo tivesse dado 0.000|8, por exemplo. Neste caso, antes de truncar o número ele seria arredondado para cima (já que 0.8 é maior que a metade: 0.5) e ficaria 0.001 cometendo-se o erro de arrendondamento de 0.0002, pois 0.001 - 0.0008 = 0.0002. Estes erros surgem porque os computadores não podem representar certas quantidades de forma exata na solução de problemas. O ponto chave é que não importa quantos bits se utilizem para representar um número, este número sempre é fixo. Ora, por definição o intervalo $(-\infty..\infty)$ possui infinitos números. Daqui surge a questão: muitos desses números terão que ser arredondados para serem representados. A questão é que projetistas e cientistas bolaram estratégias para minimizar este erro, mas ele sempre está lá.

Veja-se a propósito um exemplo em Python

```
# pequeno
def peq():
  s=0
  for i in range(10000):
    s=s+0.0001
  return s
print(peq())
```

Sua execução, que por considerações de ordem aritmética deveria dar 1, dá 0.9999999999999062

Outra origem deste erro é quando ocorre a conversão de um número de uma base para outra, e numa delas, o número é uma dízima periódica.

Um exemplo trivial: Seja operar com $0.1_{(10)}$ em binário. Precisamos então converter $0.1_{(10)} \rightarrow ?_{(2)}$. Para fazer isso, recordemos a regra: A parte inteira é convertida à parte. Depois toma-se a parte fracionária e multiplica-se sucessivamente por 2. A cada etapa a parte inteira do resultado (1 ou 0) é agregada depois do ponto e o processo reinicia apenas com a parte fracionária do resultado ATÉ que a parte fracionária seja zero.

No exemplo $0.1_{(10)}$ sua parte inteira é 0. A conversão de $0_{(10)}$ para binário é $0_{(2)}$. Então a parte inteira é 0 e vamos para a parte fracionária: $0.1 \times 2 = \underline{0}.2$ e o 0 da parte inteira do resultado vai para o nosso resultado: $0\underline{0}$. Continuando com o resultado $0.2 \times 2 = \underline{0}.4$ e este zero vai para o resultado: $0.0\underline{0}$. Continuando com o resultado: $0.4 \times 2 = \underline{0}.8$. Fica $0.00\underline{0}$. Agora $0.8 \times 2 = \underline{1}.6$ e o resultado agora é: $0.000\underline{1}$ e continua: $0.6 \times 2 = \underline{1}.2$ O resultado agora é $0.0001\underline{1}$ e continua-se: $0.2 \times 2 = \underline{0}.4$. O novo resultado é $0.00011\underline{0}$. Podemos continuar até a eternidade e o resultado será algo como 0.00011001100110011001100... Como este número tem comprimento infinito, ele terá que ser truncado (interrompido) em algum momento, já que a parte fracionária da multiplicação nunca será zero.

Outra fonte de erro é como o computador opera com flutuantes. Numa soma, por exemplo, a primeira coisa a fazer é alinhar os pontos decimais. Aliás, humanos também fazem isso. Por exemplo, para somar 18000 com 0.00067, antes precisamos alinhar os pontos:

```
18000.0
    0.00067
-----------
```

```
18000.00067
```

Em termos flutuantes a coisa fica 0.18×10^5 e 0.67×10^{-3}. Mas, para poder somar os números os expoentes têm que ser igualados: ou 0.0000000067×10^5 e daí

```
     0.18
     0.0000000067
-----------------
     0.1800000067
```

O problema ocorre quando ao expandir o menor número sua parte significativa caia após o tamanho da mantissa. Num exemplo hipotético se o computador usa-se apenas 4 dígitos de mantissa, o operando seria truncado e o resultado da soma seria apenas 0.180000 e com isto a segunda parcela sumiria...

4.3 Propagação de erros

Se se fazem cálculos com números não exatos, então o resultado do cálculo deve ter seus erros também. A questão agora é como o erro de cada número individual se propaga para o resultado obtido ? (baseado em [Kaw17]).

Veja-se um exemplo: Qual o erro na adição de dois números ? Suponha calcular $X + Y$ se $X = 1.5 \pm 0.05$ e $Y = 3.4 \pm 0.04$.

Olhando os números percebe-se que o valor máximo de cada um é $X = 1.55$ e $Y = 3.44$. Daqui $X + Y = 1.55 + 3.44 = 4.99$ que é o valor máximo para $X + Y$. Já os valores mínimos de X e Y são $X = 1.45$ e $Y = 3.36$, daqui $X + Y = 1.45 + 3.36 = 4.81$. Portanto $4.81 \leq X + Y \leq 4.99$.

Pode-se encontrar intervalos similares para as operações $X - Y$, $X * Y$ e X/Y. A questão agora é como encontrar erros propagados quando há funções envolvidas. Se $f(x_1, x_2, ...x_n)$ o valor máximo do erro em f é

$$\Delta f \approx \left|\frac{\partial f}{\partial x_1}\Delta x_1\right| + \left|\frac{\partial f}{\partial x_2}\Delta x_2\right| + ... + \left|\frac{\partial f}{\partial x_n}\Delta x_n\right|$$

Acompanhe o exemplo: A deformação em uma sessão cruzada quadrada de um determinado material é dada por

$$\in = \frac{F}{h^2.E}$$

onde F =força axial sobre o material em N, h =comprimento da sessão em m e E = módulo de Young em Pa. Dados
$F = 72 \pm 0.9N$
$h = 4 \pm 0.1mm$
$E = 70 \pm 1.5GPa$
Ache o erro máximo na medida da deformação.
Solução:

$$\in = \frac{72}{(4 \times 10^{-3})^2(70 \times 10^9)}$$

$= 64.286 \times 10^{-6}$
$= 64.286\mu$

$$\Delta \in \approx \left|\frac{\partial \in}{\partial F}\Delta F\right| + \left|\frac{\partial \in}{\partial h}\Delta h\right| + \left|\frac{\partial \in}{\partial E}\Delta E\right|$$

$$\frac{\partial \in}{\partial F} = \frac{1}{h^2 E}$$
$$\frac{\partial \in}{\partial h} = \frac{2F}{h^3 E}$$
$$\frac{\partial \in}{\partial E} = -\frac{F}{h^2 E^2}$$
$$\Delta \in = \left|\frac{1}{h^2 E}\Delta F\right| + \left|\frac{2F}{h^3 E}\Delta h\right| + \left|\frac{F}{h^2 E^2}\Delta E\right|$$
$$= \left|\frac{1}{((4\times 10^{-3})^2(70\times 10^9)} \times 0.9\right| + \left|\frac{2\times 72}{(4\times 10^{-3})^2(70\times 10^9)} \times 0.0001\right|$$
$$+ \left|\frac{72}{(4\times 10^{-3})^2(70\times 10^9)^2} \times 1.5\times 10^9\right|$$

$= 8.0357\times 10^{-7} + 3.2143\times 10^{-6} + 1.3776\times 10^{-6}$
$= 5.3955\times 10^{-6}$
$= 5.3955\mu$
$\in = (64.286\mu \pm 5.3955\mu)$ implicando que a deformação axial $\in$ está entre 58.8905μ e 69.6815μ. Em [Kaw17], capítulo 01.06, há um exemplo interessante do que ocorre quando dois números muito próximos são subtraídos, e como um erro enorme aparece aqui.

4.4 Outros erros

Além dos erros intrínsicos à computação acima estudados, há outros a espreita:

Enganos Estes erros costumam ser subestimados, mas eles sempre estão por aí: trata-se do erro grosseiro, aquele que deixa seus autores com as orelhas vermelhas de vergonha. Usando computadores, há 50 anos, eles podiam ser atribuídos à maquina. Nos dias de hoje, é muito mais provável que sejam de autoria humana. Só é possível evitá-los com conferências e uma profunda desconfiança do resultado (de qualquer resultado).

Formulação Erros atribuídos a modelos matemáticos inválidos ou incompletos. Um exemplo, é o uso das fórmulas de Newton sem atentar para os efeitos relativísticos envolvidos. Às vezes, a simplificação excessiva responde por este tipo de erro. Novamente, eles só são percebidos e corrigidos com verificações e contra-provas extensivas.

Incerteza nos dados Qualquer medida, seja com qual instrumento for, está sujeita a erros de medida. Não importa quão bem feita seja a medida. Os instrumentos de medida são inadequados (Qual a medida da costa brasileira ? Depende do tamanho da régua...). Para lutar contra estes erros, uma saída é a multipla avaliação, usando instrumentos (e pessoas) diferentes. Outra é o tratamento estatístico prévio ou posterior dos dados.

Conspiração do universo Vale citar a lei de Murphy: *Se alguma coisa pode dar errado, dará.*

A propósito, não tem a ver com cálculo numérico, mas sim com a onipresença do erro: quando um sistema tem um custo de erro inaceitável (em vidas ou mesmo em dinheiro), a ciência da computação tem técnicas para lidar: está-se falando de "sistemas tolerantes à falha". Na indústria

de software equivale a desenvolver o mesmo pacote por 2 equipes diferentes em plataformas (linguagem, computador, sistema operacional, ...) diferentes. São exemplos: as naves recuperáveis, aviões, ...

Capítulo 5

Ambientes de computação

Introdução a métodos numéricos

Um cientista do século XXI precisa dominar e usar uma grande quantidade de modelos matemáticos para propor e resolver inúmeros problemas. Muitas vezes estes problemas derivam de princípios da ciência ou da engenharia, mas muitas vezes eles são obtidos como dados experimentais. Modelos matemáticas incluem mas não estão limitados a:

1. equações não lineares
2. sistemas de equações lineares (possivelmente grandes)
3. ajuste de curvas por interpolação ou regressão
4. diferenciação e cálculo de otimização via máximos e mínimos
5. integração
6. equações diferenciais

Nessa altura da vida (possivelmente dentro de um curso de ciências ou de engenharia) você deve estar se perguntando porque reaprender essas coisas que já foram exaustivamente vistas e resolvidas em cursos de cálculo e de álgebra linear. A razão surge de algumas considerações: Uma, é que a ênfase lá era a solução simbólica ou algébrica enquanto agora a ênfase é numérica. Se lá a solução era exata, aqui é aproximada, até porque sempre prenhe de erros. Mais ainda, tais características praticamente exigiam números redondos (e pequenos). Tudo o que um problema real não tem. Por exemplo, você consegue reconhecer em

$$12670420.88x^2 + 56369x - 1908765.33$$

uma simples equação do segundo grau ?

Este fatos nos levam à segunda questão. Em pleno século XXI que se caracteriza por abundantes soluções de hardware (computadores) e de software (linguagens, pacotes, sites,...) obviamente tais soluções numéricas são obtidas e geradas por manipulações automatizadas (i. é informatizadas). Nesse sentido este estudo passa por aprender e apreender as técnicas inseridas em ambientes computacionais amigáveis e eficazes.

5.1 Como evoluiu este estudo

Nos anos 60 do século passado, as universidades buscavam se servir da nascente computação: um caso real, aqui na nossa UFPR, quando eu fiz esta disciplina no longínquo ano de 1975. Em meados dos anos 60, criou-se o NC=Núcleo de

Computação, o qual após longas tratativas foi equipado com um IBM 1130, uma pujante CPU com a capacidade de memória de 8K.

Nos anos 70 era essa máquina que era usada pelos alunos de métodos numéricos. Ela era programada em Fortran. Muita coisa mudou de lá para cá. Hoje um celular dos baratos tem memória de 8GB ou cerca de 1.000.000 vezes mais. O FORTRAN perdeu a primazia e dezenas de ambientes mais adequados estão disponíveis.

O uso de computadores pessoais é mandatório em nossos dias para qualquer profissional. A questão é qual ambiente escolher para a nossa disciplina.

5.2 FORTRAN

O único benefício desta linguagem é a sua antiguidade. Isto implica em muita literatura (Tércio Pacitti e Leonidas Barroso et alli, entre outros) e mesmo pacotes de software. Mas, é um ambiente com cheiro de mofo, embora tenha sido evoluído ao longo do tempo.

Fortran é uma linguagem muito antiga, originalmente sugerida pela IBM na década de 50. Seu texto se assemelha a uma descrição matemática e de fato, FORTRAN significa FORmula TRANslation. É, ou pelo menos era, uma linguagem franciscana de tão pobre em recursos. Foi a única linguagem ensinada em escolas de engenharia até a década de 80. O primeiro compilador de FORTRAN foi desenvolvido para o IBM 704 em 1954-57 por uma equipe da IBM chefiada por John W. Backus.

A inclusão de um tipo de dados de número complexo na linguagem tornou a linguagem Fortran particularmente apta para a computação científica.

As principais revisões são a Fortran 66, 77, 90 e 95. A mais famosa e usada é a Fortran 90. Sua principal vantagem é a eficiência em calculo numérico pesado.

A seguir, um exemplo de FORTRAN note-se que já é um Fortran revisto (e moderno): basta olhar o formato do comando IF (antes era muito mais idiossincrático e obscuro).

```
C        1         2         3         4         5         6
C234567890123456789012345678901234567890123456789012345678901234567890123456789012345
      PROGRAM BASKHARA
C
      REAL  A,B,C, DELTA, X1,X2, RE, IM
C
      PRINT *, "Este programa resolve uma equação de 2o.grau"
      PRINT *, "do tipo: a*x**2 + b*x + c = 0"
C
      PRINT  10, "Digite a,b,c: "
 10   FORMAT(A,1X,$)
 20   READ(*,*,ERR=20) A,B,C
C
      DELTA=B*B-4.*A*C
C
      IF (DELTA.GT.0) THEN      ! (DUAS RAIZES REAIS)
         X1=(-B-SQRT(DELTA))/(2.*A)
         X2=(-B+SQRT(DELTA))/(2.*A)
         PRINT *, "RAIZES:  X1=",X1
         PRINT *, "         X2=",X2
      ELSE IF (DELTA.EQ.0) THEN ! (DUAS RAIZES REAIS IGUAIS)
         X1=-B/(2.*A)
         X2=X1
         PRINT *, "RAIZES: X1=X2=",X1
      ELSE                      ! (DUAS RAIZES COMPLEXAS)
```

```
         RE=-B/(2.*A)
         IM=SQRT(-DELTA)/(2.*A)
         PRINT *, "RAIZES COMPLEXAS: X1=",RE," -",IM,"i"
         PRINT *, "                  X2=",RE," +",IM,"i"
      ENDIF
C
      END
```

Os problemas do Fortran:

- os comandos têm colunas específicas para começar e terminar
- as entradas e saídas dependem de obscuros comandos $format$ imediatamente antes
- os comandos precisam (ou não) serem numerados
- o comando if no começo era expressão, desvio para negativo, desvio para zero e desvio para positivo. Felizmente esse pesadelo foi modificado.
- o tipo da variável era determinada pela primeira letra do nome. Aqui a origem de índices com nome I.

5.3 C++

Possui a vantagem de ter sido usado na disciplina de programação aqui na UFPR. Nesse sentido, continuar seu uso poderia ser encarado como uma sequência na disciplina de programação. Mas, como se verá a seguir, existem ambientes com inúmeras *toolbox* (em programação, chama-se toolbox a uma coleção de módulos prontos, capazes de resolver problemas padrão. É uma biblioteca de programas prontos, bastando conhecê-los, escolhê-los e usá-los) confiáveis e abrangentes. Além do que C++ pela sua proximidade com a máquina exige o conhecimento de inúmeros detalhes e idiossincrasias. Mas, se alguém quiser encarar o desafio, vale o mesmo ambiente da disciplina de programação: o ambiente CODEBLOCKS.

C++ é uma extensão da linguagem C, corrigindo algumas de suas deficiências originais (como por exemplo, a incapacidade de C de generalizar o tipo de dado em uma estrutura particular), além de incluir a manipulação de objetos e a simplificação de algumas tarefas (como entrada/saída padrão, por exemplo). Mas, C++ se saiu muito bem por herdar uma compatibilidade com o legado. (Um programa fonte C já é um programa fonte C++).

A seguir, o mesmo programa feito em C e em C++. Compare e tire suas conclusões

```
#include<stdio.h>
int main(){
   long p,q,r;
   while (scanf("%ld %ld", &p, &q) != EOF) {
      if(q>p) r=q-p;
      else r=p-q.
      printf("%ld\n",r);
   }
}
```

Agora a mesma coisa em C++

```
#include<iostream>
void main()
   long long a,b,c;
   while (cin>>a>>b)
      if(b>a)
     c=b-a;
      else
     c=a-b.
      cout<<c<<endl;
```

5.4 Python

De toda a lista a seguir, Python é o ambiente mais novo. Isto traz a vantagem dele ter sido projetado já com certa abundância de recursos (memória e processador) com a consequente contrapartida de facilidade de uso. A premissa de Python era e é ser aprendido com facilidade. Hoje é usado por mais de 70% das universidades do mundo. É freeware e possui filosofia de criação de pacotes: há mais de 120.000 deles. Vamos estudar e usar 3 muito importantes:

NUMPY: É o pacote numérico (científico). Usado primordialmente para construir matrizes, mas também disponibiliza inúmeras ferramentas da álgebra linear, entre inúmeras outras coisas.

SYMPY: Pacote simbólico, vale dizer: algébrico. Permite definir funções e mandar o pacote achar integrais, derivadas, limites, pontos extremos, entre milhares de outras coisas.

MATPLOTLIB: Desenhador de gráficos. Simplifica a tarefa de desenhar gráficos, o que como veremos, é muito útil para estudar comportamento de fenômenos. A facilidade de uso é o ponto chave aqui.

Para a instalação do Python, a versão mais nova deve ser buscada em www.python.org. Depois de instalada a linguagem é conviente instalar os 3 pacotes acima (comando PIP INSTALL xxx, onde xxx é o nome do pacote) emitido no diretório SCRIPTS da instalação da linguagem. Algumas versões de Python (como por exemplo WINPYTHON) já tem tudo isso instalado e ainda pode ser baixado e usado em um pendrive.

5.4.1 Winpython

Esta implementação aparece aqui por algumas razões, sendo a principal a **portabilidade** do ambiente de desenvolvimento:

- Permite usar Python em máquinas nas quais você não é o administrador, coisa que o Python padrão exige para ser instalado.
- Como consequência, este Python é portável, e pode ser carregado em um pendrive ou similar, ainda que haja alguma demora no processo de carga: o winpython é enorme.
- Winpython é um ambiente científico, pelo que já vem com SciPi e Numpy devidamente instalados, além de eventualmente outros pacotes importantes.

5.4.2 Ipython

É um python interativo, no qual fica mais evidente o diálogo entre usuário e Python. Na sua versão original (Python) é uma linguagem de programação que conta com um enfoque interativo também. No Ipython, ao contrário, é um ambiente interativo, que por acaso também dá acesso a uma linguagem de programação. Muitas vezes associado ao Jupyter, que é um ambiente interativo construído originalmente para as linguagens JUlia, PYThon E (and) R. Posteriormente o ambiente foi migrado para muitas outras linguagens e hoje o menú é bem extenso. O ambiente é bom pois permite desenvolver notebooks que são arquivos simulando um caderno de notas mesclando texto, figuras e código.

Ele aparece aqui nesta lista pois é a versão disponibilizada nos laboratórios do curso de engenharia elétrica da UFPR.

Neste ambiente, eis o caminho das pedras:

1. Usar seu login para entrar no sistema.
2. Na tela de atividades, digitar IP e aguardar. Escolher Ipython.
3. Carregar também o editor GEDIT. Manter ambos abertos. (Ipython e GEDIT)
4. Escrever o código python no GEDIT e salvar o programa e sua execução no diretório /eletrica, com o nome de aa.py
5. Dentro do Ipython executar `%run aa.py`
6. Para corrigir, edite no GEDIT, salve e reexecute dentro do Ipython.

5.4.3 Python na WEB

Uma alternativa disponível a quem tenha acesso à web e não pretenda (ou não possa) instalar nada em sua máquina é usar um ambiente python disponível na web. Há vários, mas um funcional é `https://www.pythonanywhere.com`. Aqui você precisa criar um login e depois disso passa a acessar um ambiente python completo. Já estão instalados nele vários pacotes importantes (teste: numpy, sympy, matplotlib, django). Este ambiente é perfeito para uma consulta rápida.

5.5 Maple

O criador do ambiente simbólico na computação numérica. É um conjunto de 5 coisas: um processador numérico, simbólico, um editor de textos matemáticos, uma linguagem de programação e um gerador de gráficos. Muito poderoso e fácil de usar, mas com um grave inconveniente: não é livre e seu preço é bem salgado. Uma licença estudantil por 1 ano custa cerca de 300 reais. Uma licença plena custa cerca de 1000 dólares. Ainda que o preço não fosse uma dificuldade e tanto, há o fato de que os problemas prosseguem no uso profissional após o curso, já que as empresas também não podem fazer contrafação. [1]

[1] pirataria

5.6 MATLAB

Conhecido como Laboratório de Matrizes, também é um produto comprado (e também caro). A versão estudantil sai por preço similar ao MAPLE e a mesma dificuldade no uso após o curso prossegue. A vantagem aqui é que MATLAB é usado por inúmeras escolas de engenharia em todo o mundo. Além disso há farta bibliografia (o livro do Chapra) e mesmo inúmeros programas disponibilizados nesta plataforma.

5.7 OCTAVE

Alternativa freeware (portanto gratuita e livre) do MATLAB. Não tem toda a coleção de bibliotecas (outros dos diferenciais de Matlab, por exemplo é famosa a biblioteca de redes neurais do ML) mas para o nosso propósito é completamente funcional. Pode ser baixado em
`https://www.gnu.org/software/octave/download` bastando seguir as instruções. O Matlab e o Octave tem a facilidade da computação simbólica. No caso do Octave ela é dependente de Python e de Sympy. Então, é melhor seguir pelo original...

5.7.1 Freelab

Outra alternativa ao MATLAB, também livre e gratuíta. No entanto parece que os esforços de desenvolvimento deste ambiente têm diminuido ao longo do tempo. Parece mais adequado usar o Octave que está indo de vento em popa.

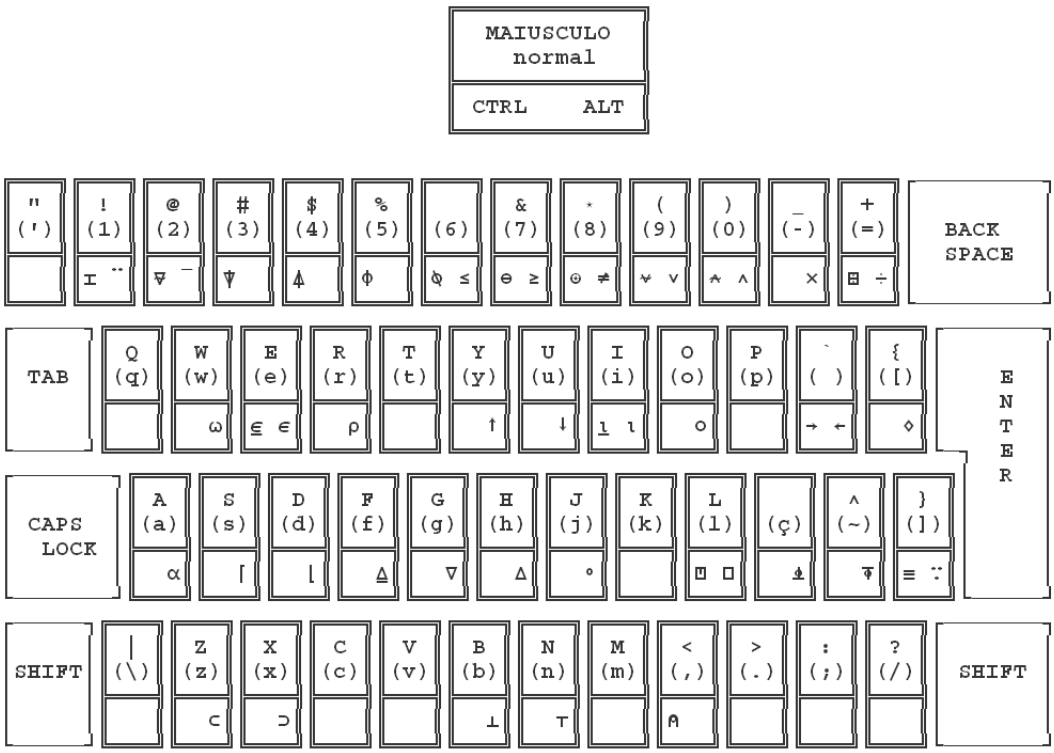
5.8 APL

Esta é uma família especial de linguagens. Lançada originalmente em 1966 sempre esteve (e ainda está) além do seu tempo. Seu diferencial é fazer muito com pouco código. Existem versões gratuítas como o APLX disponível em `https://www.dyalog.com/aplx.htm`, além do NAPL disponível em `http://www.nars2000.org/` Já a versão DIALOG, que é uma das mais completas e robustas, pode ser usada em modalidade não comercial, sem nenhum pagamento, bastando baixar a versão diretamente no site do fabricante em `https://www.dyalog.com`.

O APL nasceu do trabalho de um professor de matemática canadense de nome Keneth Iverson. Sua proposta original era a de produzir uma nova notação matemática, menos sujeita às ambiguidades da notação convencional.

Na década de 60, trabalhando na IBM em conjunto com Adin Falcoff, ambos produziram a primeira versão de APL, quando um interpretador da linguagem ficou disponível.

A principal característica de APL é o uso de um conjunto especial de caracteres que incluem algumas letras gregas (rho, iota...), símbolos matemáticos convencionais (o sinal de vezes, o de dividido...) e alguns símbolos especialmente inventados. Eis uma possível distribuição de tais símbolos em um teclado ABNT



Este fato sempre limitou a disseminação da linguagem. Até o advento das interfaces gráficas, como o windows, por exemplo, exigia-se um hardware especial para poder programar em APL.

Programas em APL em geral sempre são muito pequenos, embora poderosos. A linguagem está preparada para tratar arranjos de grandes dimensões. Por exemplo, quando em APL se escreve A+B, se A e B forem escalares (isto é um número único), a resposta também o será. Se A e B são vetores de 100 números, a resposta também o será. Idem para matrizes e até arrays-nd. Em algumas versões de APL este n chega a 256 dimensões. Aliás, o Python copiou as *ufuncs* e as *broadcasting* daqui.

Este é um comando EDIT (troca algo, de... para...)

```
      ∇edit[⎕]∇
    ∇
[0]   r←depara edit x
[1]   r←,x
[2]   r[(r=1↑depara)/⍳⍴r]←1↓depara
[3]   r←(⍴x)⍴r
    ∇ 2018-06-24 10.03.49 (GMT-4)
      x←3 3⍴?9⍴4
      x
2 1 3
1 3 3
2 2 2
      2 99 edit x
99  1  3
 1  3  3
99 99 99
```

5.9 Em resumo

A sugestão é Python com sua família (NUMPY, SYMPY e MATPLOTLIB entre outros). Eis as vantagens

- uso no mainline da comunidade (a primeira no tiobe de dez/21: era a terceira em dez/20), deixando para trás java, C, C++ ...
- freeware
- linguagem de programação completa: a principal ferramenta de aprendizagem de máquina do Google está feita em Python (o tensorflow)
- fácil de aprender e usar

Capítulo 6

Python

A pergunta: **Porque Python ?**

- É uma linguagem moderna (nasceu em 1990) e herdou tudo de bom que as demais linguagens já tinham nessa época. Uma coisa que ela não herdou foi a preocupação exigente de desempenho: Em 90, já havia computação abundante. Com isso a linguagem é redonda e não impõe trancos e barrancos para funcionar. [1]

- Multiplataforma: Unix, Windows, Apple, celular android, raspberry e outros ambientes menos famosos rodam exatamente o mesmo programa.

- Freeware: pode copiar, baixar, instalar, vender, ceder... pode fazer qualquer coisa sem infringir a lei e sem dever explicações a ninguém.

- Uso no mainstream: Olhem o que diz o Philip Guo, um blogueiro da CACM "oito dos top-10 Departamentos de Ciência da Computação e 27 dos top-39 Departamentos de Ciência da Computação das principais universidades nos EUA usam Python nos cursos introdutórios de CS0 e CS1" (CS0 e CS1 são os cursos introdutórios de Computer Science lá). As 39 universidades pesquisadas por ele incluem: MIT, Berkeley, Stanford, Un. Columbia, UCLA, UIUC, Cornell, Caltech, Un. Michigan, Carnegie Mellon, entre outras [2]

- Muito Usada: o índice TIOBE [3] lista o Python como a quinta linguagem mais usada no mundo em 2017. . Em junho de 2018, Python já é a quarta linguagem mais usada. Perde apenas para java, c e c++. E, corre por fora... Em dezembro de 21 é a linguagem mais usada de todas. A campeã.

- Pacotes, pacotes, pacotes. A comunidade Python mantém o PYPI (Python Package Index) que é um repositório freeware com pacotes de programação: hoje ele tem mais de 110.000 pacotes. Só para ter uma idéia, um pacote destes, o NUMPY (numerical computation in Python) tem um manual de referência de mais de 1500 páginas.

[1] Em compensação, quando se organiza uma maratona de programação, para um determinado problema, impõe-se 1 segundo de CPU quando ele é resolvido em C++ e pelo menos 10 (10 vezes mais) para quando ele é resolvido usando Python

[2] https://cacm.acm.org/ blogs/blog-cacm/176450 -pyhon-is-now-the -most-popular-introductory-teaching-language-at- top-u-s-universities/ fulltext, acessado em agosto/2017

[3] www.tiobe.com, acessado em agosto/17e em dezembro de 21

- Inteligência Artificial: Quando o Google resolveu aposentar o DistBelief e criar uma nova ferramenta de Machine Learning, escolheu a linguagem Python para desenvolvê-la: nasceu o Tensor Flow, que está por trás de inúmeras iniciativas google: tradução, classificação de imagens, elaboração de perfis etc. [4] Logo na entrada do produto o aviso: pode usar em Java, C++ ou Go. Mas, se quiser usar a funcionalidade completa do produto, use-o em Python, que ele está feito nessa linguagem.

- Fácil: parece portugol, ou vá lá: inglêsgol. Não tem caracteres sobrando: sem ponto e vírgula, sem chaves nem colchetes. O que se escreve, ele executa e raramente dá erro.

6.1 Instalação e alo mundo

Python é freeware, não custa nada, você pode copiar e usar para o que desejar, sem pedir licença a ninguém, e pode inclusive cobrar pelo software que você desenvolver usando Python. A partícula *free* em freeware significa liberdade para fazer o que você quiser sem nenhum tipo de restrição e não gratuidade, como alguns pensam.

Existem versões de Python para quase todos os sistemas operacionais conhecidos, eis alguns: AIX (da IBM), AROS (Amiga), AS/400, HPUnix, Linux, MacOS, MS-DOS, OS/2, PalmOS, Playstation, Solaris, VMS, Windows (32 e 64), WindowsCE, entre outros.

Esta multiplicidade de versões acrescenta uma qualidade imensa aos programas Python: desde que eles não usem nenhum recurso específico de uma dessas arquiteturas, serão 100% (ou quase isso) portáveis podendo rodar o mesmo programa em todas essas plataformas.

6.2 Alo Mundo

Tradicionalmente o primeiro programa que um novato deve construir em qualquer novo ambiente é chamado `Alo Mundo`. Sua função é apenas mostrar esta mensagem e sua execução com sucesso sinaliza que o processo de criar um programa fonte, salvá-lo, compilá-lo e depois executá-lo foi bem sucedido. Vamos a isso, então:

1. na tela do Python, escolha `File`

2. no menu que vai aparecer, escolha `New File` e lembre que estes dois passos podem ser comandados apertando `Ctrl N`

3. Vai-se abrir uma nova tela em branco, muito parecida com a tela do Python. Parece Python, mas não é Python e sim o bloco de notas do Python[5]

4. Na tela do bloco de notas do Python escreva o seguinte

```
print("alo mundo")
```

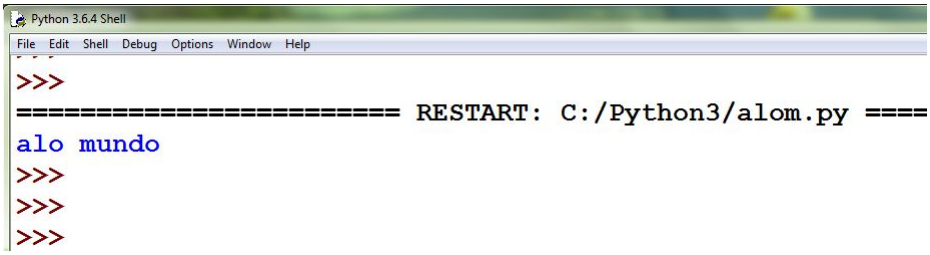
[4] www.tensorflow.org

[5] Uma maneira enviezada de identificar qual tela é qual é esta: o bloco de notas do python tem o comando Run enquanto a tela do Python não tem este comando.

5. Agora você vai comandar a compilação (interpretação) e execução desse script (tecnicamente isto não é um programa e sim um script: é uma diferença semântica meio irrelevante, mas será estudada depois). Quem faz isto é a tecla `F5`. Ou também o comando `Run`, já destacado.

6. O Python antes de qualquer coisa vai pedir para você um nome sob o qual o programa (ou script) vai ser salvo. Esta é uma excelente exigência do Python: garante que aconteça o que acontecer, o programa que você escreveu e está tentando executar, será salvo em disco, sem risco de que ele se perca, caso o pior (tela azul da morte, por exemplo ou mesmo apagão da copel) aconteça. O nome que você der terá agregado o sufixo `.py` e será salvo no lugar onde você informar [6]

7. Tendo sido salvo e compilado com sucesso a tela do Python vai mostrar



   Esta tela mostra a execução do script que você criou e a mensagem sendo mostrada.

8. Isto significa que você teve sucesso na instalação do python e na criação e execução do seu primeiro programa.

6.3 Variáveis, sua entrada e saída

Uma variável é um objeto do mundo do software que pode ter algumas abordagens. A primeira, é aquela da matemática, quando se escreve $y = f(x)$ e se quer dizer que y é uma variável que depende do valor de x. O que existe aí, nessa expressão, é uma função que associa o valor de y ao valor de x. Tanto x quanto y nesse enfoque são variáveis: isto é são valores eventualmente desconhecidos, mas que têm um valor qualquer. Outra abordagem é da ciência da computação: nela, uma variável é um pedacinho de memória dentro de um programa (que está devidamente registrado nessa memória) e que armazena um determinado valor. Cada variável precisa ter um nome, para que possa ser referenciada. Esse nome precisa ser único (não se pode ter duas variáveis com o mesmo nome), e depois que ela foi criada, usar essa variável num comando qualquer é exatamente a mesma coisa que usar o valor que a variável tem. Por exemplo, se a variável A tem o valor 10, tanto faz escrever $A+1$ como $10+1$, só que no primeiro caso, a generalidade é maior, pois independentemente do valor que ela tiver, depois de executar $A+1$, ter-se-á um valor uma unidade maior do que o valor anterior de A.

6.4 Atribuição

O comando importante aqui é o chamado **atribuição** . É ele que permite:

[6] Sugestão para quem está começando: compre e use um pendrive para guardar seus programas: eles irão com você. E não esqueça de fazer cópias deles – o famoso backup.

- criar uma variável
- alterar o valor de uma variável

As duas funções só se diferenciam pelo existência prévia da variável. Se ela não existe, usar a atribuição significa criar a variável. Se ela já existia (e portanto tinha um valor qualquer) usar a atribuição nela vai alterar o valor dela para este, que é citado agora na atribuição.

O formato do comando é

```
variável = valor
```

Onde `variável` é o nome da variável, `=` identifica o comando de atribuição e `valor` é o valor que a variável passará a ter depois desta atribuição. Por exemplo:

`A=10`	A variável de nome A é criada (ou alterada) e passa a ter o valor 10.
`X=-33.4`	Agora o valor de X é o número negativo -33.4
`NM='alfa'`	A variável agora se chama NM e seu valor é uma palavra, que é 'alfa'.

Atente para a presença das aspas, elas serão vistas com mais detalhe adiante. Algumas observações importantes. O caráter de atribuição é = e ele é lido como `recebe`. Então, o comando $A = 10$ deve ser lido como `A recebe 10`. Outra observação é que este sinal tem outras funções no mundo. Por exemplo, na matemática o sinal de igual é isso mesmo: para indicar uma igualdade. Igualdade não é a mesma coisa que atribuição. O problema é que se não se usar o igual, que outro símbolo pode-se usar ? Outras linguagens resolveram esse problema, usando dois caracteres (por exemplo, Pascal usa `:=`) enquanto outras usaram caracteres novos, como por exemplo APL, que usa ←.

A discussão de qual símbolo usar para a atribuição (que lembrando é um conceito que não existe na mátemática) é antiga e poderia ser aberta aqui. Mas, pegando carona no `C`, linguagem avô de boa parte do leque de linguagens usadas no século XXI, (tais como C++, Java, PHP, entre outras), o Python também associou o sinal de = à atribuição. A pergunta é: o que usar quando se falar sobre igualdade entre dois valores ? Neste caso, usam-se 2 caracteres iguais. Então, para perguntar se as variáveis A e B têm o mesmo valor, escreve-se $A == B$, já que se se escrever $A = B$, o que se estará mandando fazer é atribuir o valor da variável B para a variável A.

Voltando ao formato do comando de atribuição, acima descrito, vamos estudar o que pode ser `valor` citado no formato do comando. Como o nome sugere, `valor` pode ser um número como 13, 2780.4, -33 e assim por diante. Mas, também pode ser outra variável, como em `A=X`. Aqui se está mandando copiar o valor de `X` para a variável `A`. Note que não se sabe que valor é esse. Não importa, ele é copiado de `X` para `A`.

Finalmente, esse valor pode ser uma expressão aritmética como em `A=11+8`, ou em `A=V-3`, ou finalmente `A=cos(theta)`. No primeiro caso, `A` vale 19, no segundo vale o valor de `V` menos 3 e no terceiro caso `A` vale o coseno do angulo `theta`.

Agora pode-se ver a importância da presença das aspas no comando de atribuição: acompanhe no exemplo `A='alfa'` e `A=alfa`. No primeiro caso, é o conteúdo `alfa` que está sendo atribuído à variável A. No segundo pressupõe-se a existência de uma variável de nome `alfa` e é esta variável que tem seu valor (seja ele qual for) copiado para a variável `A`.

As aspas denotam um conteúdo alfanumérico, que estruturalmente é diferente de um conteúdo puramente numérico. Há muitas diferenças que ainda

serão estudadas, mas uma importante é que conteúdos (e variáveis) numéricas podem ser usadas em expressões aritméticas enquanto variáveis alfanuméricas causam erros de execução quando se tenta envolvê-las em expressões numérico-aritméticas.

`>>> A=5.5`	Cria-se a variável A, flutuante, com o valor 5.5
`>>> B='oba'`	Cria-se B alfanumérica contendo 'oba'
`>>> A+B`	Tenta-se somar as duas variáveis. Há um erro

O erro é sinalizado pela mensagem `TypeError: unsupported operand type(s) for +: 'float' and 'str'` cuja tradução é: Erro de tipo: não se pode somar (+) um tipo float com um tipo str.

6.5 Tipos

Graças a essa distinção vista acima, pode-se pontuar a respeito de uma característica importante das variáveis. Cada variável terá um tipo que sinalizará como ela será manipulada depois de criada. Os principais tipos são: numérico inteiro, numérico real, alfanumérico e lógica.

Ao contrário de outras linguagens (como o C, por exemplo), as variáveis em Python não têm um tipo explicitamente declarado. É o seu valor que determina qual é o seu tipo. [7] Então,

`A=10.3`	A é uma variável do tipo real, ou `float`
`B=5`	A variável B é do tipo inteiro.
`C=10.0`	A variável C é tipo float
`HAL='ivo viu a uva'`	A variável HAL contém a frase `ivo viu a uva`
`GA=True`	GA é uma variável do tipo lógico que vale verdadeiro
`J=False`	J é uma variável do tipo lógico e tem o valor Falso

A variável do tipo lógico só admite dois valores `True` e `False`. Obviamente, estes dois nomes não servem para dar nomes a variáveis. É isto que se chama de palavra reservada .

Os tipos numéricos podem ser inteiros (`int`) ou reais, que no Python recebem o nome de `float`. Poder-se-ia argumentar que o tipo inteiro é desnecessário (já que o conjunto dos reais contém o conjunto dos inteiros ou $\mathbb{R} \subset \mathbb{Z}$), mas isto não é verdade, já que:

- Dentro do computador, a matemática inteira é dezena de vezes mais rápida que a matemática real. Então quando aplicável, sempre vale a pena ser mais eficiente.
- Mas, principalmente, o conjunto dos inteiros tem uma característica que o real não tem: a enumerabilidade. Tal característica é muito importante (por exemplo, na indexação) e ela exige que o tipo seja inteiro em certos momentos.

Feita esta distinção, os conteúdos que uma variável numérica pode ter são:

inteiro : os 10 dígitos `0,1,2,3,4,5,6,7,8 e 9`, um sinal opcional `+ ou -`.

[7] Esta regra nem sempre é verdadeira, por exemplo, ao usar o pacote Numpy. Mas, por enquanto pode-se aceitá-la como verdadeira.

flutuante : os mesmos dígitos e sinal e além destes, o sinal de ponto decimal. Note que usa-se em Python o ponto decimal e não a vírgula decimal, herança dos países de língua inglesa. Opcionalmente pode-se ter a letra `E` ou `e` significando exponencial. E, se presente, indica o expoente de 10, que deve ser multiplicado pelo número antes do E. O expoente também pode ter opcionalmente um sinal que pode `+` ou `-`. Exemplos:

`3E4` Significa 3×10^4 ou 3000
`-4E2` -4×10^2 ou -400
`5E-3` 5×10^{-3} ou $5 \times \frac{1}{10^3}$ ou ainda 5×0.001 que é 0.005.

Em alguns momentos da programação, é necessário explicitamente modificar um tipo de variável. Esta necessidade já vai aparecer (ao tratar o comando `input`, por exemplo), mas ela vai ser vista aqui do ponto de vista formal. Para mudar o tipo de uma variável pode-se usar

`int(x)`	Pega o conteúdo de x e transforma-o (se possível) em um inteiro.
`float(x)`	Transforma x em flutuante
`str(x)`	Transforma x em uma string

Obviamente a transformação só vai acontecer se ela for possível. Não adianta, por exemplo, definir `A='alfa'` e pedir `int(A)`, já que 'alfa' não pode ser transformada em um número inteiro. Mas, se `B='123'` deve-se notar que B é uma variável string (por causa das aspas), mas ela pode ser transformada em inteira e neste caso, valerá 123, pois só é formada por dígitos numéricos. Já a conversão em string é sempre possível, e sempre que for ordenada será realizada.

Para a representação de números inteiros o Python usa um sistema de precisão ilimitada, que permite usar muitos (muitos !) dígitos, veja-se por exemplo o fatorial de 50 é
`30414093201713378043612608166064768844377641568960512000000000000`. O programador não precisa fazer nada (ao contrário de outras linguagens mais antigas como C ou C++). O Python simplesmente lança mão de quantas casas forem necessárias para a representação numérica inteira pedida.

Já na representação flutuante, podem surgir problemas derivados do fato de que o Python usa para representação interna um formato binário limitado composto de abcissa e mantissa. Em outras palavras, um número flutuante usa uma quantidade finita de bits para ser representado, o que faz com que haja perda de precisão nessa representação. Veja-se um exemplo prático (uma sessão real de Python)

```
>>>   a=0.1
>>>   print(":2.5f".format(a))
>>>   print("%2.5f" % a)
0.10000
0.10000
>>>   print(":2.30f".format(a))
>>>   print("%2.30f" % a)
0.100000000000000005551115123126
0.100000000000000005551115123126
```

O comando format acima, parece, e é, bem misterioso, mas o que ele pede é algo prosaico, que a variável a seja impressa, no primeiro caso, com 5 decimais

e no segundo com 30 decimais, em ambos no *old* e no *new styles*. Observe o que aconteceu. O surgimento de números (aparentemente aleatórios) a partir da 18^a casa é resultado de imprecisões derivadas do uso do tipo flutuante. Perceba-se que não deve haver maiores problemas: um erro na 18^a casa não é para preocupar mas, ele está lá.

6.5.1 Nomes

A regra para dar nomes em Python (regras essas que se aplicam à nomeação de variáveis) são simples:

- uma única palavra, ou seja não se admitem espaços em branco dentro dos nomes.
- pode-se usar o caracter sublinha (_) para separar palavras no nome
- o primeiro caractere deve ser uma letra
- depois do primeiro pode-se usar letras ou números
- Letras maiúsculas são diferentes das letras minúsculas (ou seja `OBA` é diferente de `Oba` que não é a mesma coisa que `oba`)
- pode-se mas NÃO se deve usar caracteres acentuados nos nomes.

O leitor não deve se preocupar muito com estas regras, substituindo-as por algum bom senso: os nomes devem ser autodeclarativos (sobre para que serve a variável), com nomes nem muito grandes (provocam erros de digitação) nem muito pequenos (para que serve mesmo a variável B ?).

6.6 Expressões aritméticas

As variáveis e constantes numéricas podem ser operadas e interligadas usando-se as operações aritméticas usuais (e algumas não tão usuais) da matemática convencional. Para isso são usados os símbolos a seguir descritos, que buscam seguir – tanto quanto possível – as convenções usuais da escrita matemática. Por exemplo

```
>>> a = 5
>>> b = 3.6
>>> a + 4
9
>>> b * 3
10.8
>>> (a + b) / 2
4.3
>>> a - 6
-1
>>> a - -8
13
```

Note que a adição é representada pelo sinal de "mais" (+), a subtração por "menos" (−), a multiplicação pelo asterísco (∗) e a divisão real pela barra (/).

Note também que a prioridade é determinada pelo uso de parênteses, que as vezes é redundante, mas nem sempre, pois se o comando fosse escrito como $2 + 8/2$ sem parênteses, o Python primeiro faria a divisão de $8/2$ dando 4 e depois a soma de 2 com 4 resultando 6. Ao se escrever $(2 + 8)/2$ primeiro a soma dando 10 e depois a divisão por 2, que dará 5 de resposta final. A seguir, uma tabela de algumas operações aritméticas:

Símb	Nome	Como funciona
$+$	adição	em $a + b$ o resultado é a adição entre a e b.
$-$	subtração	em $a - b$ o resultado é $a - b$.
$*$	multiplicação	Em $a * b$ o resultado é a multiplicação de a e b.
$/$	divisão real	Em a/b o resultado é um `float` contendo a divisão de a por b.
$//$	divisão inteira	Em $a//b$ o resultado é um inteiro contendo a divisão inteira de a por b.
%	Resto	Em $a\%b$ o resultado é o resto da divisão inteira de a por b, sendo que é um valor $0 \leq resto \leq b - 1$.
$**$	Potência	Em $a * *b$ o resultado de a^b.
abs	Módulo	Em $abs(a)$ o resultado é $-a$ se a é negativo e a se a é positivo.

Note a diferença entre as duas divisões

```
>>> 10/3
3.3333333333333335
>>> 10//3
3
>>> 10%3
1
```

A propósito do sinal +, veja-se o trecho Python

```
>>> a=5
>>> b=7
>>> a+b
7
>>> a='a'
>>> b='b'
>>> a+b
'ab'
>>> a=5
>>> a+b
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    a+b
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>>
```

6.7 Saída de Dados

Todo programa durante sua execução deve em princípio divulgar alguma coisa. Afinal para algo ele está sendo executado. Ainda que haja quase infinitas

possíveis aplicações de programas, grande parte deles vai precisar dizer algo. Uma diferença importante entre Python 2 e Python 3 é que no 2, os parâmetros de print seguem o nome da função sem parênteses enquanto que os parênteses são obrigatórios no Python 3. Aliás este é um dos mecanismos para identificar se um programa originalmente foi escrito em 2 ou 3.

A maneira mais simples de obter o conteúdo de uma variável é apenas digitando seu nome seguido de ENTER. Veja

```
>>> a = 5
>>> b = a+2
>>> b
7
```

Uma maneira mais elaborada de saída é com o comando `print()`. É muito rico, com múltiplas opções, mas vamos começar lentamente. Na sua versão mais simples, o print admite uma lista de variáveis e eventualmente constantes a serem escritas na saída padrão quando o programa for executado. A saída padrão é um conceito importante em computação pessoal: trata-se do dispositivo que o computador usará para todas as saídas, quando nada for afirmado sobre ONDE fazer a saída. Em computação usa-se o conceito de *default* nestes casos. Então a saída *default* é conhecida como saída padrão e quase sempre é o monitor de vídeo. Vejam-se alguns exemplos

`print(a)`	Imprime o valor da variável a.
`print("o valor de a: ",a)`	Agora antes do valor da variável a, imprime-se a mensagem entre aspas. Ajuda a identificar as saídas
`print("a: ",a,"b: ",b)`	Imprime a e b identificando cada uma delas
`print("Desligue agora")`	Imprime apenas a mensagem
`print("Resultado ",x+22)`	Imprime a mensagem e depois o resultado de x+22. Note que o valor de x permanece inalterado

6.8 Entrada de dados

A grande maioria dos algoritmos (programas) precisam obter algum dado do operador antes de realizar o processamento. O comando básico aqui é `input`. Seu formato sempre será

```
variavel = input("mensagem")
```

O que acontece aqui:

- A mensagem citada no comando é impressa na saída padrão
- O programa é interrompido neste ponto
- depois que o usuário digitar alguma coisa e teclar `ENTER`, o que foi digitado é colocado na variável citada no comando
- O processamento segue sem mais interrupções.

Um detalhe importante é que os conteúdos SEMPRE serão carregados no formato de string. Sem transformações, estas variáveis não podem ser usadas, por exemplo, em computações numéricas. Para tanto elas deverão ser transformadas usando (entre outras) as funções: `int` converte algo para inteiro, `float` converte algo para flutuante, e já que estamos estudando conversões, tem-se também `str` que converte algo para string. Veja alguns exemplos

```
>>> idade=input("entre com a idade ")
entre com a idade 33
>>> idade
'33'
>>> idade=int(input("entre com a idade "))
entre com a idade 33
>>> idade
33
```

Perceba que no primeiro caso (sem a transformação `int`) a variável idade é do tipo string, o que se pode perceber nas aspas que são mostradas ('33'). No segundo caso, em que há a transformação de string para inteiro, representada pela chamada à função `int`, o conteúdo da variável é um número e portanto pode ser operada aritmeticamente.

6.8.1 Um programa completo usando input e print

Seja um programa para pedir o raio de um círculo e imprimir a área do mesmo usando a fórmula da geometria

$$S = \pi \times R^2$$

```
def area():
  a=float(input("Informe raio"))
  s=3.1415 * a
  print("a área é: ",s)
area()
```

6.9 Variáveis Lógicas

Uma outra família de variáveis são as variáveis lógicas, geralmente obtidas como resultado de uma comparação ou pergunta. Essas variáveis só podem ter 2 valores que são Verdadeiro (em Python `True`) e Falso (`False`).

Usualmente, estas variáveis são obtidas ao operar condições. As principais condições em Python são

Símb	Nome	Como funciona
==	Igual	Em $a == b$, a resposta será True se a for igual a b e False em caso contrário.
>	Maior	Em $a > b$ a resposta será True se a for maior do que b. Será False em caso contrário.
<	Menor	Em $a < b$ a resposta será True se a for menor do que b. Será False em caso contrário.
>=	Maior ou igual	Em $a >= b$ a resposta será True se a for maior do que ou for igual a b. Será False em caso contrário.
<=	Menor ou igual	Em $a <= b$ a resposta será True se a for menor do que ou igual a b. Será False em caso contrário.
!=	Diferente	Também conhecida como não igual, ela responde o contrário da operação igual ==.
in	Pertence	Em a *in* b retorna True se a pertence a b. Exemplo: 1 *in* $[3, 6, 7, 8, 9]$ é False

6.9.1 Expressões Lógicas

Quando há necessidade de expressar condições compostas, as operações acima podem ser conectadas usando-se 3 conectores lógicos que são:

Símb	Nome	Como funciona
and	E	Em $c1$ *and* $c2$, o resultado será verdadeiro se ambas, $c1$ e $c2$ forem verdadeiras e Falso, senão
or	OU	Em $c1$ *or* $c2$, o resultado será verdadeiro se $c1$ for verdadeiro, ou se $c2$ for verdadeiro ou se ambas forem verdadeiras. Será falso em caso contrário.
not	NÃO	Em *not* $c1$ o resultado será verdadeiro se $c1$ for falsa e será falsa se $c1$ for verdadeiro. Note que este operador é unário (só se aplica a uma condição).

Deve-se lembrar que estas operações e conectores estão definidas na matemática, particularmente na lógica de predicados. Lá seus símbolos são: $\wedge$ para o conectivo E, $\vee$ para o conectivo OU e finalmente o símbolo $\sim$ para a negação lógica.

6.10 Comandos condicionais

Uma característica onipresente em linguagens de programação é a capacidade de executar ou não um conjunto de instruções a depender de uma variável lógica. Dizendo de outra maneira, pode-se comandar uma condição qualquer e se ela for verdadeira (que no Python é `True`) então um conjunto de instruções será executado. Se a condição for falsa (em Python `False`), ou nada é executado, ou também pode ocorrer, um segundo conjunto alternativo de instruções é executado.

O comando condicional por excelência em Python é `if` e ele têm o seguinte formato

```
if condição:
   comando_1
   comando_2
   ...
comando_n
```

Para escrever o comando, deve-se substituir a palavra condição por alguma condição relacional ou lógica, que ao final informe um valor `True` ou `False`. A regra é que os comandos 1, 2, ... só serão executados se a condição for `True`. Já ao final do bloco, o comando n será executado sempre independentemente da condição. Quem determina se o comando está ou não vinculado ao if é o espacejamento no início da linha, chamada em programação de indentação.

Em outras linguagens, usam-se caracteres delimitadores para os blocos de comando ({ e } em C, Java e C++; *begin* e *end* em Pascal e Delphi), mas em Python consoante com a proposta de limpeza da linguagem, usam-se as margens para determinar as condições.

Como tudo na vida isto tem vantagens e desvantagens. Começando pelas primeiras, o código fica limpo e legível. Quem sempre exigia dos programadores que respeitassem as margens, agora ganha um aliado importante: se as margens forem bagunçadas o programa não funcionará bem. Além disso economizam-se caracteres dispensáveis.

Já a desvantagem decorre principalmente da inabilidade de muitos ambientes para manusear este tipo de arquivo. Tem também o mantra do *formato livre* repetido à exaustão depois do Fortran...

Muitos editores se arvoram em trocar um certo número de brancos por um ou mais caracteres de tabulação, e fazer isto é receita quase certa para problemas de operação dos programas. Além disso, muitos gerenciadores de download deixam de respeitar essas margens, fazendo com o que o recebedor de um programa fonte Python tenha sempre que revisar com cuidado o resultado de descargas de código. Um exemplo clássico é o código obtido via PDF.

Um segundo formato para o comando condicional, conhecido como condicional composta é aquele que apresenta uma condição e dois caminhos: um para o True e outro para o resultado False. Agora a separação entre um e outro é sinalizado pela palavra `else` que também deve ser seguida por dois pontos. Acompanhe no exemplo

```
a = 7
if a>3:
   print('oba')
else:
   print('ufa')
print('foi')
```

Acompanhe o raciocínio: Primeiro a é comparado com maior para 3. A resposta é `True` já que $7 > 3$ e com isso a impressão de 'oba' é feita. Como o resultado foi True, salta-se o bloco referente ao else:. Encerra o comando a impressão de 'foi' que independe do resultado da comparação. Veja agora um trecho parecido

```
a = 2
if a>3:
   print('oba')
else:
   print('ufa')
print('foi')
```

Agora a comparação resulta False (já que $2 > 3$ é falso) e com isso a impressão de 'oba' é saltada e depois a impressão de 'ufa' acontece. Novamente 'foi' é

impresso independentemente da comparação inicial.

6.11 Repetições

As repetições são estruturas da linguagem que asseguram a execução repetida de trechos de comandos, até que uma determinada condição for satisfeita. Note que este conceito faz parte do nosso dia a dia.

Por exemplo, uma caixa dágua posta a encher o fará até que atinja a capacidade total. Um forno assando um bolo deverá operar até que o bolo esteja assado, uma esteira carregadeira de um navio opera até que o depósito do navio esteja cheio ou enquanto o silo abastecedor tiver conteúdo, parando na condição que primeiro for atingida, um pneu de carro receberá ar comprimido até que uma certa pressão seja atingida e a água do café vai esquentar até ferver ou um pouco antes.

O que não falta na nossa vida são operações repetitivas que ocorrem até que uma condição seja atendida, ou o que praticamente é a mesma coisa, enquanto certa condição perdurar.

6.11.1 while

Os comandos básicos em Python são `while` que pode ser traduzido por enquanto. Seu formato é

```
while <condição>:
   comando_1
   comando_2
   ...
   comando_i
comando_n
```

Quando ele for encontrado será assim interpretado: primeiro a condição associada ao while será avaliada. Se ela for falsa, ocorre um desvio desprezando todo o bloco indentado abaixo. No exemplo acima, se a condição for falsa, o próximo comando a ser executado é o $comando_n$. Agora, se a condição é verdadeira, o bloco indentado é executado. Quando ele acabar, há um retorno (daí a repetição) ao início do bloco e a reavaliação da condição. Enquanto ela for verdadeira, o bloco fica em *loop* (laço).

Logo, aqui vale uma advertência: quando usar um *while* garanta que em algum momento do bloco, a condição seja tornada falsa, sob pena do seu programa nunca mais sair daí.

6.11.2 break

Um comando que permite modificar o funcionamento do while é o break. Normalmente ele é colocado dentro de uma condição, embora isto não seja obrigatório. Quando break é interpretado ele faz com que o ciclo do while, na verdade qualquer ciclo, seja interrompido, forçando uma saída do bloco indentado.

Exemplo, seja um programa para calcular o triplo de um número digitado. O programa deve operar até ser digitado um número negativo, que significará fim de processamento.

```
a=1
```

```
while 1==1:
  if a<0:
    break
  a=int(input("informe o valor "))
  print(3*a)
print('adeus')
```

Note que como vai-se usar o *break* para a saída do laço, escreve-se uma condição que sempre é verdadeira (conhecida na lógica como uma tautologia), que neste caso é 1 == 1, que, por óbvio, é sempre verdadeira.

6.11.3 Repetição aninhada

Quando se cria um bloco de repetição, dentro do bloco, o Python admite qualquer comando, inclusive um outro comando de repetição. Então surgem os blocos aninhados que são muito importantes em programação. Acompanhe o exemplo

```
i=1
while i<=3:
   j=10
   while j<=30:
      j=j+10
      print (i,j)
   i=i+1
print('acabou')
```

que vai gerar a seguinte saída

```
(1, 20)
(1, 30)
(1, 40)
(2, 20)
(2, 30)
(2, 40)
(3, 20)
(3, 30)
(3, 40)
acabou
```

6.11.4 Continue

Este comando também serve para interromper um ciclo de processamento, mas diferentemente de `break` que encerra inapelavelmente o laço, este comando `continue` encerra o processamento de uma interação e avança para a próxima interação. Acompanhe a diferença entre os dois comandos

```
i=3
while i<10:
   print(i,' ')
   if i%7==0:
      break
   i=i+1
print('acabou')
```

Esta execução teve o seguinte resultado:

```
3
4
5
6
7
acabou
```

Agora acompanhe o seguinte exemplo usando continue

```
i=3
while i<10:
   print(i,' ')
   if i%7==0:
      continue
   i=i+1
print('acabou')
```

Neste caso, não ponha para rodar, pois o programa vai entrar em laço infinito. Veja porque: O processamento começa com i=3, 4, 5 e 6. Quando o valor 7 for atingido, o continue vai comandar um retorno à próxima interação, mas sem alterar o valor de i (já que esta alteração está depois do `continue`. Aí ela vai imprimir 7,7,7,7,.... e não vai parar nunca mais.

6.11.5 For

O For é o grande interador do Python. Ele é uma expansão do comando `for` do C++. Lá no C++ o for funciona como se fosse uma simplificação do while, mas aqui ele é mais do que isso. A mudança se deve ao conceito de interador. Numa rápida explicação um interador é uma generalização do conceito de índice. Ele é uma generalização já que o índexador só se aplica a conjuntos homogêneos indexados (conhecidos como arrays) e o interador se aplica a qualquer coleção de objetos Python. Pode ser um array, mas pode ser também registros em um arquivo, itens em um conjunto, etc etc.

Veja um exemplo radical disso

```
s="Curitiba - Paraná - Brasil"
qtd=0
for c in s:
  if c=="a":
    qtd=qtd+1
print("achei ",qtd," a")
```

Ao executar o código acima, ele vai responder `achei 4 a"`. Note que são 4 letras 'a', sendo que o 'a' acentuado de Paraná, não conta, já que é outro caracter.

6.12 Listas

Uma lista em Python é uma coleção de coisas. Em outras linguagens, essas coisas precisam ser de mesmo tipo, mas em Python, não. Podem ser coisas bem heterogêneas.

Cria-se uma lista em python usando-se os delimitadores colchetes. Uma lista é criada assim:

```
>>> a=[]
```

Criou-se aqui uma lista de nome `a`, inicialmente vazia. Para incluir coisas na lista, usa-se o método `append`, veja como

```
>>> a=[]
>>> a.append(1)
>>> a
[1]
>>> a.append('oi')
>>> a
[1, 'oi']
```

Outro jeito de criar uma lista é usando a função `list` como em

```
>>> b=list(range(6))
>>> b
[0, 1, 2, 3, 4, 5]
```

Note que o primeiro elemento da lista pode ser acessado para leitura fazendo-se

```
>>> a[0]
1
```

E podem ser alterados, como em

```
>>> a[0]='bacana'
>>> a
['bacana', 'oi']
```

Veja mais alguns exemplos, e estude-os...

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
```

Os índices para as listas começam em 0 e crescem para a direita quando de usam números inteiros positivos. A função `len(lista)` devolve o tamanho da lista. Índices negativos vem da direita para a esquerda. `lista[-1]` devolve o último elemento da lista, `lista[-2]` o penúltimo e assim por diante. Veja isso tudo:

```
>>> x=[10,20,30,'oi','viva',4.5,99]
>>> x
[10, 20, 30, 'oi', 'viva', 4.5, 99]
>>> x[0]
10
>>> x[1]
20
```

```
>>> len(x)
7
>>> x[-1]
99
>>> x[-2]
4.5
```

6.12.1 Fatiamento

É uma operação fundamental em Python, pode ser usada em listas, tuplas, strings... É muito usada em MatLab, mas sinceramente eu não consigo dizer quem é o ovo e quem é a galinha, isto quem veio antes. Tanto faz. Uma fatia é uma especificação envolvendo um ou dois "dois pontos" e ela extrai um pedaço de uma lista, tupla ou string. A especificação é

`[começo da fatia:final da fatia:incremento]`

começo o começo da fatia é o índice do seu início, sempre relativo a zero. Se nada for escrito aqui o valor *default* é 0.

final o final da fatia é o próximo número ALÉM do final. Ou seja este número é compatível com o início da numeração em zero. O valor *default* é o tamanho da lista ou `len(lista)`.

incremento É um valor muitas vezes omitido (e neste caso o valor *default* é 1), mas quando presente ele indica quanto somar ao valor atual para obter o próximo valor.

A função embutida len() também se aplica a listas:

```
>>>
>>> a = ['a', 'b', 'c', 'd']
>>> len(a)
4
```

6.12.2 Mais sobre condições

As condições de controle usadas em `while` e `if` podem conter quaisquer operadores, não apenas comparações.

Os operadores de comparação `in` e `not in` verificam se um valor ocorre (ou não ocorre) em uma dada sequência. Os operadores `is` e `is not` comparam se dois objetos são na verdade o mesmo objeto; isto só é relevante no contexto de objetos mutáveis, como listas. Todos os operadores de comparação possuem a mesma precedência, que é menor do que a prioridade de todos os operadores numéricos.

Comparações podem ser encadeadas: Por exemplo a < b == c testa se a é menor que b e também se b é igual a c.

Comparações podem ser combinadas através de operadores booleanos *and* e *or*, e o resultado de uma comparação (ou de qualquer outra expressão), pode ter seu valor booleano negado através de *not*. Estes possuem menor prioridade que os demais operadores de comparação. Entre eles, *not* é o de maior prioridade e *or* o de menor. Dessa forma, a condição A and not B or C é equivalente a (A and (not B)) or C. Naturalmente, parênteses podem e devem ser usados para expressar o agrupamento desejado.

Os operadores booleanos *and* e *or* são operadores short-circuit: seus argumentos são avaliados da esquerda para a direita, e a avaliação para quando o resultado é determinado. Por exemplo, se A e C são expressões verdadeiras, mas B é falsa, então A and B and C não chega a avaliar a expressão C. Em geral, quando usado sobre valores genéricos e não como booleanos, o valor do resultado de um operador atalho é o último valor avaliado na expressão.

Observe que em Python, diferente de C, atribuição não pode ocorrer dentro de uma expressão. Programadores C podem resmungar, mas isso evita toda uma classe de problemas frequentemente encontrados em programas C: digitar = numa expressão quando a intenção era ==.

6.13 Numpy

É o pacote de computação numérica em Python. Ele é central na manipulação de matrizes (e mais genericamente de arrays). Para vetores, o uso da primitiva *lista* do Python é mais do que suficiente, mas a implementação de estruturas com mais de 1 eixo, é problemática no Python original, daí a razão da importância do Numpy. Centenas de pacotes usam-no como pré-requisito.

Para usar o numpy:

Importação Se quiser chamar depois os módulos como `np.modulo` importe assim

```
>>> import numpy as np
```

Se quiser chamar direto sem usar a notação de ponto, importe assim

```
>>> from numpy import *
```

Para criar matriz O elemento `array` é central em numpy:

```
a=np.array([[1,2,3],[3,4,5],[6,6,7]])
ou
a=np.array([(1,2,3),(3,4,5),(6,6,7)])
>>> a
array([[1, 2, 3],
       [3, 4, 5],
       [6, 6, 7]])
```

ambos jeitos devolvem. A explicação para isto é que na criação, tanto faz fornecer uma lista ou uma tupla. Deve-se salientar que operações em `arrays` são até 2 ordens de magnitude mais rápidas que idênticas operações em `lists`. (referência DEI20, pág. 240). Até 450 pacotes do mundo Python são dependentes de Numpy. (Isto é: exigem que Numpy esteja instalado antes de funcionarem).

Para indexar

```
a[1][2] é igual a 5 OU a[1,2] que igualmente é 5
```

in É usado para testar se um elemento está num array ou não:

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> 2 in a
True
>>> 0 in a
False
```

Criação de PA A função arange() cria uma PA, cujos termos são a=np.arange(vl_inicial, até_vl_final, incremento), como em

```
a=np.arange(1,5,0.5)
>>> a
array([ 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
```

Atributos de ndarray Um ndarray tem os seguintes atributos:

.ndim = número de dimensões

.shape = uma tupla com as dimensões

```
>>> a
array([1, 2, 3, 4])
>>> a.shape
(4,)
```

Cópia de array Para criar um array como cópia de outro, fazer

```
s3 = s2.copy()
```

Reorganizar um array Para reorganizar um array:

```
>>> a1 = np.arange(0.0, 12.0, 1.0)
>>> a1
[ 0. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11.]
>>> a2 = a1.reshape( (2,6) )
>>> print a2
[[ 0. 1. 2. 3. 4. 5.]
[ 6. 7. 8. 9. 10. 11.]]
```

Reorganizar in place Para reorganizar in-place

```
>>> a1
[ 0. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11.]
>>> a1.resize([2,6])
>>> a1
[[ 0. 1. 2. 3. 4. 5.]
[ 6. 7. 8. 9. 10. 11.]]
```

Média Para retornar a média, variância, soma e produto:

```
A.mean() -> média dos valores em A
A.var() -> variancia dos valores de A
A.sum() -> soma
A.prod() -> produto
A.max() -> maior valor
A.min() -> menor valor
A.argmax() -> índice do maior valor
A.argmin() -> índice do menor valor
unique(a) -> retorna os valores únicos
             (sem repetição) de a
```

Funções Universais Para funções universais (ufuncs)

```
>>> a=[1,2,3,4]
>>> b=[0.1, 0.7, 8, 9]
>>> a
[1, 2, 3, 4]
>>> b
[0.1, 0.7, 8, 9]
>>> a+b
[1, 2, 3, 4, 0.1, 0.7, 8, 9]
>>> a=np.array([1,2,3,4])
>>> b=np.array([0.1, 0.7, 8, 9])
>>> a+b
array([  1.1,   2.7,  11. ,  13. ])
```

Broadcasting Para broadcasting (aplicar um escalar a um array)

```
>>> a
array([1, 2, 3, 4])
>>>
>>> a+3
array([4, 5, 6, 7])
>>> b*5
array([  0.5,   3.5,  40. ,  45. ])
```

multiplicação matricial Multiplicação matricial:

```
>>> a1 = np.array([[1, 2, -4], [3, -1, 5]])
>>> a2 = np.array([[6, -3], [1, -2], [2, 4]])
>>> np.dot(a1,a2)
array([[  0, -23],
       [ 27,  13]])
```

Produto vetorial O produto vetorial é

```
>>> x = (1, 2, 0)
>>> y = (4, 5, 6)
```

```
>>> print np.cross(x, y)
[12 -6 -3]   para resolver isto ache o det da matriz
i j k
1 2 0
4 5 6, com resultados em i, j e k
```

determinante Para obter o determinante de uma matriz

```
>>> m = np.array(((2,3), (-1, -2)))
>>> m
[[ 2 3]
[-1 -2]]
>>> np.linalg.det(m)
-1.0
```

Matriz inversa E a matriz inversa

```
>>> good = np.array(((2.1, 3.2), (4.3, 5.4)))
>>> np.linalg.inv(good)
[[-2.23140496 1.32231405]
[ 1.7768595 -0.8677686 ]]
```

Sistema de equações lineares A solução de um sistema de equações lineares por exemplo

```
2x + y = 19
x - 2y = 2
cuja solução é (x=8 and y=3):
>>> coeffs = np.array([[2,1], [1,-2]])
>>> coeffs
[[ 2 1]
[ 1 -2]]
>>> consts = np.array((19, 2))
>>> consts
[19 2]
>>> np.linalg.solve(coeffs, consts)
[ 8. 3.]
```

Transposta

```
>>> a = np.array(range(6), float).reshape((2, 3))
>>> a
array([[ 0., 1., 2.],
[ 3., 4., 5.]])
>>> a.transpose()
array([[ 0., 3.],
[ 1., 4.],
[ 2., 5.]])
```

Concatenação

```
>>> a = np.array([1,2], float)
>>> b = np.array([3,4,5,6], float)
>>> c = np.array([7,8,9], float)
>>> np.concatenate((a, b, c))
array([1., 2., 3., 4., 5., 6., 7., 8., 9.])

>>> a = np.array([[1, 2], [3, 4]], float)
>>> b = np.array([[5, 6], [7,8]], float)
>>> np.concatenate((a,b))
array([[ 1., 2.],
[ 3., 4.],
[ 5., 6.],
[ 7., 8.]])
>>> np.concatenate((a,b), axis=0)
array([[ 1., 2.],
[ 3., 4.],
[ 5., 6.],
[ 7., 8.]])
>>> np.concatenate((a,b), axis=1)
array([[ 1., 2., 5., 6.],
[ 3., 4., 7., 8.]])
```

Funções matemáticas abs, sign, sqrt, log, log10, exp, sin, cos, tan, arcsin, arccos, arctan, sinh, cosh, tanh, arcsinh, arccosh, and arctanh, floor, ceil, and rint fornecem o *lower*, *upper*, or *nearest* (rounded) inteiro.

Lista de nomes definidos Para retornar uma lista de nomes definidos

```
>>> dir()
```

subpacotes Lista de subpacotes que ajudam na programação:

subpacote	Objetiv
core	basic objects
lib	additional utilities
linalg	basic linear algebra
fft	discrete Fourier transforms
random	random number generators
distutils	enhanced build, distrib improvements
testing	unit-testing utility functions
f2py	automatic wrapping of Fortran code

Geração de aleatórios Compreendidos entre 0 e 1.

```
>>> np.random.rand(3,2)
array([[ 0.14022471, 0.96360618], #random
       [ 0.37601032, 0.25528411], #random
       [ 0.49313049, 0.94909878]]) #random
```

Inteiros aleatórios Veja o exemplo

```
>>> randint(low[, high, size]) # Return random integers
        from low (inclusive) to high (exclusive)
```

Para mostrar

```
>>> import matplotlib.pyplot as plt
>>> d1 = np.random.random_integers(1, 6, 1000)
>>> d2 = np.random.random_integers(1, 6, 1000)
>>> dsums=d2+d2
>>> count,bins,ignored = plt.hist(dsums,11,normed=True)
>>> plt.show()
```

Distribuição normal Esta função retorna simulações de uma distribuição normal. Para obter $N(\mu, \sigma^2)$, basta fazer $sigma * np.random.randn(...) + mu$. Onde $\mu =$ média da distribuição e σ^2 é a variância. Veja-se o exemplo

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.4940, 4.0095, -1.8181, 7.2971], #random
       [ 0.3992, 4.6845,  4.9939, 4.8405]]) #random
```

Cria um array $2, 4$ contendo simulações de $3, 6.25$. Lembre que $2.5^2 = 6.25$.

Distribuição binomial Cria simulações desta distribuição. Usa 2 parâmetros: `n`, que significa o número de tentativas e `p` que vem a ser a probabilidade de sucesso da distribuição ($0 \leq p \leq 1$). A função retorna para cada simulação o número de sucessos. Lembrando que a probabilidade é

$$P(N) = \left(\begin{array}{c} n \\ N \end{array} \right) p^N (1-p)^{n-N}$$

) Veja-se o exemplo:

```
>>> n=10
>>> p=0.5
>>> s=np.random.binomial(n,p,1000)
```

O exemplo acima faz 1000 simulações, cada uma jogando 10 moedas e informando a quantidade de caras em cada simulação.

```
>>> a = np.array([[1,4],[3,1]])
>>> np.sort(a) # sort along the last axis
array([[1, 4],
       [1, 3]])
>>> np.sort(a,axis=None) #sort flattened array
array([1, 1, 3, 4])
>>> np.sort(a,axis=0) # sort along first axis
array([[1, 1],
       [3, 4]])
```

Desvio padrão em arrays

```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.std(a)
1.1180339887498949
>>> np.std(a, axis=0)
array([ 1., 1.])
>>> np.std(a, axis=1)
array([ 0.5, 0.5])
```

6.14 Sympy

O Symbolic Python é um pacote Python que implementa um sistema de computação simbólica (CAS=Computer Algebra System) no ambiente Python. Para entender o que é um sistema deste tipo, acompanhe:

computação numérica sqrt(8) é igual a 2.8284271247461903

computação simbólica sqrt(8) é igual a 2*sqrt(2)

Sendo que aqui tem-se uma constatação: ir do numérico para o simbólico não é possível, mas ir do simbólico para o numérico é fácil (neste caso basta fazer `sqrt(8).evalf()` cuja resposta será a mesma de cima – até porque é calculada do mesmo jeito: 2.8284271247461903).

Fica evidente que a computação simbólica abrange a computação numérica mas é muito (muito !) mais do que ela.

As vantagens de usar o sympy versus o Maple (por exemplo) são pelo menos duas: sympy é de graça (Maple custa 1.000 dólares) e por ser em python admite que você use tudo o que já sabe de python e também permite usar os mais de 135.000 pacotes também de graça.

Instalação e uso O sympy precisa ser instalado. Na versão para windows, você deve localizar o diretório scripts e nele fazer `pip install sympy` em um computador que tenha acesso à Internet. Se quiser usar winpython, nele o sympy já está pré-instalado. Num ou noutro, precisa importar o pacote. Faça `from sympy import *` ou algum comando equivalente.

symbols Como você está num ambiente 100% python, precisa dizer que algumas variáveis são simbólicas. Isto é feito pelo método `symbols` como em

```
from sympy import symbols
>>> x,y = symbols('x y')
>>> expr = x+2*y
>>> expr
x + 2*y
>>> expr + 1
x + 2*y + 1
>>> expr - x
2*y
>>> x*expr
x*(x + 2*y)
>>> expand(x*expr)
```

```
x**2 + 2*x*y
>>> factor(x**2 + 2*x*y)
x*(x + 2*y)
```

O sympy pode fazer todo o tipo de computação simbólica, como simplificar, calcular derivadas, integrais e limites, resolver equações, trabalhar com matrizes, e muito mais, tudo de maneira simbólica. Ele inclui módulos para plotagem, impressão, geração de código, física, estatística, combinatória, teoria dos números, geometria, lógica, etc. Vejamos um exemplo para calcular: a derivada de

$$sin(x) \times e^x$$

,

$$\int ((e^x \times sin(x) + e^x \times cos(x))dx$$

e também

$$\int_{-\infty}^{\infty} sin(x^2)dx$$

```
>>> from sympy import *
>>> x, t, z, nu = symbols('x t z nu')
>>> init_printing(use_unicode=True)
>>> diff(sin(x)*exp(x), x)
 x           x
e .sin(x) + e .cos(x)
>>> integrate(exp(x)*sin(x) + exp(x)*cos(x), x)
 x
e .sin(x)
>>> integrate(sin(x**2), (x,-oo, oo))
```

Substituição

```
>>> expr = cos(x)+1
>>> expr.subs(x,y)
cos(y) + 1
>>> expr.subs(x,0)
2
>>> expr = x**y
expr = expr.subs(y, x**y)
>>> expr
x**(x**y)
>>> expr = x**3 + 4*x*y - z
>>> expr.subs([(x,2),(y,4),(z,0)])
40
```

Sympify Não confundir com `simplify`. Transforma um string em uma expressão SymPy.

```
a = "x**2 + 3*x - 1/2"
ex = sympify(a)
ex.subs(x,2)
19/2
```

evalf Permite achar o valor em ponto flutuante de uma expressão

```
>>> ex = sqrt(8)
>>> ex.evalf(20)  # 20=tamanho
2.8284271247461900976
>>> ex2 = cos(2*x)
>>> ex2.evalf(subs=x:2.4)
0.0874989834394464
```

Simplify simplificação de expressões.

```
>>> simplify(sin(x)**2 + cos(x)**2)
1
>>> simplify((x**3+x**2-x-1)/(x**2+2*x+1))
x - 1
```

Expand Expande expressões.

```
>>> expand((x+1)**2)
 2
x + 2.x + 1
>>> expand((x+2)*(x-3))
 2
x . x - 6
```

Factor Fatora expressões.

```
>>> factor(x**2*z + 4*x*y*z + 4*y**2*z)
           2
z.(x + 2.y)
```

Outros comandos collect = coleciona potências comuns
cancel = transforma polinômio na forma p/q
apart = executa uma decomposição em frações parciais
trigsimp = simplificação trigonométrica
expand_trig = expansão trigonométrica
factorial, binomial, gamma, power, powers, ...

Derivadas podem ser chamadas com `diff(expressao)` e com `Derivative(expressao)`.

```
>>> diff(cos(x),x)
-sin(x)
```

Integrais o comando que integra é `integrate`. O comando `Integral` serve para representar a integral e pode ser calculado depois com `x.doit()`. Atente que integrate é com minúscula e Integral é com maiúscula. Acompanhe

```
>>> x = Integral(log(x)**2,x)
>>> x
```

```
  /
 |     2
 | log (x) dx
/
>>> x.doit()
     2
x.log (x) - 2.x.log(x) + 2.x
>>>
sin(x)
```

Limites

```
>>> limit(sin(x)/x, x, 0)
1
```

Solve Em Sympy a igualdade em uma equação é dada por Eq como em $x^2 = 1 \Rightarrow$ `Eq(x**2,1)` e para resolver esta equação escreve-se

```
>>> solveset(Eq(x**2,1), x)
-1, 1
```

Note que isto pode ser escrito como `solveset(Eq(x**2-1,0), x)` e neste caso pode ser também `solveset(x**2-1, x)` (já que é igual a zero). Para resolver um sistema de equações lineares o comando é `linsolve`, veja

```
>>>linsolve([x+y+z-1,x+y+2*z-3],(x,y,z))
(-y-1, y, 2)
>>>linsolve(([1,1,1,1],[1,1,2,3])),(x,y,z))
(-y-1, y, 2)
```

Para sistemas não lineares, o comando é `nonlinsolve`.

```
>>>nonlinsolve([x*y-1,x-2],x,y)
(2, 1/2)
```

Para equaçoes diferenciais, use `dsolve`.

Matrizes Para fazer matrizes em Sympy use o objeto `Matrix`, como em `Matrix([[1, -1], [3, 4], [0, 2]])`. Matrizes não são imutáveis, podendo ser alteradas.

Para a matriz `M`, sua forma é $M.shape$. Para achar a inversa fazer $M**-1$. Para achar a transposta, faça $M.T$. Para o determinante, use $M.det$. Como detalhe perceba como o pacote simbólico mostra valores racionais (na forma de uma fração irredutível).

```
>>> M=Matrix([[1,2,3],[10,20,51],[8,-1,9]])
>>> M
+1   2   3 +
|           |
|10  20  51|
```

```
|           |
+8   -1   9 +
>>> M**-1
+ 11                  +
| --    -1/17   2/17  |
| 17                  |
|                     |
|106                  |
|---    -5/119  -1/17 |
|119                  |
|                     |
|-10                  |
|--    1/21     0     |
+ 21                  +
>>> M.T
+1   10   8 +
|           |
|2   20   -1|
|           |
+3   51   9 +
```

Segue a documentação tratando de teoria dos números (basicamente as tratativas referentes a números primos, depois criptografia, depois matemática concreta, depois geometria, depois integração, lógica, séries, ... a documentação do pacote tem (em maio de 2018) 1.878 páginas.

6.14.1 Exercício

1. Resolva (usando `solveset`) a equação:

$$3x^3 + 4x^2 + 32x + 256 = 0$$

no domínio dos números reais.

2. Resolva a integral definida a seguir,

$$\int_6^8 (3 \times sin(x))$$

de maneira numérica, usando o comando `Integral` de SymPy nos intervalos dados e depois calcule o valor numérico da integral usando o método `as_sum` com 50 intervalos e tomando como parâmetro o ponto médio de cada intervalo: (Forneça a resposta com 3 casas decimais)

```
>>> init_printing(use_unicode=True)
>>> e = Integral(...funçao...,(x,a,b))
>>> e
>>> e.as_sum(50,'midpoint').evalf()
```

1	2

6.15 Um exemplo de plotagem de uma função

A seguir, a apresentação de uma função e suas duas derivadas (a primeira e a segunda) com as funções, calculadas pelo sympy, resolvidas por ele e desenhadas pelo matplotlib. No script a seguir, o usuário tem que alterar 3 coisas:

- a função f lançada na forma de texto
- o valor vi indicando a coordenada x inicial da plotagem
- o valor vf indicando a coordenada x final da plotagem.

O programa emite as 3 funções (f, df/dx e d^2f/dx^2), as raízes das 3 funções e finalmente o gráfico gerado, no qual a cor azul é usada para f, vermelho para df/dx e verde para d^2f/dx^2. Veja-se um exemplo:

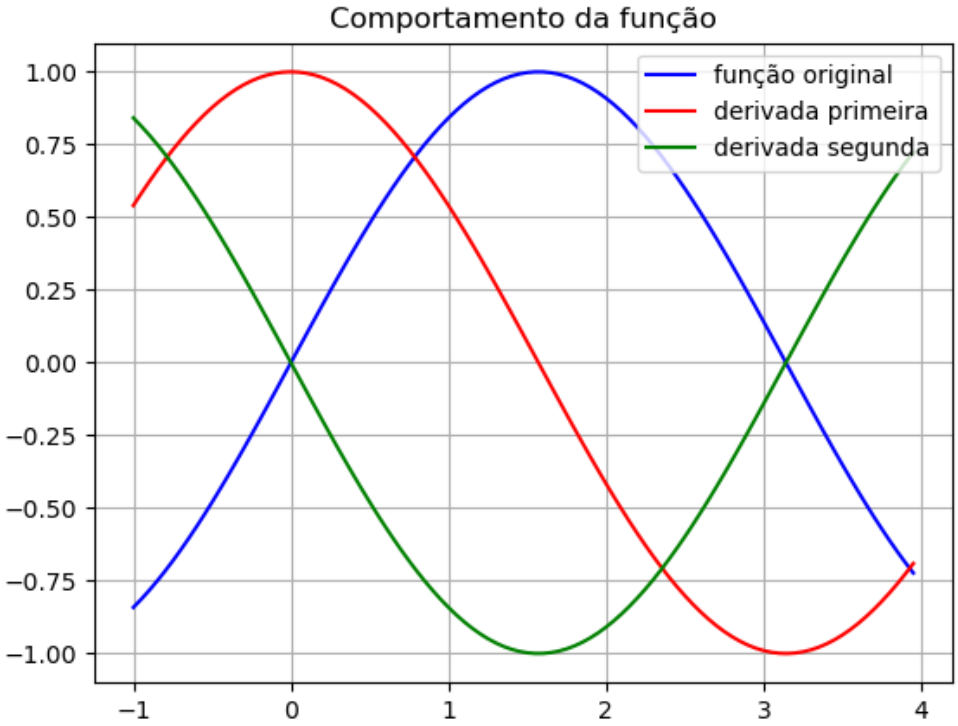
```
# funcao para desenhar uma função genérica e suas 2 derivadas
import matplotlib.pyplot as plt
import numpy as np
from sympy import *
x, y, z, t = symbols('x y z t')
k, m, n = symbols('k m n', integer=True)
f, g, h = symbols('f g h', cls=Function)
init_printing()
#f=x**3-2*x**2+x
f=sin(x)
g=diff(f,x)
h=diff(g,x)
print('f       = ',end='')
pprint(f)
print('df/dx   = ',end='')
pprint(g)
print('d2f/dx2 = ',end='')
pprint(h)
print('raizes f       = ',solve(f,x))
print('raizes df/dx   = ',solve(g,x))
print('raizes d2f/dx2 = ',solve(h,x))
vi=-1
vf=4
x1=np.arange(vi,vf,((vf-vi)/100))
y1=[0.0]*100
y2=[0.0]*100
y3=[0.0]*100
for i in range(100):
  y1[i]=f.subs(x,x1[i])
  y2[i]=g.subs(x,x1[i])
  y3[i]=h.subs(x,x1[i])
plt.plot(x1,y1,'b-',label='função original')
plt.plot(x1,y2,'r-',label='derivada primeira')
plt.plot(x1,y3,'g-',label='derivada segunda')
plt.grid()
plt.legend()
```

```
plt.title("Comportamento da função")
plt.show()
```

A resposta literal foi

```
f       = sin(x)
df/dx   = cos(x)
d2f/dx2 = -sin(x)
raizes f       =  [0, pi]
raizes df/dx   =  [pi/2, 3*pi/2]
raizes d2f/dx2 =  [0, pi]
```

e o desenho gerado foi



6.15.1 Resposta

-4 3.317

Usando sympy para obter as derivadas

Como mandar o python (sympy) calcular as derivadas de uma determinada função

```
import matplotlib.pyplot as plt
import numpy as np
from sympy import *
x= symbols('x')
f, g, h = symbols('f g h', cls=Function)
init_printing()
f=2*x**3-27*x**2+84*x+25
g=diff(f,x)
```

```
h=diff(g,x)
print('f       = ',end='')
pprint(f)
print('df/dx   = ',end='')
pprint(g)
print('d2f/dx2 = ',end='')
pprint(h)
print('raizes f       = ',solve(f,x))
print('raizes df/dx   = ',solve(g,x))
print('raizes d2f/dx2 = ',solve(h,x))
vi=0
vf=10
x1=np.arange(vi,vf,((vf-vi)/100))
y1=[0.0]*100
y2=[0.0]*100
y3=[0.0]*100
for i in range(100):
  y1[i]=f.subs(x,x1[i])
  y2[i]=g.subs(x,x1[i])
  y3[i]=h.subs(x,x1[i])
plt.plot(x1,y1,'b-',label='função original')
plt.plot(x1,y2,'r-',label='derivada primeira')
plt.plot(x1,y3,'g-',label='derivada segunda')
plt.grid()
plt.legend()
plt.title("Comportamento da função")
plt.show()
```

Ele imprimiu os seguintes resultados (numéricos)

```
f       =          3       2
               2 x  - 27 x  + 84 x + 25
df/dx   =     2
           6 x  - 54 x + 84
d2f/dx2 = 12 x - 54
raizes f       =  [9/2 - 75/(4*(-1/2 - sqrt(3)*I/2)*(2079/8
+ 27*sqrt(606)*I/2)**(1/3))
 - (-1/2 - sqrt(3)*I/2)*(2079/8
+ 27*sqrt(606)*I/2)**(1/3)/3, 9/2 - (-1/2 +
sqrt(3)*I/2)*(2079/8 + 27*sqrt(606)*I/2)**(1/3)/3
 - 75/(4*(-1/2 + sqrt(3)*I/2)*
(2079/8 + 27*sqrt(606)*I/2)**(1/3)), 9/2 - (2079/8
 + 27*sqrt(606)*I/2)**(1/3)/3
 - 75/(4*(2079/8 + 27*sqrt(606)*I/2)**(1/3))]
raizes df/dx   =  [2, 7]
raizes d2f/dx2 =  [9/2]
```

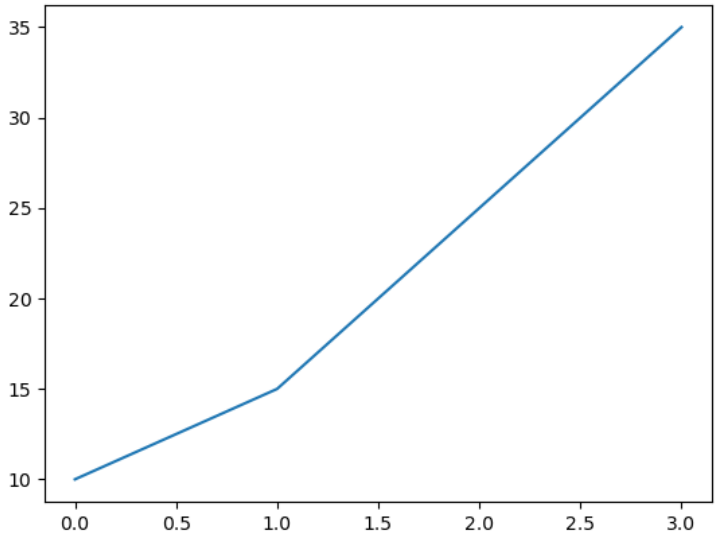
6.16 Matplotlib

Gráfico simples de linhas

Seja plotar os valores 10, 15, 25 e 35 em um gráfico comum de linha:

```
import matplotlib.pyplot as plt
def pl1():
  x=[10, 15, 25, 35]
  plt.plot(x)
  plt.show()
pl1()
```

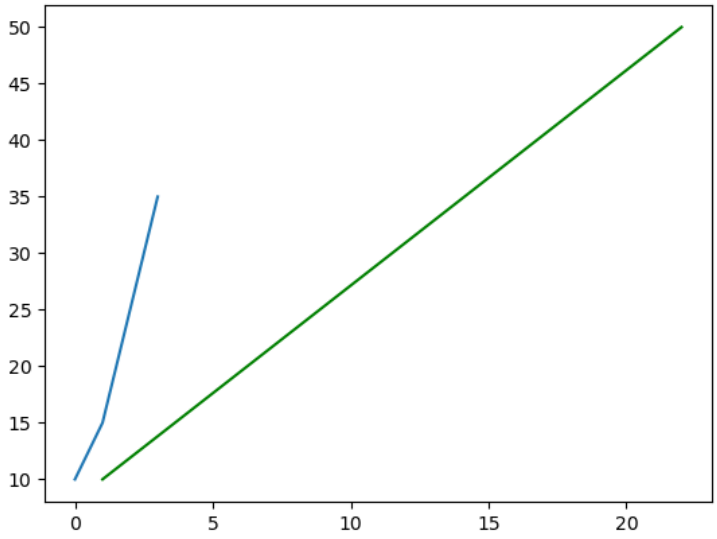
com este resultado



Riscos aleatórios no gráfico Seja escrever riscos aleatórios. Por exemplo, um risco de 1,10 até 22,50

```
xd=[1,22]
yd=[10,50]
plt.plot(xd,yd,'g') #g=green
```

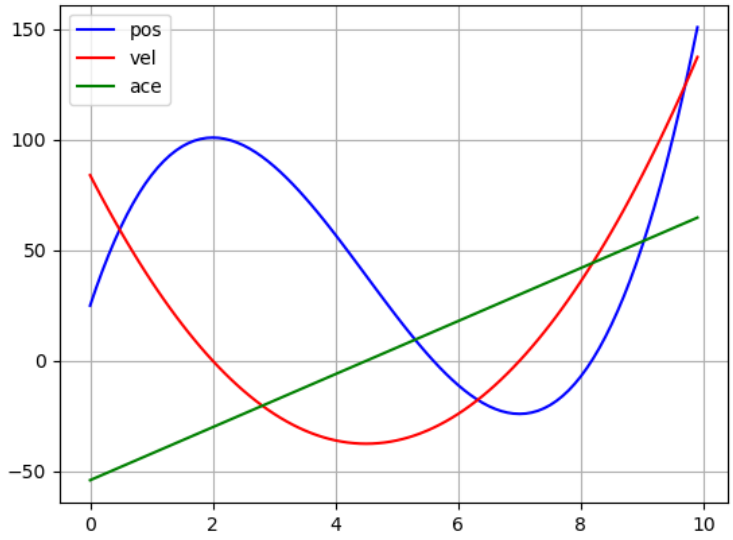
com este resultado (agregado com o anterior...)



Múltiplas curvas de funções Seja estudar um fenômeno através de 3 curvas

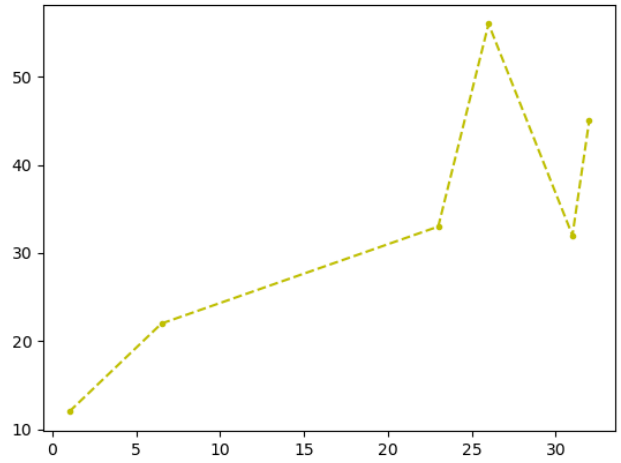
```
import numpy as np
import matplotlib.pyplot as plt
def s(t):
  return 2*t**3-27*t**2+84*t+25
def v(t):
  return 6*t**2-54*t+84
def a(t):
  return 12*t-54
def p40():
  t=np.arange(0,10,0.1)
  ss=s(t)
  vv=v(t)
  aa=a(t)
  plt.plot(t,ss,'b-',label='pos')
  plt.plot(t,vv,'r-',label='vel')
  plt.plot(t,aa,'g-',label='ace')
  plt.grid()
  plt.legend()
  plt.show()
p40()
```

com este resultado



Operando com pares de números

```
import matplotlib.pyplot as plt
def des3():
  x=[1, 6.5, 23, 26, 31, 32]
  y=[12, 22, 33, 56, 32, 45]
  plt.plot(x,y,'y.--')
  plt.show()
des3()
```



Usando várias funções

```
import matplotlib.pyplot as plt
import numpy as np
def des3():
  x=np.arange(0,5,0.1)
  y=np.sin(x)
  z=np.cos(x)
  w=np.tan(x)
  plt.plot(x,y,'r',label='sen(x)')
  plt.plot(x,z,'g',label='cos(x)')
  plt.plot(x,w,'b',label='tan(x)')
  plt.axis([0,5,-15,5])
  plt.grid()
  plt.title('principais f. trigonométricas')
  plt.legend()
  plt.show()
des3()
```

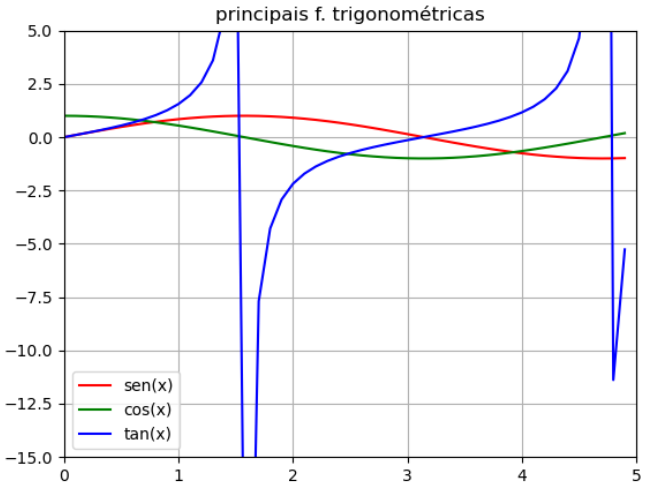
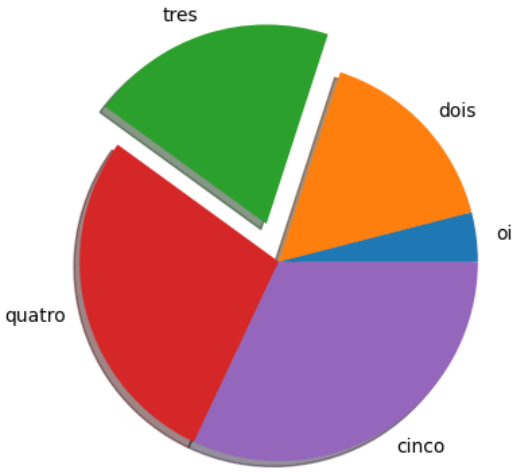


Gráfico de pizza

```
import matplotlib.pyplot as plt
def torta():
  plt.pie([1,4,5,7,8],labels=['oi','dois','tres',
          'quatro','cinco'],
          explode=[0,0,0.2,0,0],shadow=True)
  plt.show()
torta()
```

Com o seguinte resultado



Este é um dos pacotes disponíveis no mundo Python para elaborar melhor as saídas na forma gráfica. Existem outros, vale a pena uma pesquisa no PyPi (Python Package Index) .

O pacote a ser instalado é o matplotlib (pip install matplotlib) e ele exige um monte de dependências (o python, numpy, além de inúmeras outras, que o instalador mesmo verifica).

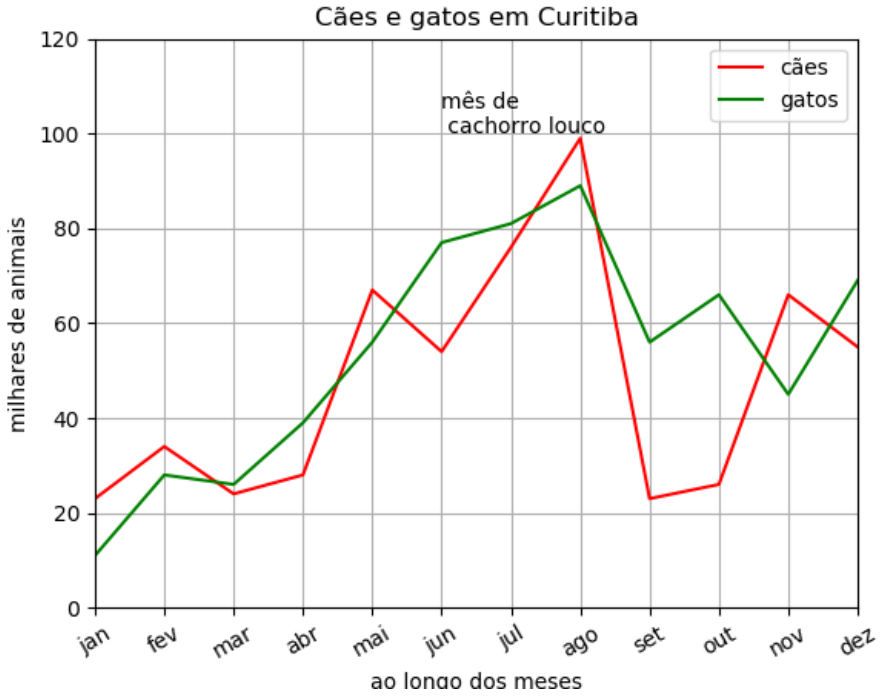
Modos de uso

Há 2 modos de uso do pacote: usar a interface procedural `pyplot` ou usar a API nativa da matplotlib, esta mais orientada a objetos. A primeira é constituída por métodos (como xlabel, legend, etc) e elas podem ser usadas de duas maneiras: se chamadas sem argumentos, devolvem o valor atual do parâmetro e se se chamadas com parâmetros elas atualizam este valor. Veja a seguir uma chamada usando este método:

Para criar um gráfico simples, via métodos

```
# caogato
import matplotlib.pyplot as plt
import numpy as np
def caogato():
 ca=[23,34,24,28,67,54,76,99,23,26,66,55]
 ga=[11,28,26,39,56,77,81,89,56,66,45,69]
 plt.plot(ca,'r-',label='cães')
```

```
 plt.plot(ga,'g-',label='gatos')
 plt.legend()
 plt.title('Cães e gatos em Curitiba')
 plt.xlabel('ao longo dos meses')
 plt.ylabel('milhares de animais')
 plt.axis([0,11,0,120])
 plt.xticks(np.arange(12),('jan','fev','mar',
 'abr','mai','jun','jul','ago','set','out',
 'nov','dez'),rotation=30)
 plt.text(5,100,'mês de /n cachorro louco')
 plt.grid()
 plt.savefig('c:/p/python/caogato.png')
 plt.show()
caogato()
```



No caso de usar a API, cada ação corresponde a dois métodos na API. Por exemplo, no caso de `xlabel`, há o método `axe.set_xlabel` para colocar coisas lá e o método `axe.get_xlabel` para recuperar o que tem lá.

O método plot() Serve para plotar `y` versus `x` com linhas e/ou marcadores. Vejamos alguns exemplos:

```
plot(y)
```

Marca os valores de `y` usando `x` os índices do array `0..n-1`. Veja um exemplo

```
def exeplo1():
  y=[12,3,50,33]
  plt.plot(y)
  plt.show()
```

Agora, x pode ser explicitamente estabelecido, através de pares `x,y`. Veja o exemplo:

```
def exeplo2():
  x=[8,17,33,66]
  y=[12,3,50,33]
  plt.plot(x,y)
  plt.show()
```

Um segundo conjunto de dados (`x2,y2`) pode ser desenhado junto

```
def exeplo3():
  x=[8,17,33,66]
  y=[12,3,50,33]
  x2=[0,3,7,12]
  y2=[33,88,55,77]
  plt.plot(x,y,x2,y2)
  plt.show()
```

Mas isto pode ser feito em duas chamadas a `plot`, com o mesmo resultado, como em

```
def exeplo4():
  x=[8,17,33,66]
  y=[12,3,50,33]
  x2=[0,3,7,12]
  y2=[33,88,55,77]
  plt.plot(x,y)
  plt.plot(x2,y2)
  plt.show()
```

A formatação das coisas pode ser feita com propriedades de `Line2D` ou com palavras chave. Assim, ambos os valores abaixo terão o mesmo resultado

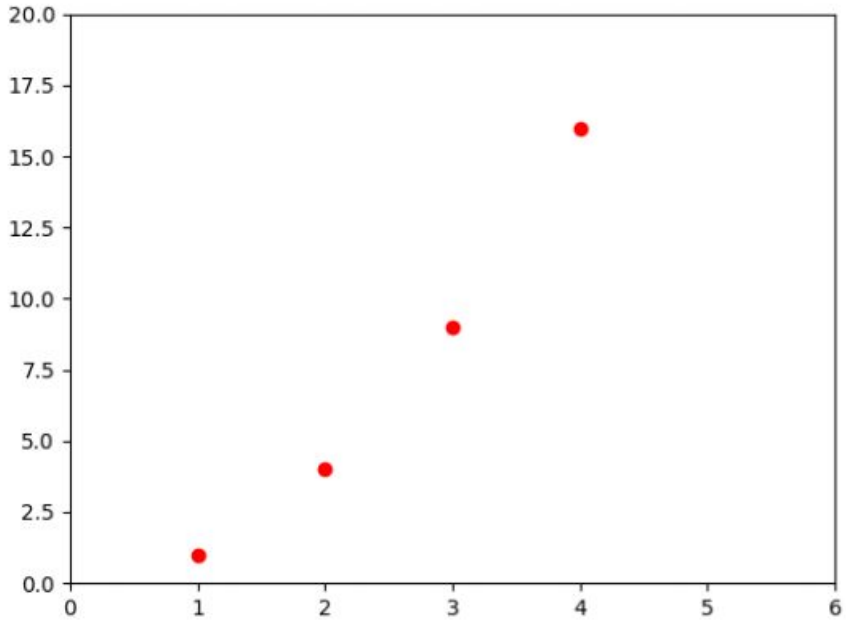
```
>>> plot(x, y, 'go--', linewidth=2, markersize=12)
>>> plot(x, y, color='green', marker='o',
    linestyle='dashed',
    linewidth=2, markersize=12)
```

As cores são: 'b'=blue, 'g'=green, 'r'=red, 'c'=ciano, 'm'=magenta, 'y'=yellow, 'k'=black e 'w'=white. Os marcadores são '.'=ponto, 'o'=bola, 'v'=triângulo, "=triângulo, '<'=triângulo, '>'=triângulo, 's'=quadrado, 'p'=pentágono, '*'=estrela, '+'=mais, 'x'=vezes, 'D'=diamante, 'd'=diamante estreito, '|'=linha vertical, '_'=linha horizontal, '-'=linha sólida, '–'=linha picotada, '-.'=linha ponto, ':'=linha pontilhada (tem mais...)

Um exemplo bobinho

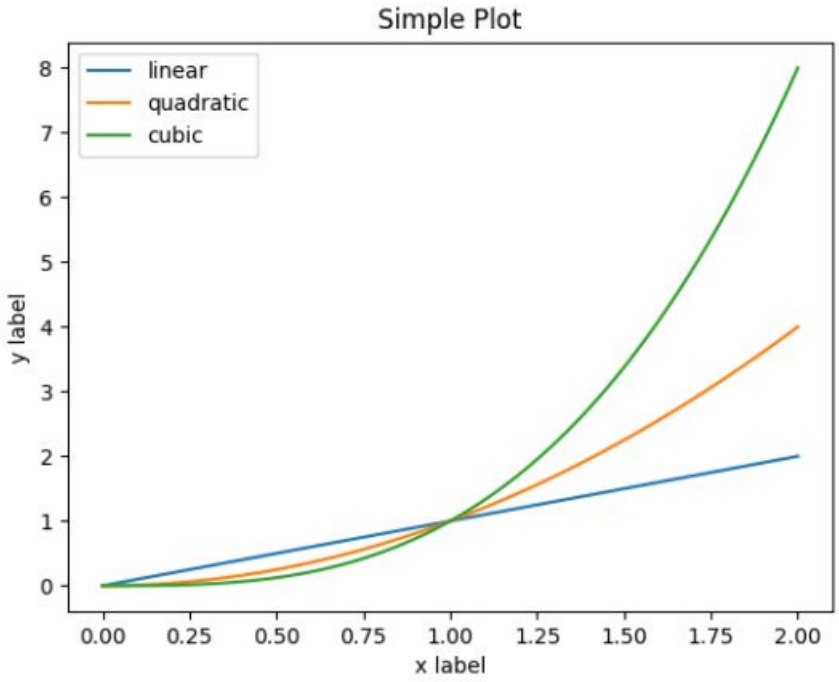
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```

`ro` significa 'red circles'. `b-` seria risco azul. `r--` é red dashes, `bs` é blue squares e `g^` é green triangles.



Um exemplo de plotagem simples

```
x = np.linspace(0, 2, 100) # pontos espaçados
plt.plot(x, x, label='linear')
plt.plot(x, x**2, label='quadratic')
plt.plot(x, x**3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
plt.show()
```



Plotando pontos Para plotar os pontos (1,1), (2,1), (1,-1), (-2,2) e (-2,-2) deve-se fazer

```
x=[1,2,1,-2,-2]
y=[1,1,-1,2,-2]
plt.scatter(x,y)
```

Se quiser identificar os pontos faça

```
plt.text(1,1,"1")
plt.text(2,1,"2")
plt.text(1,-1,"3")
plt.text(-2,2,"4")
plt.text(-2,-2,"5")
```

Para desenhar os eixos

```
plt.vlines(x=0,ymin=-2.5,ymax=2.5)
plt.hlines(y=0,xmin=-2.5,xmax=5.5)
```

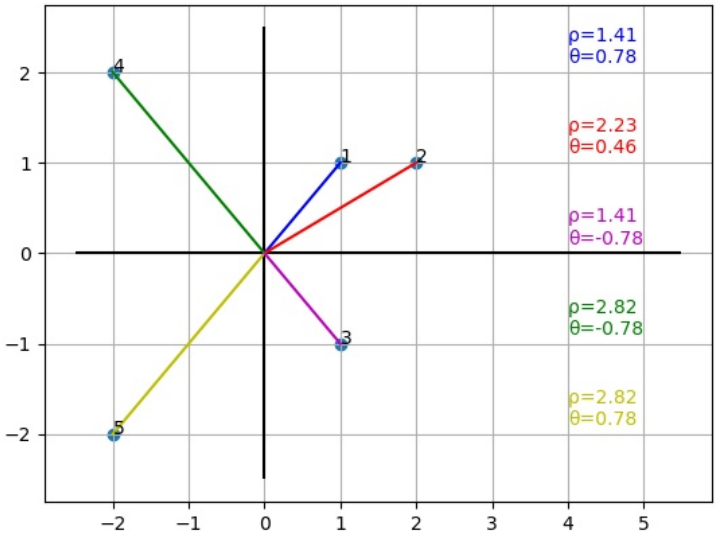
Finalmente, para conectar cada ponto à origem, faça

```
xdata=list([0, 1])
ydata=[_ for _ in xdata]
plt.plot(xdata,ydata,'b')

xdata=list([0, 2])
ydata=[_/2 for _ in xdata]
plt.plot(xdata,ydata,'r')

xdata=list([0, -2])
ydata=[abs(_) for _ in xdata]
plt.plot(xdata,ydata,'g')

xdata=list([0, -2])
ydata=[_ for _ in xdata]
plt.plot(xdata,ydata,'y')

xdata=list([0, 1])
ydata=[-_ for _ in xdata]
plt.plot(xdata,ydata,'m')
```

Para escrever algo no texto:

```
rho=u'\u03C1'
theta=u'\u03B8'
print(rho)
plt.text(4,2.1,rho+"=1.41 \n"+theta+"=0.78",c='b')
plt.text(4,1.1,rho+"=2.23 \n"+theta+"=0.46",c='r')
plt.text(4,0.1,rho+"=1.41 \n"+theta+"=-0.78",c='m')
plt.text(4,-0.9,rho+"=2.82 \n"+theta+"=-0.78",c='g')
plt.text(4,-1.9,rho+"=2.82 \n"+theta+"=0.78",c='y')
```

Eis o resultado deste programa:

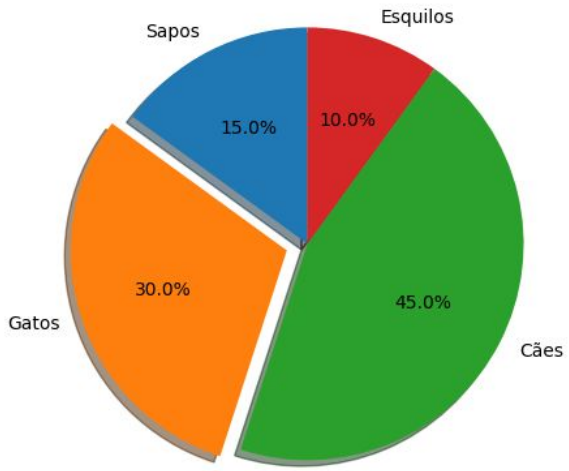


Para encerrar, uma dica preciosa: Para desenhar uma reta que vá de $P_1 = (x_1, y_1)$ até $P_2 = (x_2, y_2)$, basta fazer

```
xd=[x1, x2]
yd=[y1, y2]
plt.plot(xd,yd,'r')  # r=red
```

Tortas

```
import matplotlib.pyplot as plt
# Pie chart, where the slices will be
ordered and plotted counter-clockwise:
labels = 'Sapos', 'Gatos', 'Cães', 'Esquilos'
sizes = [15, 30, 45, 10]
explode = (0, 0.1, 0, 0)  # only "explode" t
                    he 2nd slice (i.e. 'Gatos')
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels,
    autopct='%1.1f%%',shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio
         #ensures that pie is drawn as a circle.
plt.show()
```



Histogramas Desenha um histograma. Seu formato

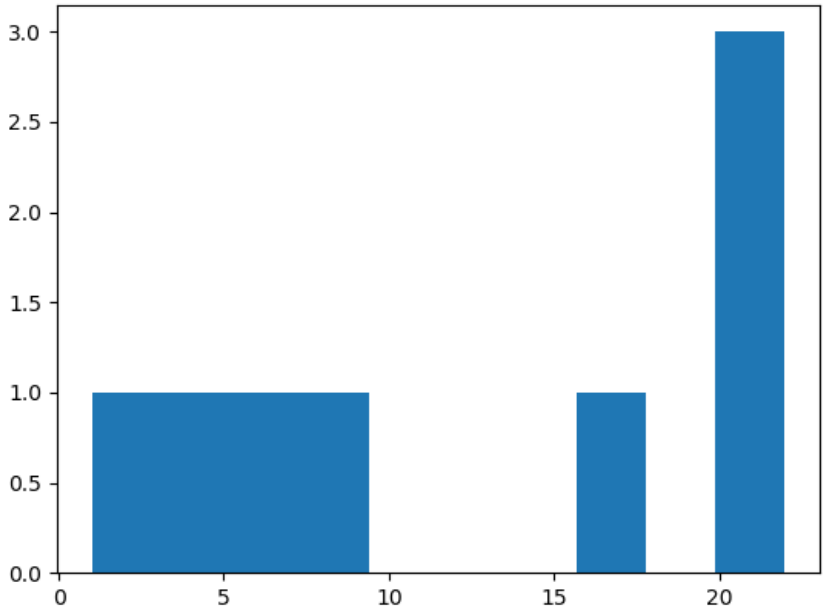
```
hist(x, bins=None, range=None,
density=None, weights=None,
cumulative=False, bottom=None,
histtype='bar', align='mid',
orientation='vertical', rwidth=None,
log=False, color=None, label=None,
stacked=False, normed=None, hold=None,
data=None, **kwargs)
```

x pode ser um array ou uma sequência de arrays, que não precisam ter o mesmo comprimento.

Nota $**kwargs$ é aqui (como em todo o matplotlib) uma lista adicional (e opcional) de palavras chave e argumentos que podem controlar mais aspectos do que está sendo pedido. Veja a documentação, a respeito.

Veja um exemplo simplório:

```
import matplotlib.pyplot as plt
def hist1():
  x=[1,4,7,8,16,22,22,22]
  plt.hist(x)
  plt.show()
  plt.savefig('hist2.png')
hist1()
```



Uma observação final, mas importante: Nos códigos Python a seguir transcritos, os dados de entrada do algoritmo estão escritos diretamente após o código. Não é a maneira mais *ergonômica* de construir uma eventual biblioteca (*toolbox*) de soluções. Para este caso, seria melhor que cada algoritmo fizesse a leitura individual dos dados. Fica para uma eventual evolução destes códigos. O que se fez aqui foi facilitar o uso ao máximo. Além disso, a colocação dos dados junto ao código evita que os dados tenham ser redigitados a cada novo teste do programa.

Capítulo 7

Matlab e Octave

Embora a preferência do autor seja pelo ambiente Python, há concorrentes de peso no pedaço: Se não houvesse alternativa *freeware*, a peleja estaria praticamente decidida, mas há: o Octave embaralha as peças do jogo. Seja como for, segue uma mini-apresentação dos ambientes Matlab e Octave. Vai-se personalizar a apresentação para o Octave, mas ela vale para ambos.

7.1 Ambiente do Octave

A primeira abordagem ao uso do software é na forma de uma calculadora. Para tanto, o Octave abre 3 janelas:

- A janela de comandos que é a usada para inserir comandos e dados
- Janela de gráficos usada para mostrar gráficos
- Janela de edição, que é aquela que permite definir programas/funções

Começando pela janela de comandos, ao ficar disponível ela sinaliza

```
>>
```

Ao lado deste *prompt* pode ser fornecido qualquer comando e ele será prontamente executado pelo Octave. Por exemplo

```
>> 22 * 10
```

Se o resultado não for atribuído explicitamente a nenhuma variável, o Octave criará uma variável de nome *ans* (*answer*). Esta variável pode ser chamada em expressões futuras (embora não seja boa idéia fazer isso). O Octave salta uma linha entre o nome ans e o seu valor. Aqui essa linha será omitida por economia de papel.

A atribuição de variáveis em Octave é proporcionada pelo sinal = como é mais ou menos padrão no mundo do software. Nomes são sensíveis à caixa usada, portanto $a \neq A$.

Para criar um escalar escreve-se $a = 12$, para criar um vetor escreve-se $v =$ [1 2 3 4] e para criar uma matriz escreve-se $m =$ [1 2 3; 4 5 6; 7 8 9]. Aqui o ; separa cada uma das linhas.

Em qualquer ponto da sessão, uma lista das variáveis criadas pode ser obtida pelo comando *who* (só os nomes) e com o comando *whos* (nomes e

estruturas). Para saber o conteúdo de uma das variáveis, basta escrever seu nome e teclar < *enter* >.

O operador dois-pontos, : é usado para gerar sequências. Colocado entre 2 números, gera a o conjunto de números entre eles, com incremento unitário. Se separar 3 números, gera a sequência entre o primeiro e o terceiro com o incremento dado pelo segundo.

```
>> 2:5
ans = 2  3  4  5
>> 2:0.5:4
ans =    2.0000    2.5000    3.0000    3.5000    4.0000

% em resumo: inicio:incremento:final
```

Note que no exemplo acima usou-se o caracter % que denota comentário e não gera nenhum processamento no Octave.

As operações aritméticas em Octave são `+`: adição, `-`: subtração e sinal de negativo, `*`: multiplicação, `/`: divisão real, `^`: exponenciação e `\`: divisão à esquerda (definida apenas na álgebra matricial).

Um detalhe importante é que o Octave espera trabalhar com matrizes. Então `[1 2 3]*[3 4 5]'` é o produto interno entre o vetor linha `[1 2 3]` e o vetor coluna `[3 4 5]'` que vale 26, neste caso. O vetor coluna é a transposta (`'`) de um vetor linha.

Igualmente pode-se obter o produto externo `[3 4 5]' * [1 2 3]`

```
>> [3 4 5]' * [1 2 3]
ans =
    3    6    9
    4    8   12
    5   10   15
```

A multiplicação de matrizes é também facilmente executada. Veja

```
>> a=[1 2 3 4;4 5 6 7]
a =
   1   2   3   4
   4   5   6   7
>> b=[1 2 3;4 5 6;7 8 9; 10 11 12]
b =
    1    2    3
    4    5    6
    7    8    9
   10   11   12
>> a*b
ans =
    70    80    90
   136   158   180
```

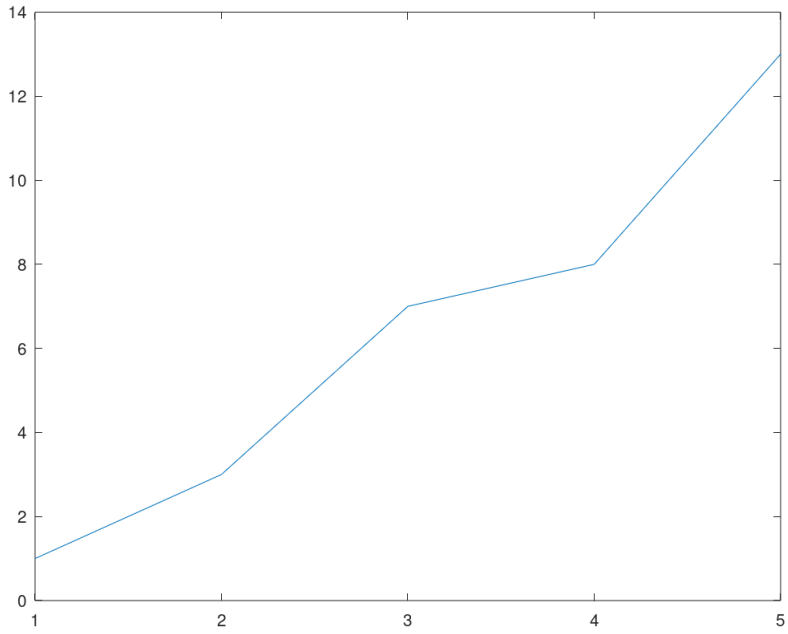
As funções primitivas de Octave são inúmeras e possuem nomes como log, exp, log2, log10, sqrt, abs, sin, acos, tanh, round, ceil, floor, sum, min, max, mean, prod, sort, sqrtm (se $X = sqrtm(m) \Rightarrow X * X = A$),...

Foge ao escopo desta apresentação indicar o que cada uma faz e como opera. Uma saída fácil é chamar `help xxx` onde xxx é o nome da função. Este comando responde com uma boa explicação.

7.2 Gráficos

são facilmente gerados veja por exemplo

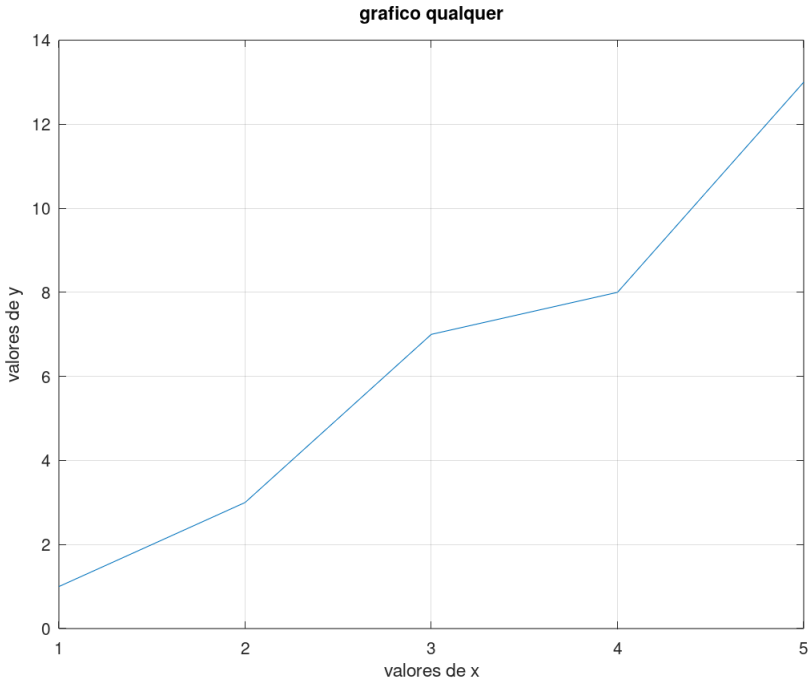
```
>> x=[1 3 7 8 13]
>> plot(x)
```



O gráfico é gerado em outra janela (window) e pode ser manipulado facilmente Para melhorar o gráfico podem ser emitidos os comandos usuais

```
>> title('grafico qualquer')
>> xlabel('valores de x')
>> ylabel('valores de y')
>> grid
```

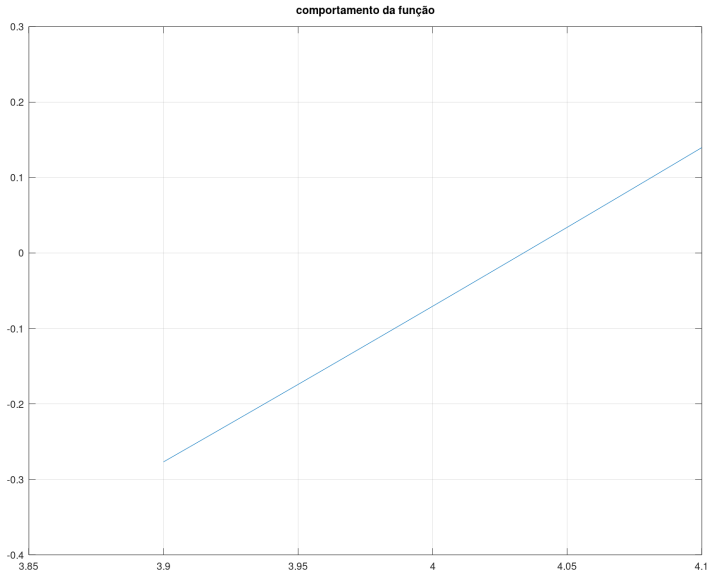
Com o resultado



Eis um exemplo de geração de um gráfico

```
g=@(x) 5.1.^(1+x/8.55)-11;
x=3.9:0.01:4.1;
plot(x,g(x));
title('comportamento da função')
grid
```

Com o seguinte resultado



O Octave tem uma facilidade legal, a saber `lookfor xxx` que responde com todas as referências a `xxx`. Faça uma experiência, execute `lookfor graph` e verá todas as referências a `graph`. Um exercício muito rico.

7.3 Programação

A maneira mais simples de executar algo em Octave é escrevendo um script. Trata-se de um conjunto de comandos que é digitado e salvo na janela `edit`. Depois, na janela de comandos basta fornecer o nome com que o script foi salvo e tudo que está lá será executado como se tivesse sido digitado na janela de comandos. A vantagem é óbvia. O script pode ser executado tantas vezes quantas se quiser simplesmente dando seu nome. Mais ainda, se algum erro for detectado, ele pode ser corrigido diretamente na janela de edit, salvo de novo e executado. Não há necessidade de reescrever todos os comandos a cada execução.

Outra funcionalidade é começar o arquivo com a palavra `function` Depois o formato do arquivo é

```
function saida = nome-função(entrada)
   ...
   saida=...
endfunction
```

Este arquivo obrigatoriamente deverá ser salvo com o nome `nome-função.m` em um diretório que seja o acessado pelo Octave. Deste ponto em diante a função criada passa a se comportar como se fosse primitiva do Octave. As variáveis definidas dentro da função são locais a ela e portanto não existem mais após o final da execução (ao contrário das variáveis definidas em scripts: estas continuam existindo, são globais).

Se houver mais de uma saída, a lista de seus nomes deve estar entre colchetes. Se houver mais de uma entrada, basta escrever seus nomes entre vírgulas dentro do parênteses.

Dentro da programação podem ser usados os comandos específicos como

```
if condição
   ...
[else
   ...]
end
```

Condição pode ser ==: igual, = não igual, >: maior, < menor, >=: maior ou igual, <=: menor ou igual, &: conectivo-e, |: conectivo-or.

Um laço pode ser emitido, como em

```
for indice = início:passo:fim
   ...
end
```

A variável `indice` recebe o valor `inicio`. A seguir ele é comparado com `fim` e se menor ou igual, o `...` é executado. Quando chegar ao final, `indice` é incrementado com o valor `passo` e retorna-se ao início do bloco.

Outro laço pode ser com `while` como em

```
while condição
   ...
end
```

Aqui, a condição é avaliada e se verdadeira, o `...` é executada. Ao final, retorna-se ao início do bloco. Pode-se colocar dentro do bloco um comando `break` que forçará a saída incondicional do bloco.

Vale a referência de que estas estruturas (como em qualquer linguagem de programação) podem ser aninhadas umas dentro de outras. Uma função pode chamar outras, que podem estar no mesmo arquivo.m ou em outros arquivos.m

7.4 Funções anônimas

Uma função anônima é usada para definir inline, sem precisar construir um arquivo.m. Elas são definidas diretamente na janela de comandos usando a síntaxe

```
>> nome=@(parametros) expressao
```

Acompanhe o exemplo

```
>> pit=@(a,b) sqrt(a^2+b^2)
pit =
@(a, b) sqrt (a ^ 2 + b ^ 2)
>> pit(3,4)
ans =  5
```

Foge ao escopo desta apresentação, mas funções podem ser passadas como parâmetros para outras funções generalizando muito a usabilidade do ambiente.

Capítulo 8

Zeros de funções

8.1 Isolamento de raízes

Suponha que precisamos usar o computador para resolver equações. Já paramos para entender o que é resolver uma equação ? Conhece-se a forma geral de uma equação $y = f(x)$. Resolver esta equação vem a ser descobrir quais os valores de x que fazem $f(x)$ ser igual a zero. Se olharmos para o gráfico, veremos que busca-se o valor da abcissa x que determina o cruzamento da curva de $f(x)$ no eixo dos x.

Por exemplo, suponha-se precisar descobrir a raiz 11 do número 1739 ? Ou seja, qual o número que multiplicado por ele mesmo 11 vezes dá como resultado 1739 ?

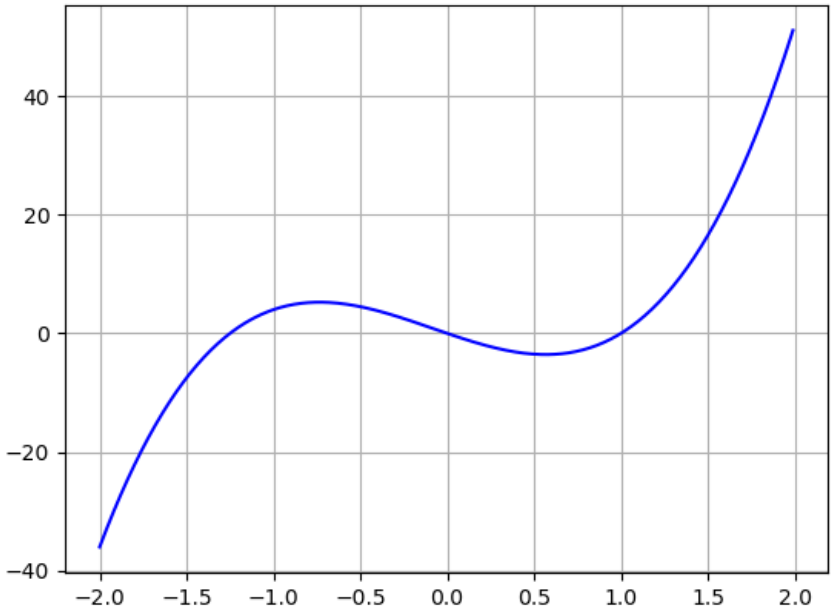
Dizendo isso na forma matemática, pode-se escrever $x^{11} = 1739$, ou melhor ainda $x^{11} - 1739 = 0$. Com isso, chegamos ao ponto.

Antes de continuar, vale uma reflexão: No ensino fundamental e médio investiram-se milhares de horas de ensino e aprendizagem de matemática para resolver inúmeros tipos e classes de equações. Pode dar a impressão que depois disso sabe-se resolver toda e qualquer equação, Nada mais falso, a quantidade do que não se sabe é sempre muito maior do que aquilo que se sabe. Daí a presença de *cálculo numérico* nessa altura da vida.

No material [Alm21], aula S3a, existe um exemplo $f(x) = 8x^3 + 2x^2 - 10x$ e lá está o gráfico gerado por uma programação Octave. Aqui, a equivalente em Python

```
# exemplo de [alm21] s3a, pag.2
import matplotlib. pyplot as plt
import numpy as np
def f(x):
  return 8*x**3 + 2*x**2 - 10*x
x1=np.arange(-2,2,0.01)
plt.plot(x1,f(x1),'b-')
plt.grid()
plt.show()
```

Com a geração do seguinte gráfico



O mesmo resultado pode ser obtido usando o pacote sympy, com a vantagem de ter-se as raízes calculadas

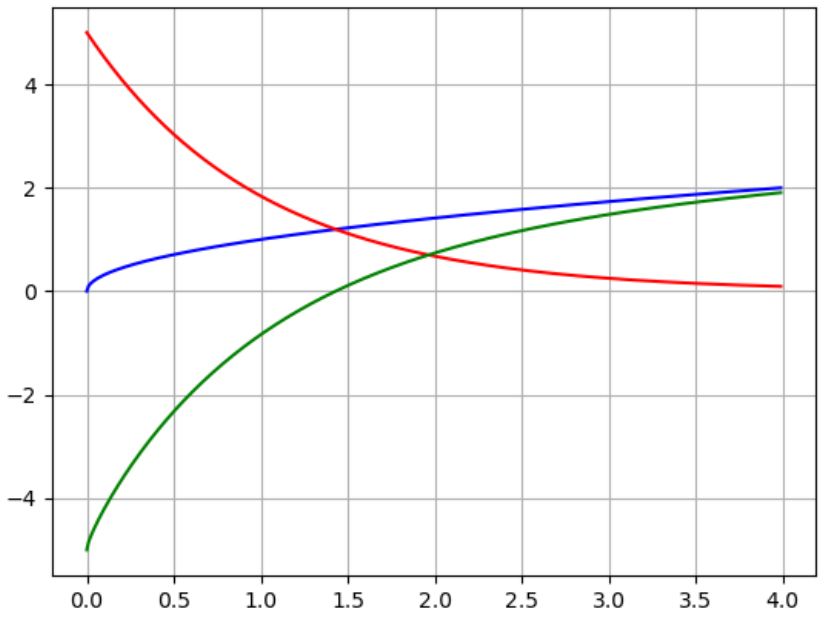
```
# exemplo de [alm21] s3a, pag.2
import matplotlib. pyplot as plt
import numpy as np
import sympy as sp
x=sp.symbols('x')
f=sp.symbols('f',cls=sp.Function)
f=8*x**3 + 2*x**2 - 10*x
print(sp.solve(f,x))
x1=np.arange(-2,2,0.01)
y1=[0.0]*len(x1)
for i in range(len(x1)):
  y1[i]=f.subs(x,x1[i])
plt.plot(x1,y1,'b-')
plt.grid()
plt.show()

[-5/4, 0, 1]
```

Ainda no material citado há um exercício que pede manipular a função $f(x) = \sqrt{x} - 5e^{-x}$ onde e é a constante de Euler. Determine $g(x)$ e $h(x)$ de formar que $f(x) + g(x) + h(x)$ ou seja para $f(x) = 0 \Rightarrow g(x) = h(x)$. Esboce $f(x)$, $h(x)$ e $g(x)$. Supondo $h(x) = \sqrt{x}$ e $g(x) = 5e^{-x}$ o seguinte programa Python resolve nosso problema

```
import matplotlib. pyplot as plt
import numpy as np
import sympy as sp
```

```
x=sp.symbols('x')
f,g,h=sp.symbols('f g h',cls=sp.Function)
f=x**0.5-5*sp.exp(-x)
g=x**0.5
h=5*sp.exp(-x)
print(sp.solve(f,x))
x1=np.arange(0,4,0.01)
f1=[0.0]*len(x1)
g1=[0.0]*len(x1)
h1=[0.0]*len(x1)
for i in range(len(x1)):
  f1[i]=f.subs(x,x1[i])
  g1[i]=g.subs(x,x1[i])
  h1[i]=h.subs(x,x1[i])
plt.plot(x1,g1,'b-')
plt.plot(x1,h1,'r-')
plt.plot(x1,f1,'g-')
plt.grid()
plt.show()

[1.43044508899111]
```

e gera a seguinte imagem



8.2 Bissecção

Se se olhar o gráfico de uma função, ver-se-á que no entorno da raiz, de um lado a função é positiva, e do outro lado é negativa ou vice versa, já que a curva cruza o eixo x. Se a função for contínua, é óbvio. Este método começa pela seleção de 2 pontos x_a e x_b sendo que a função $y = f(x)$ têm sinais opostos nesses dois pontos. Graças a essa constatação, pode-se afirmar que a raiz procurada está entre x_a e x_b. Dizendo de outra maneira. O valor de x

procurado, está entre a e b. O Método da bissecção vai dividindo o intervalo a, b pela metade, questionando a seguir em qual das metades a curva cruza o eixo e consequentemente o x está. Descoberto isso, despreza-se a metade que não contém x. Levando este processo ao limite, descobre-se qual o valor de x aceitável para este problema. Em resumo, eis as etapas do método:

1. Arbitrar intervalo em $[a, b]$ que compreenda a raiz.
2. Arbitrar um erro permitido ϵ
3. Calcular $x_m = (a + b)/2$
4. Se $f(a).f(x_m) < 0$ o novo intervalo será $[a, x_m]$
5. Se $f(a).f(x_m) > 0$ o novo intervalo será $[x_m, b]$
6. Enquanto o erro $b - a > \epsilon$ voltar ao passo 3.

Seja um exemplo, calcular $f = e^x - sin(x) - 2 = 0$ nos limites $a = 1$ e $b = 1.25$ com $\epsilon = 0.01$. No primeira etapa tem-se

```
Limite Superior: 1.25
Limite Inferior: 1
Tolerancia: 0.01
  -----a  -----b  ----xm  ---fxm  ----fa  ----fb  --erro
  1.1250  1.0000  1.1250  0.1779  0.5414 -0.1232  0.1250
  1.0625  1.0000  1.0625  0.0200  0.1779 -0.1232  0.0625
  1.0625  1.0312  1.0312 -0.0534  0.0200 -0.1232  0.0312
  1.0625  1.0469  1.0469 -0.0171  0.0200 -0.0534  0.0156
  1.0547  1.0469  1.0547  0.0013  0.0200 -0.0171  0.0078
Raiz:  1.05078125
Erro:  -0.0078125
```

Eis o programa Python que resolve este caso

```
import numpy as np
def bissec():
  a=float(input("Limite Superior: "))
  b=float(input("Limite Inferior: "))
  tol=float(input("Tolerancia: "))
  print("  ----a  ----b  ---xm  --fxm  ---fa  ---fb  -erro")
  while np.abs(b-a)>tol:
    xm=(a+b)/2
    fxm=np.exp(xm)-np.sin(xm)-2
    fa=np.exp(a)-np.sin(a)-2
    fb=np.exp(b)-np.sin(b)-2
    if fa*fxm>0:
      a=xm
    else:
      b=xm
    print(':8.4f:8.4f:8.4f:8.4f:8.4f
   :8.4f:8.4f'.format(a,b,xm,fxm,fa,fb,np.abs(b-a)))
  xm=(a+b)/2
  print("Raiz: ",xm)
  print("Erro: ",b-a)

bissec()
```

Uma implementação mais elegante do mesmo código, para a função $5.1^{1+\frac{x}{8.55}} - 11$, nos limites entre 3.9 e 4.1 pode ser

```
import numpy as np
def bissec(a,b,tol):
  def fu(x):
    return 5.1**(1+x/8.55)-11
  while np.abs(b-a)>tol:
    xm=(a+b)/2
    print('a=',a,' b=',b,' xm=',xm)
    fxm,fa,fb=fu(xm),fu(a),fu(b)
    if fa*fxm>0:
      a=xm
    else:
      b=xm
  xm=(a+b)/2
  print("Raiz: ",xm)
  print("Erro: ",np.abs(a-b))
bissec(3.9,4.1,0.0001)
```

A mesma implementação em Octave:

```
function res=bissec(a,b,tol)
  function r=fu(x)
    r=5.1.^(1+x/8.55)-11;
  endfunction
  while abs(b-a)>tol
    xm=(a+b)/2;
    fxm=fu(xm);
    fa=fu(a);
    fb=fu(b);
    if fa*fxm>0
      a=xm;
    else
      b=xm;
    endif
    xm=(a+b)/2;
  endwhile
  res=xm;
  return
endfunction
```

8.2.1 Qualidades do método da bissecção

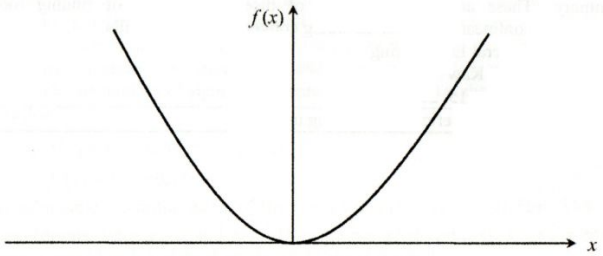
- Sempre converge. Desde que o intervalo dado contenha a raiz, o método acha a raiz.
- A cada interação o intervalo é reduzido à metade. Dessa maneira pode-se calcular o erro na solução da equação.

8.2.2 Defeitos do método da bissecção

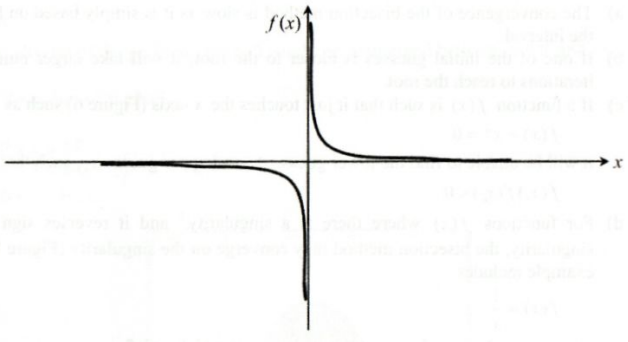
- A convergência é lenta, já que baseada apenas na divisão por 2 do intervalo.

- Se um limites está próximo à raiz, a quantidade de interações é grande até chegar na raiz.

- Se a $f(x)$ toca o eixo x na raiz (como em $f(x) = x^2 = 0$ o método pode ser incapaz de achar os intervalos inferior x_i e superior x_s tal que $f(x_i).f(x_s) < 0$. Veja a figura

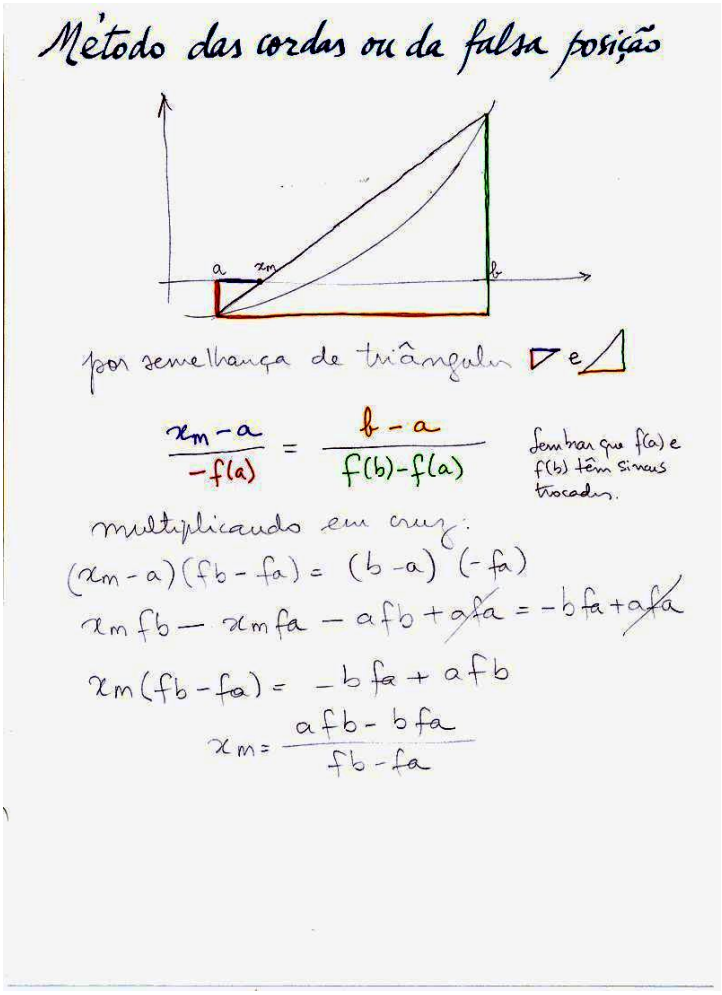


- Se a $f(x)$ tem uma singularidade (torna-se infinita) e inverte o sinal na singularidade, como por exemplo em $f(x) = \frac{1}{x}$. Entretanto aqui, a função é descontínua e o teorema que informa que a raiz existe não é aplicável.



  Note que no exemplo acima, a função $f(x) = \frac{1}{x} = 0$ não tem raiz, mas ela troca de sinal.

8.3 Falsa Posição



O método começa com uma análise do gráfico da função nas proximidades da raiz. Por semelhança de triângulos, pode-se obter a formulação proposta. Eis os passos do algoritmo:

1. Arbitrar um intervalo $[a, b]$ que compreende a raiz
2. Arbitrar ϵ
3. Calcular $x_m = \frac{a.f(b)-b.f(a)}{f(b)-f(a)}$
4. Se $f(a).f(x_m) < 0$ o novo intervalo será $[a, x_m]$
5. Se $f(a).f(x_m) > 0$ o novo intervalo será $[x_m, b]$
6. Enquanto o erro $f(x_m) > \epsilon$ voltar ao passo 3

Veja a seguir a solução da mesma equação, $(e^x - sen(x) - 2 = 0)$ com os mesmos intervalos do método anterior.

```
Limite inferior: 1
Limite superior: 1.25
Tolerância: 0.001
  -----a  -----b  ----xm  ---fxm  ----fa  ----fb  --erro
  1.0463  1.2500  1.0463 -0.0184 -0.1232  0.5414
  1.0530  1.2500  1.0530 -0.0026 -0.0184  0.5414
  1.0540  1.2500  1.0540 -0.0004 -0.0026  0.5414
Raiz:  1.0539728656764054
Erro:  -0.0003663936835978099
```

Eis o programa Python que resolve este caso

```
from numpy import *
def cordas():
  a=float(input("Limite inferior: "))
  b=float(input("Limite superior: "))
  tol=float(input("Tolerância: "))
  fxm=1
  print("  ----a  ----b  ---xm  --fxm  ---fa  ---fb  -erro")
  while abs(fxm)>tol:
    fa=exp(a)-sin(a)-2
    fb=exp(b)-sin(b)-2
    xm=((a*fb)-(b*fa))/(fb-fa)
    fxm=exp(xm)-sin(xm)-2
    if fa*fxm>0:
      a=xm
    else:
      b=xm
    print(':8.4f:8.4f:8.4f:8.4f:8.4f
       :8.4f'.format(a,b,xm,fxm,fa,fb))
  print("Raiz: ",xm)
  print("Erro: ",fxm)
cordas()
```

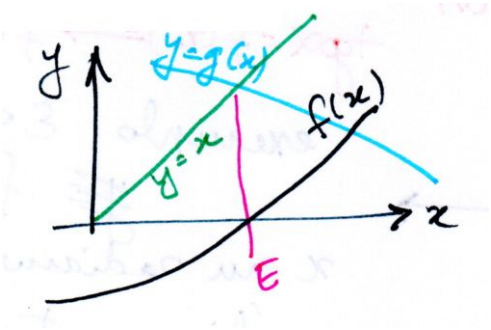
A mesma implementação mais genérica para este método, na função $5.11^{1+\frac{x}{8.55}} - 11$ entre 3.9 e 4.1 pode ser

```
import numpy as np
def fp(a,b,tol):
  def fu(x):
    return 5.1**(1+x/8.55)-11
  while np.abs(b-a)>tol:
    xm=(a*fu(b)-b*fu(a))/(fu(b)-fu(a)) # cuidado com ()s
    print('a=',a,' b=',b,' xm=',xm)
    if fu(a)*fu(xm)>0:
      a=xm
    else:
      b=xm
  print("Raiz: ",xm)
  print("Erro: ",np.abs(a-b))

fp(3.9,4.1,0.0001)
```

8.4 Iteração Linear

Este método merece ser estudado mais pelos conceitos que ele apresenta do que pela sua eficiência computacional. Considerando uma $f(x)$ contínua em $[a, b]$ intervalo que garantidamente contém uma raiz da equação $f(x) = 0$. Neste método deve-se transformar a $f(x)$ em um sistema formado por $y = x$

e $y = g(x)$ como se vê em



Ao fazer isto, transforma-se o problema de encontrar uma raiz de $f(x)$ no problema de encontrar um ponto fixo de $g(x)$. Neste caso, $g(x)$ é chamada função de iteração para $f(x)$. Há uma certa *arte* envolvida aqui na escolha de $g(x)$. A partir de uma aproximação inicial x_0 pode-se gerar uma sequência x_k de aproximações à raiz pela relação $x_{k+1} = g(x_k)$. A função $g(x)$ é chamada função de iteração.

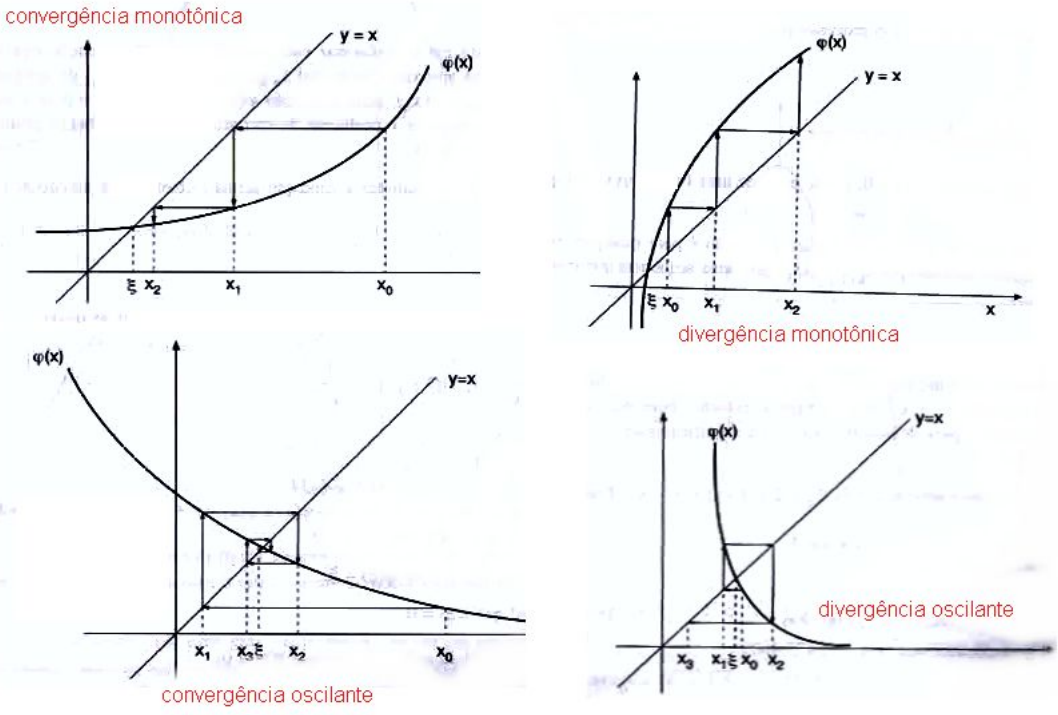
Veja-se um exemplo. Suponha $f(x) = x^3 - 5x + 3 = 0$. Para achar $g(x)$ deve-se estudar $f(x)$ "isolando" um x do lado esquerdo para cada candidato a $g(x)$. São candidatos a $g(x)$:

$g(x) = \frac{x^3+3}{5}$
$g(x) = \frac{5x-3}{x^2}$
$g(x) = \sqrt[3]{5x-3}$
$g(x) = \frac{-3}{x^2-5}$

Nessa situação, pode-se ter 4 casos de convergência a depender da derivada primeira de $g(x)$, a saber:

convergência monotônica:	$0 < \|g'(x)\| < 1$
convergência oscilante:	$-1 < \|g'(x)\| < 0$
divergência monotônica:	$\|g'(x)\| > 1$
divergência oscilante:	$\|g'(x)\| < -1$

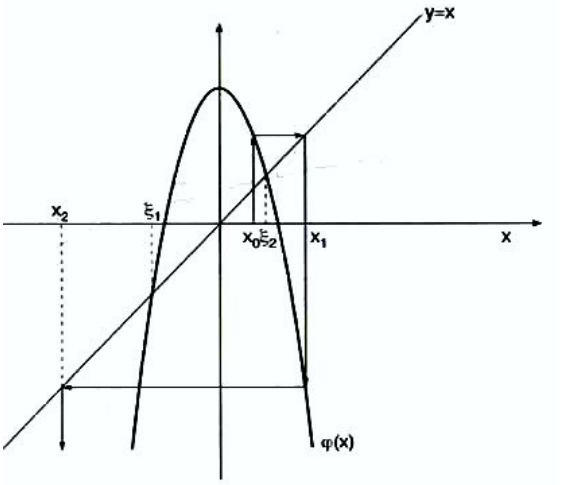
Veja-se na figura a seguir os casos acima (adaptado de [Rug88])



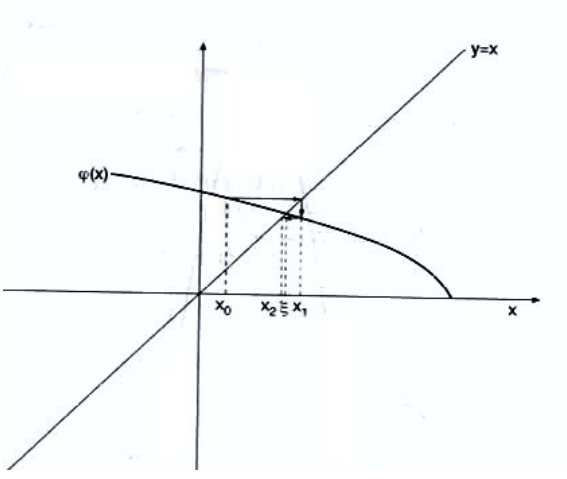
Acompanhe mais um exemplo. Seja estimar $x^2 + 2x - 4$ com $\epsilon \leq 2 \times 10^{-1}$ no intervalo $[1, 1.5]$.

A escolha de $g(x)$ passa por
$g(x) = \frac{x^2-4}{-2} \rightarrow g'(x) = \frac{-2x}{2}$ divergente pois $|g'| > 1$
$g(x) = \frac{4}{x+2} \rightarrow g'(x) = \frac{-4}{(x+2)^2}$ convergente pois $|g'| < 1$
$g(x) = \sqrt{4-2x} \rightarrow g'(x) = \frac{-2}{2\sqrt{4-2x}}$ convergente pois $|g'(x)| < 1$
A escolhida é $g = \sqrt{4-2x} \rightarrow$oscilante já que $-1 < g'(x) < 0$. O extremo inicial $x_0 = 1$ pois $|g'(1)| < |g'(1.5)|$ e a raiz desejada é x=1.31...

Mais um exemplo, este apenas pedagógico: seja achar as raízes da equação $x^2 + x - 6$. (por Báskara, sabe-se que $\xi_1 = -3$ e $\xi_2 = 2$. Considerando $g(x) = 6 - x^2$. Executando o método com o chute inicial $x_0 = 1.5$
$x_1 = g(x_0) = 6 - 1.5^2 = 3.75$
$x_2 = g(x_1) = 6 - 3.75^2 = -8.0625$
$x_3 = g(x_2) = -59.003906$
$x_4 = g(x_3) = -3475.4609$
e como se pode ver esta série está divergindo como se pode ver na figura

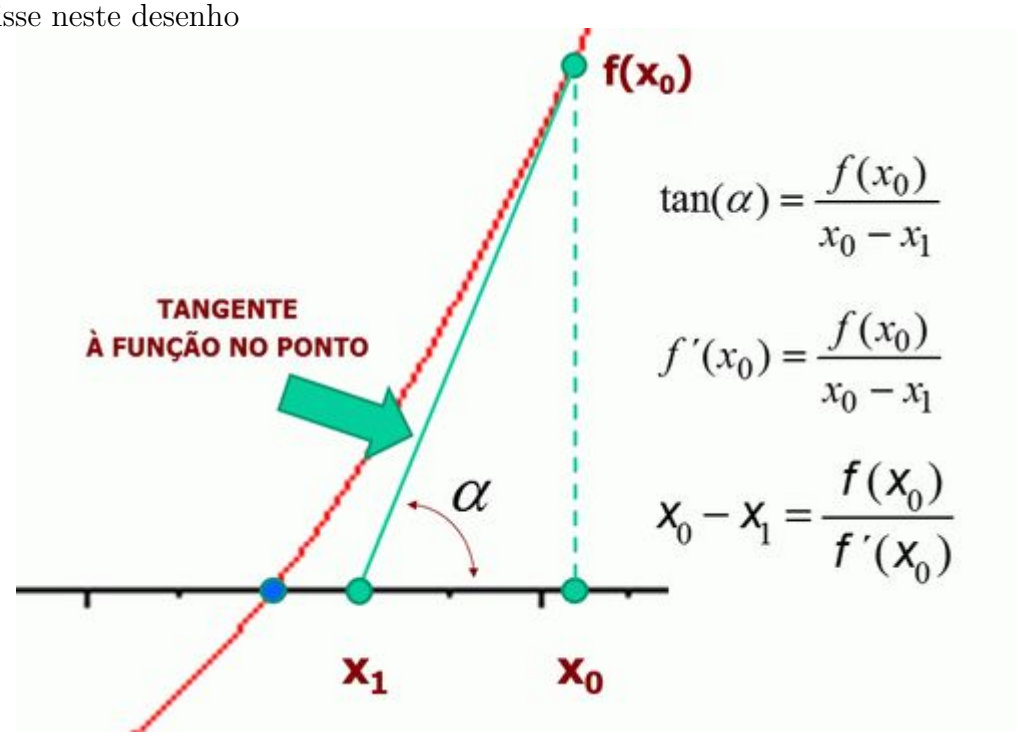


Usando agora $g(x) = \sqrt{6-x}$, fica:
$x_1 = g(x_0) = \sqrt{6-1.5} = 2.12132$
$x_2 = g(x_1) = \sqrt{6-2.12132} = 1.96944$
$x_3 = g(x_2) = 2.00763$
$x_4 = g(x_3) = 1.99809$
$x_5 = g(x_4) = 2.00048$ e a sequência está convergindo para $\xi_2 = 2$ como se pode ver em



8.5 Newton-Raphson

Este método começa com a interpretação geométrica da derivada. Lembrando, Newton é um monstro da ciência. Entre outras coisas, ele criou o cálculo diferencial e integral, a teoria da luz, a gravitação, as leis do movimento, além de, no fim da vida ter – como funcionário público – dirigido a casa da moeda inglesa, pendurando na forca alguns falsários delinquentes. Voltando ao cálculo diferencial, uma das interpretações da derivada de uma função em um ponto é a tangente da curva da função naquele ponto. Veja-se o que se disse neste desenho



Eis as etapas do método:

1. Obter a derivada da função a resolver
2. Arbitrar um ponto inicial x_0 para a função
3. Arbitrar o erro ϵ
4. Calcular $f(x_0)$ e $df(x_0)$
5. Calcular o "novo" x, $x = x_0 - fx/dfx$
6. Se $|fx| > \epsilon$ retornar ao passo 4

Por exemplo, seja calcular as raízes da mesma equação acima que é $e^x - sen(x) - 2 = 0$. A derivada é $\frac{\partial}{\partial x} = e^x - cos(x)$. Obteve-se a seguinte tabela

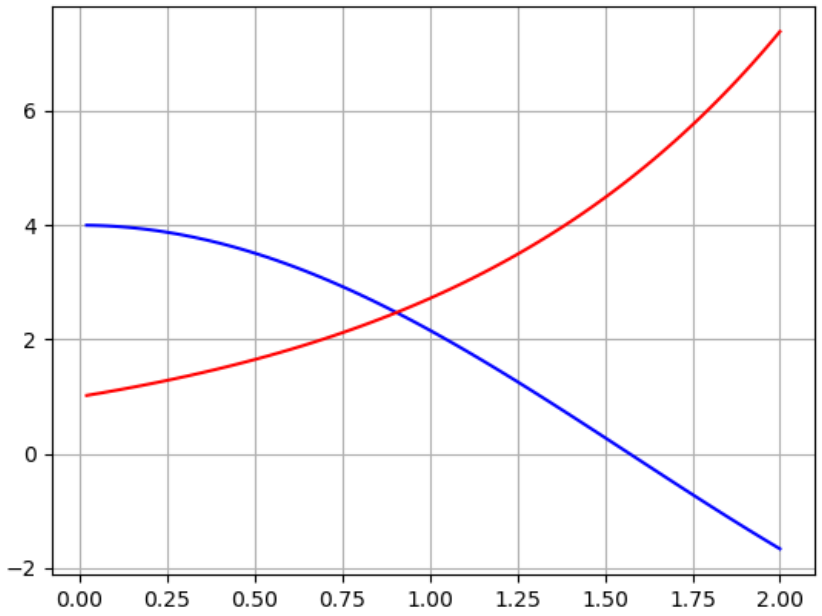
```
   --------x   -----------------fx   -------dfx
   1.0000000 -0.12318915634885141    2.1779795
   1.0565612  0.00579321190120519    2.3845934
   1.0541318  0.00001105001756940    2.3754999
   1.0541271  0.00000000004045120    2.3754825
Raiz:  1.0541271241082415
Erro:  4.0451197946822504e-11
```

Eis o programa Python que resolve este caso

```
from numpy import *
```

```
def newton1():
  x=1
  tol=0.00001
  fx=exp(x)-sin(x)-2
  dfx=exp(x)-cos(x)
  print("   --------x   ----------------fx   -------dfx")
  print(":12.7f :20.17f :12.7f".format(x,fx,dfx))
  while abs(fx)>tol:
    x=x-fx/dfx
    fx=exp(x)-sin(x)-2
    dfx=exp(x)-cos(x)
    print(":12.7f :20.17f :12.7f".format(x,fx,dfx))
  print("Raiz: ",x)
  print("Erro: ",fx)
newton1()
```

Mais um exemplo baseado em similar de [Fra07], pág 77. Determinar, usando o método de Newton-Raphson a menor raiz positiva da equação $4cos\ x - e^x = 0$ com erro inferior a 10^{-2}. Para obter um valor inicial deve-se desenhar a função em um gráfico. Para isto, pode-se dividir a equação em duas mais simples, chamadas y_1 e y_2. Note que para este procedimento usou-se apenas a $f(x) = 0$ original. Fazendo $y_1 = 4cos\ x$ e $y_2 = e^x$ e colocando ambas no mesmo gráfico fica-se com (y_1 em azul e y_2 em vermelho)



Claramente a intersecção das duas curvas é o ponto procurado ($f(x) = 0$) já que quando o valor de ambas y_i é igual, a subtração de uma da outra dá 0. Depois da análise do gráfico pode-se tomar $x_0 = 1.0$. Da equação original se obtém $f(x) = 4cos\ x - e^x$ e $f'(x) = -4sen\ x - e^x$. A propósito se tiver dificuldade com a derivada, faça [1]

[1] Este trecho foi singelamente escrito com $math.cos(x) - math.exp(x)$ usando-se as constantes do pacote math. Entretanto este procedimento dá origem ao erro "can't convert expression to float" causado pela mistura do pacote math (numérico) com o sympy (simbólico)

```
import sympy as sp
def nr2():
  f,g=sp.symbols('f g',cls=sp.Function)
  x=sp.symbols('x')
  f=4*sp.cos(x)-sp.exp(x)
  g=sp.diff(f,x)
  sp.pprint(g)
nr2()
```

Para obter a raiz, faça

```
import numpy as np
import math
def newt2():
  x=1.0
  tol=0.00000000001
  ct=0
  fx=4*math.cos(x)-math.exp(x)
  dfx=-4*math.sin(x)-math.exp(x)
  while abs(fx)>tol:
    x=x-fx/dfx
    fx=4*math.cos(x)-math.exp(x)
    dfx=-4*math.sin(x)-math.exp(x)
    print('%15.8f    %3.15f     %10.8f ' %(x,fx,dfx))
  print('raiz = ',x)
  print('erro = ',fx)
newt2()
```

Com o seguinte resultado

```
     0.90843895    -0.020537584152596     -5.63462617
     0.90479406    -0.000032822284591     -5.61661464
     0.90478822    -0.000000000084398     -5.61658576
     0.90478822    0.000000000000000     -5.61658576
raiz =  0.9047882178730188
erro =  4.440892098500626e-16
```

A mesma implementação de $y = 5.1^{1+\frac{x}{8.55}} - 11$ pode ser

```
import sympy as sp
def fex4():
  x,y,z=sp.symbols('x,y,z')
  y=5.1**(1+x/8.55)-11
  z=sp.diff(y,x)
  print(y) # deu: 5.1**(0.116959064327485*x + 1) - 11
  print(z) # deu: 0.190554449091261*5.1**
               (0.116959064327485*x + 1)
fex4()
def fex5(vi):
  def fx(x):
    return 5.1**(1+x/8.55)-11
  def dfx(x):
    return 0.190554*5.1**(0.116959*x+1)
```

```
  eps=0.000001
  velho,novo=vi,vi
  while abs(fx(novo))>eps:
    velho=novo
    novo=velho-fx(novo)/dfx(novo)
    print(velho,novo,fx(novo),dfx(novo))
  print(novo)
fex5(4.0)
#4.0 4.033889086612887 0.00022857716830593233 2.096136670106631
#4.033889086612887 4.033780039730798 1.7394956586258559e-09
      2.096093114186739
#4.033780039730798
```

O mesmo código feito em Octave

```
function r=newt(vi)
  function re=fx(x)
    re=5.1.^(1+x./8.55)-11;
  endfunction
  function re=dfx(x)
    re=0.190554*5.1.^(0.116959*x+1);
  endfunction
  eps=0.000001;
  velho=vi;
  novo=vi;
  while abs(fx(novo))>eps
    velho=novo;
    novo=velho-fx(novo)/dfx(novo);
  endwhile
  r=novo;
  return
endfunction
```

8.5.1 Problemas no uso de Newton-Raphson

- Divergência perto de pontos de inflexão. Um ponto de inflexão ocorre quando a concavidade da curva muda. A condição necessária (mas não suficiente) é $f''(x) = 0$ Se o ponto inicial (ou algum ponto calculado) cair perto deste ponto, o algoritmo pode divergir.

- Divisão por zero. Novamente, se um ponto gerar divisão por zero, ou por próximo de zero, deixa de haver convergência.

- Proximidade a máximo ou mínimo. Se houver esta proximidade o ponto pode convergir para o máximo ou mínimo ao invés da raiz.

- Salto entre raízes. Havendo várias raízes, deve-se escolher um valor inicial próximo a uma raiz. Entretanto o método pode dar um salto e convergir para outra raiz. (Veja-se [Kaw17], cap 3.4 para ver exemplos numéricos destes problemas)

8.6 Secante

Definição: secante a uma curva é qualquer reta que cruza dois ou mais pontos dessa curva. No contexto em que estamos, a secante $\rightarrow$ tangente à medida em que $x \rightarrow 0$. Considerando que uma desvantagem do método de Newton-Raphson é a necessidade de obter $f'(x)$ pode-se contornar essa *dificuldade* substituindo a derivada $f'(x)$ pelo quociente das diferenças

$$f'(x_k) = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

onde x_k e e x_{k-1} são duas aproximações para a taiz. Sendo assim, no começo do método são necessárias duas aproximações para a raiz. A função de iteração fica

$$\varphi(x_k) = x_k - \frac{f(x_k)}{\frac{f(x_k)-f(x_{k-1})}{x_k - x_{k-1}}}$$

ou simplificando

$$\varphi(x) = x_{k+1} = \frac{x_{k-1}.f(x_k) - x_k.f(x_{k-1})}{f(x_k) - f(x_{k-1})}$$

Exemplo Usando o método da secante para calcular $\sqrt[3]{5}$. Fica $x = \sqrt[3]{5}$ e $x^3 = 5$ e $x^3 - 5 = 0$. Como $1^3 = 1 < 5$ e $2^3 = 8 > 5$ a raiz está entre 1 e 2. Usando estes valores como $x_0 = 1$ e $x_1 = 2$ fica

iter	x_k	$f(x_k)$
0	1	-4
1	2	3
2	1.57142	-1.11953
3	1.68789	-0.19117
4	1.71188	0.01674
5	1.70995	-0.00021
6	1.70997	0.000000

e $\sqrt[3]{5} \approx 1.70997$

8.7 Misto

Trata-se da aplicação em sequência do método de Newton-Raphson e da falsa posição, nesta ordem. Assim,

$$x_n = \frac{x_n^{NR} + x_n^{FP}}{2}$$

O objetivo é garantir uma convergência mais rápida. Seja o exemplo: Calcular $x^2 + 3x - cos(x) - 2.45 = 0$ no intervalo $[0, 1]$ com $\varepsilon \leq 0.003$.

NR

$f'(x) = 2x + 3 + sen(x)$ e $f(x) > 0$ para $x \in [0, 1]$

$f''(x) = 2 + cos(x) \rightarrow f''(x) > 0$ para $x \in [0, 1]$

sendo $f(0) = -3.45$, $f(1) = 1.0097$ e $f''(x) > 0$ então $x_0 = 1$ pois $f(1) \times f''(x) > 0$.

$NR \rightarrow x_1 = 0.8272$ e o extremos a trocar é $[0, 0.8272]$

FP

$x^{FP} = a - \frac{(b-1)f(a)}{f(b)-f(a)} = 0 - ... = 0.8180$ $\varepsilon = |0.8180 - 0.8272| = 0.0092 > \varepsilon$ o extremo a trocar é $[0.8180, 0.8272]$

NR
$x_2 = x_1^{FP} - \frac{f(x^{FP})}{f'(x^{FP})} = 0.8200$ e o extremo a trocar é $[0.8180, 08200]$. Pergunta: Parar ? $|x_2^{FP} - x_2^{NR}| < \varepsilon$? Sim, então $x_2^{FP} = 0.8180$ e a raiz procurada é $\frac{0.8180+0.8200}{2} = 0.8190$

8.8 Mandando o sympy calcular

A rigor, este nem é um método a estudar, mas já que se está com a mão na massa (em termos computacionais), a maneira de comandar a solução da equação é

```
import sympy as sp
def resolver():
  x,y=sp.symbols('x y')
  y=5.1**(1+x/8.55)-11
  z=sp.solve(y,x)
  return z
print(resolver())
>> [4.03378003890093]
```

O Octave também possui seu processador simbólico, que na verdade é o sympy. Entretanto, aqui a interface é a do Octave e não a do Python. Eis como seria solucionada a mesma equação

```
pkg install -forge symbolic
pkg load symbolic

syms x
x=5.1.^(1+x/8.55)-11
y=solve(x)
eval(y)
>> ans = 4.0338
```

Capítulo 9

Sistemas Lineares

> The need to solve linear systems arises in an estimated 75% of all scientific computed problems (Dahlquist & Bjorck in *Numerical Methods*, 1974).

9.1 Equações e Sistemas Lineares

Uma equação é linear se cada um de seus termos contém não mais do que uma variável e se cada variável aparece na primeira potência. Por exemplo $3x+2y+3z=8$ é linear, mas $3x+2yz+3w=27$ não é, pois o segundo termo tem yz e também $2x^2+4y+5z=90$ também não é linear, pois aparece x^2.

Vamos considerar n equações lineares com n incógnitas e vamos nos referir a esse conjunto como um Sistema Linear de ordem n. Uma solução para tal sistema é um conjunto de n valores para as incógnitas, de maneira a que quando esses valores substituem as variáveis nas esquações, todas elas são simultaneamente satisfeitas. Por exemplo, o sistema

$$\left\{\begin{array}{rcrcrcl} x & + & y & + & z & = & 1 \\ x & - & y & - & z & = & 1 \\ 2x & + & 3y & - & 4z & = & 9 \end{array}\right.$$

tem a solução $x=1$, $y=1$ e $z=-1$. Você pode verificar e se certificar disso substituindo estas variáveis no sistema. O mesmo sistema também pode ser escrito na forma matricial

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & -1 & -1 \\ 2 & 3 & -4 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 9 \end{pmatrix}$$

Para quem, como nós, conhece a multiplicação matricial, o sistema acima também pode ser escrito como

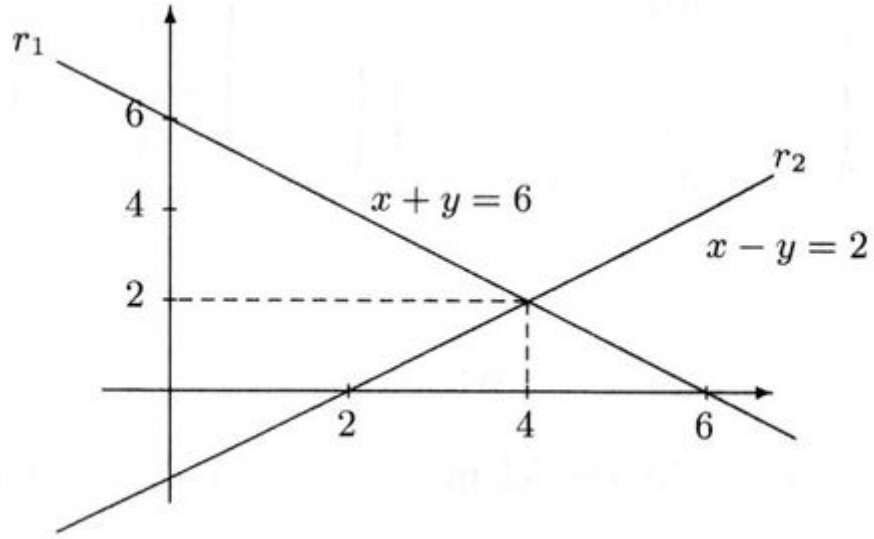
$$A.x = b,$$

onde A é chamada Matriz de Coeficientes, b é o termo independente e x é o vetor solução. Dado um sistema linear qualquer o mesmo pode ser classificado em termos do número de soluções que ele admite:

- Sistema possível com solução única. Por exemplo, seja

$$\left\{\begin{array}{rcrcl} x & + & y & = & 6 \\ x & - & y & = & 2 \end{array}\right.$$

Este sistema, que pode ser representado na figura

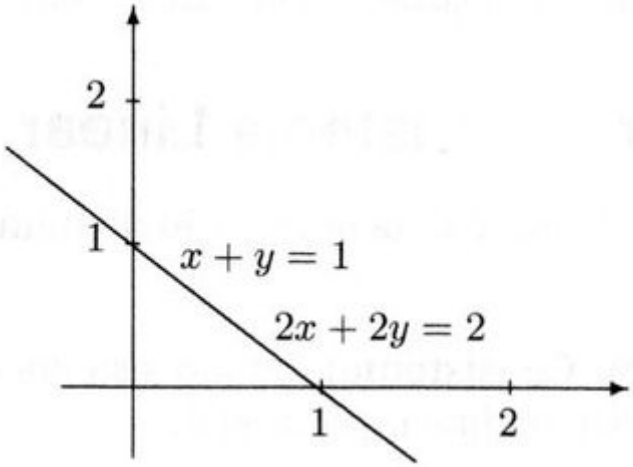


admite uma única solução, representada pelo ponto $(4, 2)$.

- Sistema possível com infinitas soluções. Por exemplo, seja

$$\begin{cases} x + y = 1 \\ 2x + 2y = 2 \end{cases}$$

Como se pode ver na figura

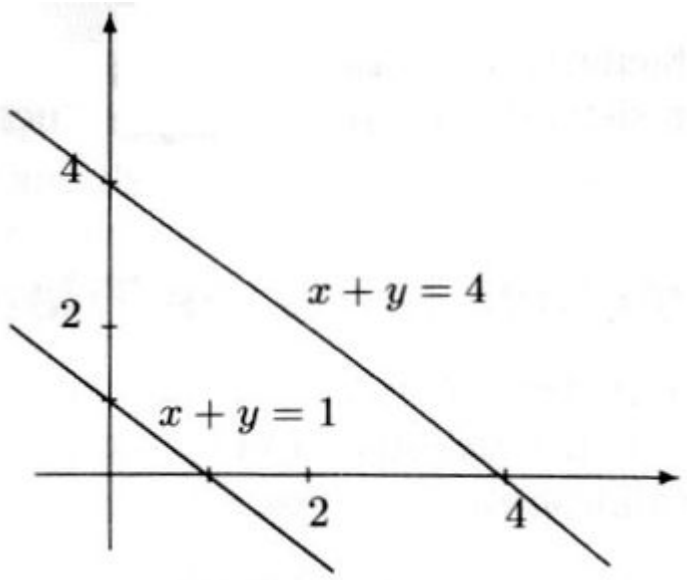


as soluções são infinitas (sobre toda a linha)

- Sistema impossível, sem solução. Por exemplo

$$\begin{cases} x + y = 1 \\ x + y = 4 \end{cases}$$

Como se pode ver na figura, as retas são paralelas e portanto não têm ponto comum.



9.2 Método de Escalonamento de Gauss

Também conhecido como método de escalonamento de Gauss, ele é indicado para sistemas densos (poucos zeros na matriz de coeficientes) é conhecido por isso mesmo como sistema *näive (ingênuo)* de Gauss. A presença de zeros pode dificultar ou inviabilizar a utilização do método pela proibição de divisão de alguma coisa por zero.

Resolver um sistema linear $A.x = b$ é achar um sistema equivalente $A'.x = b'$ que tenha solução imediata, ou pelo menos mais simples do que o sistema original. Dois sistemas são ditos equivalentes se, embora diferentes, têm a mesma solução. As operações que são realizadas sobre os sistemas e que não lhes alteram as soluções, são:

- Trocar uma linha (equação) de lugar com outra.
- Multiplicar uma linha (equação) por uma constante
- Trocar uma linha por uma soma de 2 linhas

Seja por exemplo, resolver o sistema

$$\left\{\begin{array}{rcrcrcl} x & + & 4y & + & 4z & = & 23 \\ 3x & + & 3y & + & z & = & 27 \\ 2x & + & 5y & + & 2z & = & 25 \end{array}\right.$$

Passando para uma matriz mais simples

$$\left(\begin{array}{ccc|c} 1 & 4 & 4 & 23 \\ 3 & 3 & 1 & 27 \\ 2 & 5 & 2 & 25 \end{array}\right)$$

aplicando à terceira linha a propriedade que diz $L_3 = L_3 - p.L_1$ e escolhendo p de modo que surja um zero na primeira posição da linha, fica:$2, 5, 2, 25 - p.1, 4, 4, 23$ e escolhendo $p = 2$, a nova linha passa a ser $0, -3, -6, -21$ e a matriz fica:

$$\left(\begin{array}{ccc|c} 1 & 4 & 4 & 23 \\ 3 & 3 & 1 & 27 \\ 0 & -3 & -6 & -21 \end{array}\right)$$

aplicando à segunda linha a propriedade que diz $L_2 = L_2 - p.L_1$ e fazendo $p = 3$ para que surja um zero na primeira posição e a linha fica: $3, 3, 1, 27 - p.1, 4, 4, 23$. Com $p = 3$, a nova linha fica sendo $0, -9, -11, -42$ e a matriz agora é:

$$\left(\begin{array}{ccc|c} 1 & 4 & 4 & 23 \\ 0 & -9 & -11 & -42 \\ 0 & -3 & -6 & -21 \end{array}\right)$$

Finalmente, elimina-se o -3 na matriz acima de maneira a deixar a matriz escalonada: $L_3 = L_3 - p.L_2$ e fazendo $p = 0.333333$, a segunda linha fica sendo $0, 0, -2.33333, -7$ e a matriz escalonada agora é:

$$\left(\begin{array}{ccc|c} 1 & 4 & 4 & 23 \\ 0 & -9 & -11 & -42 \\ 0 & 0 & -2.33333 & -7 \end{array}\right)$$

Examinando a última linha, percebe-se que $z = \frac{-7}{-2.3333} = 3$. Levando este valor para a linha anterior, acha-se $y = 1$ e levando os dois valores na primeira linha, acha-se $x = 7$. Ou seja, o vetor $\{7, 1, 3\}$ é a solução procurada do sistema. O algoritmo em Python que resolve este sistema é

```
def gauss(a,b):
    n=len(b)
    x=[0]*n
    for passo in range(0,n-1):
        for i in range(passo+1,n):
            pivot=a[i,passo]/a[passo,passo]
            for j in range(0,n):
                a[i,j]=a[i,j]-(pivot*a[passo,j])
            b[i]=b[i]-pivot*b[passo]
    x[n-1]=b[n-1]/a[n-1,n-1]
    for m in range(n-1,-1,-1):
        x[m]=b[m]
        for j in range(m+1,n):
            x[m]=x[m]-a[m,j]*x[j]
        x[m]=x[m]/a[m,m]
    return(x)
import numpy as np
a=np.array([[3.0,-1,1],[1,2,-1],[4,-2,-1]],float)
b=np.array([11.0,6,-5],float)
x=gauss(a,b)
print(x))
```

A seguir o mesmo código em Octave

```
function r=gauss(a,b)
  n=length(b)
  x=zeros(1,n)
  for passo = 1:n
    for i = passo+1:n
      pivot=a(i,passo)/a(passo,passo)
      for j=1:n
```

```
        a(i,j)=a(i,j)-pivot*a(passo,j)
      endfor
      b(i)=b(i)-pivot*b(passo)
    endfor
  endfor
  x(n)=b(n)/a(n,n)
  for m=n:-1:1
    x(m)=b(m)
    for j=m+1:n
      x(m)=x(m)-a(m,j)*x(j)
    endfor
    x(m)=x(m)/a(m,m)
  endfor
  r=x
endfunction
```

O Python é espetacular Se seus dados forem complexos, o mesmo programa acima os resolve. Veja-se por exemplo

```
a=np.array([[1+2j,-3j,5],[2+3j,1+1j,1-1j],
     [4,2j,3-2j]])
b=np.array([10-16j,-5+12j,13+2j])
x=gauss(a,b)
```

Note que o número $1+j$ é escrito como `1+1j` e não como `1+j`. A execução dará como resposta:

```
===== RESTART: C:/p/ufpr/ci202/gauss1.py =====
[[1.+2.j -0.  -3.j   5.       +0.j        ]
 [0.+0.j  1.6 +5.8j -7.       +0.j        ]
 [0.+0.j  0.  +0.j   5.4198895+1.97790055j]]
---------------
[ 10. -16.j   -17.8+39.6j
  24.17127072-15.74585635j]
---------------
[(2.999999999999999+4j),
(2.0000000000000004+1.494200711594962e-15j),
(2.999999999999999-3.9999999999999996j)]
```

Sendo que a resposta procurada é 3+4j, 2 e 3-4j.

9.3 Pivoteamento Parcial

Como se viu um dos problemas do método de escalonamento de Gauss é a possível divisão por zero durante o cálculo do pivot. Para minimizar este problema pode-se usar o pivoteamento parcial ou total.

O pivoteamento parcial consiste em analisar o sistema e no início de cada passo de eliminação escolher o a_{kk} quem tiver o maior módulo entre os coeficientes que estão na coluna e abaixo de a_{kk}. Se necessário, as linhas i e k devem trocar de lugar. Isto não altera o sistema original. Veja-se um exemplo: Seja

o sistema

$$\begin{cases} -0.319x_1 + 0.884x_2 + 0.279x_3 &= 0 \\ 0.421x_1 + 0.784x_2 - 0.207x_3 &= 0 \\ 0.448x_1 + 0.832x_2 + 0.193x_3 &= 1 \end{cases}$$

ou na sua matriz expandida

$$A = \left[\begin{array}{ccc|c} -0.319 & 0.884 & 0.279 & 0 \\ 0.421 & 0.784 & -0.207 & 0 \\ 0.448 & 0.832 & 0.193 & 1 \end{array}\right]$$

Na primeira coluna, escolhe-se a linha de maior valor absoluto e troca-se de lugar esta linha com a linha 1 e fica

$$A = \left[\begin{array}{ccc|c} 0.448 & 0.832 & 0.193 & 1 \\ 0.421 & 0.784 & -0.207 & 0 \\ -0.319 & 0.884 & 0.279 & 0 \end{array}\right]$$

agora, é conduzida a eliminação gaussiana e fica

$$A = \left[\begin{array}{ccc|c} 0.448 & 0.832 & 0.193 & 1 \\ 0 & 0.002170 & -0.3884 & -0.9397 \\ 0 & 1.476 & 0.4164 & 0.7121 \end{array}\right]$$

O processo continua, agora na coluna 2. Fica de fora a linha 1. O maior valor absoluto da coluna 2, está na linha 3, e ela troca de lugar com a linha 2 e fica

$$A = \left[\begin{array}{ccc|c} 0.448 & 0.832 & 0.193 & 1 \\ 0 & 1.476 & 0.4164 & 0.7121 \\ 0 & 0.002170 & -0.3884 & -0.9397 \end{array}\right]$$

triangulando agora fica

$$A = \left[\begin{array}{ccc|c} 0.448 & 0.832 & 0.193 & 1 \\ 0 & 1.476 & 0.4164 & 0.7121 \\ 0 & 0 & -0.3893 & -0.9412 \end{array}\right]$$

Fazendo a retrosubstituição, acha-se a solução que é $S = \{1.561, -0.1997, 2.418\}$.

Com o processo de pivoteamento parcial, diminuí-se possíveis pivôs nulos e consegue-se uma redução nos efeitos cumulativos de erros de arredondamento pois os pivôs são os maiores possíveis em cada coluna gerando os menores fatores multiplicativos, com a ocorrência de arredondamento nos dígitos menos significativos.

9.4 Pivoteamento total

Neste processo, procura-se o elemento de maior módulo entre todos os elementos da matriz promovendo-se troca de linhas e de colunas, levando-se em conta que

- a troca de linhas significa apenas troca de lugar de equações sem maiores consequências e
- a troca de colunas significa trocar a ordem das incógnitas do sistema.

Por exemplo, resolvendo o mesmo sistema acima, fica Seja o sistema

$$\begin{cases} -0.319x_1 + 0.884x_2 + 0.279x_3 &= 0 \\ 0.421x_1 + 0.784x_2 - 0.207x_3 &= 0 \\ 0.448x_1 + 0.832x_2 + 0.193x_3 &= 1 \end{cases}$$

ou na sua matriz expandida

$$A = \left[\begin{array}{lll|l} -0.319 & 0.884 & 0.279 & 0 \\ 0.421 & 0.784 & -0.207 & 0 \\ 0.448 & 0.832 & 0.193 & 1 \\ x_1 & x_2 & x_3 & \end{array}\right]$$

Agora, busca-se o maior valor na matriz A e ele é encontrado na linha 3 e coluna 2 (0.884). Acrescenta-se uma linha à matriz, indicando a ordem de apresentação das incógnitas, que será importante quando as colunas forem trocadas.

Trocando-se linhas e colunas, fica

$$A = \left[\begin{array}{lll|l} 0.884 & -0.319 & 0.279 & 0 \\ 0.784 & 0.421 & -0.207 & 0 \\ 0.832 & 0.448 & 0.193 & 1 \\ x_2 & x_1 & x_3 & \end{array}\right]$$

triangulando, fica

$$A = \left[\begin{array}{lll|l} 0.884 & -0.319 & 0.279 & 0 \\ 0 & 0.7039 & -0.4544 & 0 \\ 0 & 0.7482 & -0.06959 & 1 \\ x_2 & x_1 & x_3 & \end{array}\right]$$

Agora, busca-se o maior elemento na matriz, a partir da segunda linha e da segunda coluna e ele é o valor 0.7482 (terceira linha, segunda coluna). Efetua-se apenas troca de linhas, uma vez que na coluna a matriz já está naturalmente pivotada. Fica

$$A = \left[\begin{array}{lll|l} 0.884 & -0.319 & 0.279 & 0 \\ 0 & 0.7482 & -0.06959 & 1 \\ 0 & 0.7039 & -0.4544 & 0 \\ x_2 & x_1 & x_3 & \end{array}\right]$$

e triangularizando, tem-se

$$A = \left[\begin{array}{lll|l} 0.884 & -0.319 & 0.279 & 0 \\ 0 & 0.7482 & -0.06959 & 1 \\ 0 & 0 & -0.3889 & -0.9408 \\ x_2 & x_1 & x_3 & \end{array}\right]$$

retrossubstituindo, fica (lembrando que a ordem das incógnitas foi alterada): $S = \{1.561, -0.2002, 2.419\}$. Vale a referência de poucos sistemas têm solução significativamente melhor com pivotação total em relação à solução obtida com pivotação parcial. Não foi este caso aqui resolvido, cuja solução total é equivalente à solução parcial.

9.5 Gauss-Jordan

Similar ao método de eliminação de Gauss, mas ele prossegue até que a matriz dos coeficientes seja transformada na matriz identidade. Não há o processo de retrossubstituição, uma vez que a leitura dos resultados $x_0, x_1, ..., x_n$ é imediata na última coluna da matriz expandida.

Vamos acompanhar o exemplo do sistema

$$\begin{cases} 2x_1 + 7x_2 + 8x_3 + 4x_4 &= 43 \\ 2x_1 + x_2 + 3x_3 + x_4 &= 16 \\ -1x_1 - 1x_2 + 2x_3 + 3x_4 &= 24 \\ 2x_1 + 2x_2 + x_3 + x_4 &= 9 \end{cases}$$

ou na sua matriz expandida

$$A = \left[\begin{array}{cccc:c} 2 & 7 & 8 & 4 & :43 \\ 2 & 1 & 3 & 1 & :16 \\ -1 & -1 & 2 & 3 & :24 \\ 2 & 2 & 1 & 1 & :9 \end{array}\right]$$

Zerando a primeira coluna e

```
[[  2.    7.    8.    4.   43. ]
 [  0.   -6.   -5.   -3.  -27. ]
 [  0.    2.5   6.    5.   45.5]
 [  0.   -5.   -7.   -3.  -34. ]]
```

e depois dividindo pelo elemento não nulo, fica

```
[[  1.    3.5   4.    2.   21.5]
 [  0.   -6.   -5.   -3.  -27. ]
 [  0.    2.5   6.    5.   45.5]
 [  0.   -5.   -7.   -3.  -34. ]]
```

A segunda coluna, fica

```
[[  1.          0.          1.0833333   0.25        5.75    ]
 [  0.         -6.         -5.         -3.        -27.      ]
 [  0.          0.          3.91666667  3.75       34.25    ]
 [  0.          0.         -2.8333333  -0.5       -11.5     ]]
```

e divididindo pelo elemento não nulo, fica

```
[[  1.          0.          1.0833333   0.25        5.75    ]
 [ -0.          1.          0.8333333   0.5         4.5     ]
 [  0.          0.          3.9166667   3.75       34.25    ]
 [  0.          0.         -2.8333333  -0.5       -11.5     ]]
```

A terceira coluna fica:

```
[[ 1.          0.          0.         -0.78723404 -3.72340426]
 [-0.          1.          0.         -0.29787234 -2.78723404]
 [ 0.          0.          3.9166667   3.75       34.25      ]
 [ 0.          0.          0.          2.21276596 13.27659574]]
```

e dividindo pelo elemento não nulo, fica

```
[[ 1.          0.          0.         -0.78723404 -3.72340426]
 [-0.          1.          0.         -0.29787234 -2.78723404]
 [ 0.          0.          1.          0.95744681  8.74468085]
 [ 0.          0.          0.          2.21276596 13.27659574]]
```

Finalmente a quarta coluna

```
[[ 1.          0.          0.          0.          1.        ]
 [ 0.          1.          0.          0.         -1.        ]
 [ 0.          0.          1.          0.          3.        ]
 [ 0.          0.          0.          2.21276596 13.27659574]]
```

e dividindo pelo não nulo

```
[[ 1.  0.  0.  0.  1.]
 [ 0.  1.  0.  0. -1.]
 [ 0.  0.  1.  0.  3.]
 [ 0.  0.  0.  1.  6.]]
```

e a resposta é imediata: $S = \{1, -1, 3, 6\}$. Eis o programa Python

```
# jordan
import numpy as np
def jordan(a):
  t=len(a)
  for i in range(t):
    for j in range(t):
      if i!=j:
        pivot=-a[j][i]/a[i][i]
        for k in range(t+1):
          a[j][k]=a[j][k]+a[i][k]*pivot
    alvo=a[i][i]
    #print(a)                    tire os ####
    #print('---')                se quiser ver a construcao
    for j in range(t+1):
      a[i][j]=a[i][j]/alvo
    #print(a)                    linha a
    #print('-----------------')  linha
  return a
a=np.array([[2.0,7,8,4,43],[2,1,3,1,16],
           [-1,-1,2,3,24],[2,2,1,1,9]])
print(jordan(a))
```

9.6 Fatoração LU

No método LU, deve-se transformar a matriz A (dos coeficientes) em um produto matricial $L.U$ onde a matriz U (de Upper) é a própria matriz A devidamente escalonada (como visto no método de Gauss) e a matriz L (de Lower) é uma matriz que contém unidades na diagonal principal, zeros acima destas posições e que contém os pivots do método de Gauss abaixo da diagonal principal.

1

[1] Qual o mais rápido: eliminação de Gauss ou LU ? Para um mesmo sistema de n equações e n incógnitas, o gasto de CPU em ciclos de máquina (4 ciclos para adição, subtração e multiplicação e 16 ciclos para

Em termos esquemáticos, se começa com o sistema $[A].\{x\} = \{b\}$. A primeira coisa a fazer é calcular L e U de modo que $[L].[U] = [A]$. Agora, fica $[L].[U].\{x\} = \{b\}$. Separando esta expressão, se faz $[U].\{x\} = \{y\}$ e finalmente $[L].\{y\} = \{b\}$. Ou seja, primeiro se calcula y em $[L].\{y\} = \{b\}$ e depois se calcula x em $[U].\{x\} = \{y\}$.

A principal vantagem deste método é separar o trabalho em 2 etapas: primeiro o cálculo de L e U e depois y e x. Para sistemas onde há que se calcular muitos x' em função de muitos b', mantendo-se constante A – o que é a imensa maioria dos casos, como se verá no exercício da tomografia computadorizada, este método adianta o "serviço" diminuindo o trabalho lá na frente.

9.6.1 Condições de existência

A decomposição LU pode ser feita desde que $det|A_n| \neq 0$ para $n = 1, 2, ..., k-1$ onde k é a ordem original de A. A_1 corresponde ao primeiro elemento de A. A_2 é matriz 2×2 do canto esquerdo alto de A.
Vejamos um exemplo, seja $A = \begin{pmatrix} 5 & 2 & 1 \\ 3 & 1 & 4 \\ 1 & 1 & 3 \end{pmatrix}$ agora, $A_1 = |5|$ e $A_2 = \begin{vmatrix} 5 & 2 \\ 3 & 1 \end{vmatrix}$
e o $det(A_1) = 5$ e o $det(A_2) = 5 \times 1 - 3 \times 2 = -1$ ambos diferentes de zero, portanto A pode ser decomposta em $LU = A$.

9.6.2 Processo de obtenção de LU

Tudo começa olhando-se o seguinte esquema:

$$[L].[U] = [A]$$

ou

$$\begin{pmatrix} 1 & 0 & 0 \\ ? & 1 & 0 \\ ? & ? & 1 \end{pmatrix} . \begin{pmatrix} ? & ? & ? \\ 0 & ? & ? \\ 0 & 0 & ? \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

Daqui, e usando-se a álgebra matricial básica sai que $u_{1j} = a_{1j}$ ou seja, que a primeira linha de U é igual à primeira linha de A.
A primeira coluna de L é $\ell_{i1} = \frac{a_{i1}}{u_{11}}$ para $(i = 2, 3...)$
A segunda linha de U é $u_{2j} = a_{2j} - \ell_{21} \times u_{1j}$
A segunda linha de L é $\frac{a_{i2} - \ell_{i1} \times u_{12}}{u_{22}}$ e assim por diante.

Chega-se à seguinte formulação:
L e U podem ser calculado assim

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} \ell_{ik} \times u_{kj}, \quad i \leq j$$

$$\ell_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} \ell_{ik} \times u_{kj} \right) / \; u_{jj}, \quad i > j$$

Seja por exemplo, resolver o sistema

$$\begin{cases} x & + & 4y & + & 4z & = & 23 \\ 3x & + & 3y & + & z & = & 27 \\ 2x & + & 5y & + & 2z & = & 25 \end{cases}$$

divisão) nos dois métodos é o mesmo e é de $qc = \left(\frac{8n^3}{3} + 12n^2 + \frac{4n}{3} \right)$. Mas claramente o método LU é melhor. Simplesmente porque divide o trabalho em 2 e permite reaproveitar para a solução de outras instâncias.

Aqui, as matrizes L e U são calculadas desta maneira:

$$\begin{pmatrix} 1 & 0 & 0 \\ ? & 1 & 0 \\ ? & ? & 1 \end{pmatrix} . \begin{pmatrix} ? & ? & ? \\ 0 & ? & ? \\ 0 & 0 & ? \end{pmatrix} = \begin{pmatrix} 1 & 4 & 4 \\ 3 & 3 & 1 \\ 2 & 5 & 2 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ ? & 1 & 0 \\ ? & ? & 1 \end{pmatrix} . \begin{pmatrix} \mathbf{1} & \mathbf{4} & \mathbf{4} \\ 0 & ? & ? \\ 0 & 0 & ? \end{pmatrix} = \begin{pmatrix} 1 & 4 & 4 \\ 3 & 3 & 1 \\ 2 & 5 & 2 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ \mathbf{3} & 1 & 0 \\ \mathbf{2} & ? & 1 \end{pmatrix} . \begin{pmatrix} 1 & 4 & 4 \\ 0 & ? & ? \\ 0 & 0 & ? \end{pmatrix} = \begin{pmatrix} 1 & 4 & 4 \\ 3 & 3 & 1 \\ 2 & 5 & 2 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 2 & ? & 1 \end{pmatrix} . \begin{pmatrix} 1 & 4 & 4 \\ 0 & \mathbf{-9} & \mathbf{-11} \\ 0 & 0 & ? \end{pmatrix} = \begin{pmatrix} 1 & 4 & 4 \\ 3 & 3 & 1 \\ 2 & 5 & 2 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 2 & \mathbf{0.33} & 1 \end{pmatrix} . \begin{pmatrix} 1 & 4 & 4 \\ 0 & -9 & -11 \\ 0 & 0 & ? \end{pmatrix} = \begin{pmatrix} 1 & 4 & 4 \\ 3 & 3 & 1 \\ 2 & 5 & 2 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 2 & 0.33 & 1 \end{pmatrix} . \begin{pmatrix} 1 & 4 & 4 \\ 0 & -9 & -11 \\ 0 & 0 & \mathbf{-2.33} \end{pmatrix} = \begin{pmatrix} 1 & 4 & 4 \\ 3 & 3 & 1 \\ 2 & 5 & 2 \end{pmatrix}$$

```
# calculo de L,U conforme [Fra07=Calculo Numerico,
#                          Neide B Franco] pag. 126
import numpy as np
def lu(a):
  t=len(a[0])
  u=np.zeros((t,t),float)
  l=np.zeros((t,t),float)
  for i in range(0,t):
    l[i,i]=1
  for i in range(0,t):
    for j in range(0,t):
      if i<=j:
        s=0
        for k in range(0,i):
          s=s+l[i,k]*u[k,j]
        u[i,j]=a[i,j]-s
      if i>j:
        s=0
        for k in range(0,j):
          s=s+l[i,k]*u[k,j]
        l[i,j]=(a[i,j]-s)/u[j,j]
  return(l,u)

l,u=lu(np.array([[1,4,4],[3,3,1],[2,5,2]]))
print(l)
print(u)
```

A seguir o mesmo cálculo em Octave

```
function [rl,ru]=lu(a)
  t=length(a)
  u=zeros(t)
  l=eye(t)
  for i=1:t
    for j=1:t
      s=0
      if i<=j
        for k=1:i
          s=s+l(i,k)*u(k,j)
        endfor
        u(i,j)=a(i,j)-s
      else
        for k=1:j
          s=s+l(i,k)*u(k,j)
        endfor
        l(i,j)=(a(i,j)-s)/u(j,j)
      endif
    endfor
  endfor
  rl=l
  ru=u
endfunction
```

O cálculo de L e U podem ser feito usando-se os pivots e os multiplicadores do método de Gauss, como se pode ver em

```
import numpy as np
def lua1(a,b):
  print('----------a-------')
  print(a)
  n=len(a)
  x=[0]*n
  y=[0]*n
  L=np.zeros((n,n),float)
  for passo in range(0,n):
    for i in range(passo+1,n):
      pivot=a[i,passo]/a[passo,passo]
      for j in range(0,n):
        a[i,j]=a[i,j]-pivot*a[passo,j]
      L[i,passo]=pivot
  for i in range(0,n):
    L[i,i]=1
  y[0]=b[0]
  for i in range(1,n):
    y[i]=b[i]
    for j in range(0,i):
      y[i]=y[i]-L[i,j]*y[j]
```

```
  x[n-1]=y[n-1]/a[n-1,n-1]
  for i in range(n-1,-1,-1):
    x[i]=y[i]
    for j in range(i+1,n):
      x[i]=x[i]-a[i,j]*x[j]
    x[i]=x[i]/a[i,i]
  print('----------L-------')
  print(L)
  print('----------y-------')
  print(y)
  print('----------u-------')
  print(a)
  print('----------x-------')
  print(x)
import numpy as np
a=np.array([[1,4,4],[3,3,1],[2,5,2]],float)
b=np.array([23,27,25],float)
lua1(a,b)
```

Como é de se esperar, ambos os métodos devem dar o mesmo resultado.

Uma implementação mais genérica do cálculo das matrizes L L e U pode ser

```
def lu(a):
  t=len(a[0])
  u=np.zeros((t,t),float)
  l=np.eye(t,dtype=float)
  for i in range(t):
    for j in range(t):
      s=0
      if i<=j:
        for k in range(i):
          s=s+l[i,k]*u[k,j]
        u[i,j]=a[i,j]-s
      else:
        for k in range(j):
          s=s+l[i,k]*u[k,j]
        l[i,j]=(a[i,j]-s)/u[j,j]
  return(l,u)
```

Lembrando que no pacote NUMPY está lá a multiplicação matricial. Basta fazer
$a1 = np.array([[...],[...],...], float)$ e $a2 = np.array([[...],[...],...], float)$ e depois $print(np.dot(a1, a2))$

9.6.3 Determinante eficiente

Vale a observação que calcular LU=A é uma das maneiras mais eficientes de obter o determinante de uma matriz, uma vez que

$$det(A) = det(L).det(U) = u_{11} \times u_{22} \times ...u_{kk}$$

Antes de encerrar este tema, o código de [Jus18] para o cálculo das matrizes L e U que vale pela sua concisão

```
import numpy as np
def fatoraLU(A):
  U=np.copy(A)
  n=np.shape(U)[0]
  L=np.eye(n)
  for j in np.arange(n-1):
    for i in np.arange(j+1,n):
      L[i,j]=U[i,j]/U[j,j]
      for k in np.arange(j+1,n):
        U[i,k]=U[i,k]-L[i,j]*U[j,k]
      U[i,j]=0
  return L,U
A=np.array([[1,4,4.0],[3,3,1],[2,5,2]])
print(fatoraLU(A))
```

Vale a referência de que o pacote Numpy do Python não tem a função LU (embora seja fácil calcular, como se viu acima) mas o pacote Scipy, subpacote linalg, tem a função `scipy.linalg.lu` que quando chamada devolve a matriz P de permutação e as matrizes L e U.

9.7 Usando a matriz inversa

Relembrando pode-se ter $|A| = |B|.|C|$ onde A é o resultado do produto interno de B por C. Algum mais desavisado pode lançar mão do conhecimento prévio de matemática e determinar que $|B| = \frac{|A|}{|C|}$. O mais próximo da divisão matricial que se pode chegar é a chamada matriz inversa, que poderia ser descrita como $A^{-1} = \frac{1}{A}$ e daqui sim que se pode escrever $A.A^{-1} = I$ onde I é a matriz identidade. A^{-1} é a matriz inversa de A.

A matriz inversa ajuda a resolver sistemas lineares de maneira bastante elegante. De fato, a maioria dos sistemas lineares reais têm uma matriz A mais ou menos fixa (já que ela reflete as interdependências das variáveis na obtenção do resultado numérico final) e uma matriz B que varia de caso a caso. Nos métodos anteriores, (Gauss, jacobi, LU...) este fato era desconsiderado e a cada nova instância do sistema tudo era recalculado, incluíndo-se aí a matriz A.

Agora, A gera a sua inversa A^{-1} uma única vez. E daí, a cada novo B apenas este é recalculado, usando-se o seguinte desenvolvimento:
$A.x = B$, multiplicando-se ambos os lados por A^{-1} fica $A^{-1}.A.x = A^{-1}.B$
e lembrando que $A.A^{-1} = I$, fica $x = A^{-1}.B$

O sistema

$$\begin{cases} a_{11}\ x_1 + a_{12}\ x_2 + ... + a_{1n}\ x_n & = & b_1 \\ a_{21}\ x_1 + a_{22}\ x_2 + ... + a_{2n}\ x_n & = & b_2 \\ ... & & \\ a_{m1}\ x_1 + a_{m2}\ x_2 + ... + a_{mn}\ x_n & = & b_m \end{cases}$$

é equivalente à seguinte equação matricial

$$\begin{pmatrix} a_{11}\ a_{12}\ ...\ a_{1n} \\ a_{21}\ a_{22}\ ...\ a_{2n} \\ ... \\ a_{m1}\ a_{m2}\ ...\ a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ . \\ . \\ . \\ b_m \end{pmatrix}$$

no sentido de que toda solução do primeiro é igualmente solução do segundo. Dessa maneira pode-se resumir qualquer dos sistemas acima na forma matricial escrevendo

$$Ax = B$$

Propriedades

1. só existe A^{-1} se $det(A) \neq 0$
2. A inversa da inversa é a própria matriz $(A^{-1})^{-1} = A$
3. $(n \cdot A)^{-1} = n^{-1} \cdot A^{-1}$
4. $(A_1 A_2 A_3 ... A_n)^{-1} = A_n^{-1} ... A_3^{-1} A_2^{-1} A_1^{-1}$
5. $(A^t)^{-1} = (A^{-1})^t$
6. $A.A^{-1} = A^{-1}.A = I$ (em geral $A.B \neq B.A$)
7. $I^{-1} = I$
8. $(A.B)^{-1} = B^{-1}.A^{-1}$

Formas de Calcular

Usando a teoria Usando a teoria, dada uma matriz como por exemplo $\begin{pmatrix} 3 & 5 \\ 2 & 3 \end{pmatrix}$

Uma possibilidade é querer encontrar escalares x, y, z e w para os quais $\begin{pmatrix} 3 & 5 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x & y \\ z & w \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ ou

$\begin{pmatrix} 3x+5z & 3y+5w \\ 2x+3z & 2y+3w \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

ou ainda equações que satisfaçam $\begin{cases} 3x+5z=1 \\ 2x+3z=0 \end{cases}$ e $\begin{cases} 3y+5w=0 \\ 2y+3w=1 \end{cases}$

Resolvendo estas equações, acha-se a matriz inversa que é $\begin{pmatrix} -3 & 5 \\ -2 & 3 \end{pmatrix}$.

Usando determinantes Outros métodos (extraídos do excelente site `http://mathworld.wolfram.com`) vão a seguir descritos: Dada uma matriz 2×2 representada por $A = \begin{vmatrix} a & b \\ c & d \end{vmatrix}$, a inversa é dada por

$A^{-1} = \frac{1}{|A|} \begin{vmatrix} d & -b \\ -c & a \end{vmatrix} = \frac{1}{ad-bc} \begin{vmatrix} d & -b \\ -c & a \end{vmatrix}$.

Para uma matriz 3×3 veja lá no site, que o espaço aqui é muito pequeno.

Usando Gauss-Jordan Neste método escreve-se a matriz da qual se quer achar a inversa ao lado esquerdo da matriz identidade, formando este arranjo (matriz escalonada reduzida por linhas).

$$[AI] = \left|\begin{array}{c|c} a_{11} \ ... \ a_{1n} & 1\ 0\ ...\ 0 \\ a_{21} \ ... \ a_{2n} & 0\ 1\ ...\ 0 \\ ... & ... \\ a_{n1} \ ... \ a_{nn} & 0\ 0\ ...\ 1 \end{array}\right|$$

Aplicam-se agora a este arranjo as técnicas de eliminação de Gauss, sobre o conjunto todo, de maneira a ficar com o seguinte arranjo $\left|\begin{array}{c|c} 1\ 0\ ...\ 0 & b_{11} \ ... \ b_{1n} \\ 0\ 1\ ...\ 0 & b_{21} \ ... \ b_{2n} \\ ... & ... \\ 0\ 0\ ...\ 1 & b_{n1} \ ... \ b_{nn} \end{array}\right|$

Neste ponto, a matriz (b_{ij}) é a matriz inversa de A. Ou seja $A^{-1} = B$ e também $AB = I$.

Usando cofatores Dada a matriz A, a sua inversa pode ser calculada pelo processo dos cofatores. Vejamos como.

$$A^{-1} = \frac{1}{det(A)}.\bar{A}$$

onde $\bar{A}$ é a matriz adjunta de A.

A matriz adjunta de A é a matriz transposta da matriz dos cofatores de A.

Para calcular os cofatores C_{ij} faça

$$C_{ij} = (-1)^{i+j}.det(X)$$

onde a matriz X é a resultante de A quando são eliminadas as linhas i e j da mesma.

Por exemplo seja $A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 0 & 5 \\ 2 & 1 & 1 \end{bmatrix}$

O determinante de $A = det(A) = 17$. Agora vão-se calcular os cofatores:

- Linha 1 coluna 1

  $A = \begin{bmatrix} \not{1} & \not{2} & \not{3} \\ \not{2} & 0 & 5 \\ \not{2} & 1 & 1 \end{bmatrix} = (-1)^{1+1} \times [(0 \times 1) - (1 \times 5)]$

  $= 1 \times -5 = -5$

- Linha 1 coluna 2

  $A = \begin{bmatrix} \not{1} & \not{2} & \not{3} \\ 2 & \not{0} & 5 \\ 2 & \not{1} & 1 \end{bmatrix} = (-1)^{1+2} \times [(2 \times 1) - (2 \times 5)]$

  $= -1 \times -8 = 8$

- Linha 1 coluna 3

  $A = \begin{bmatrix} \not{1} & \not{2} & \not{3} \\ 2 & 0 & \not{5} \\ 2 & 1 & \not{1} \end{bmatrix} = (-1)^{1+3} \times [(2 \times 1) - (2 \times 0)]$

  $= 1 \times 2 = 2$

- Linha 2 coluna 1
  $A = \begin{bmatrix} \not{1} & 2 & 3 \\ \not{2} & \not{0} & \not{5} \\ \not{2} & 1 & 1 \end{bmatrix} = (-1)^{2+1} \times [(2 \times 1) - (1 \times 3)]$
  $= -1 \times -1 = 1$
- Linha 2 coluna 2
  $A = \begin{bmatrix} 1 & \not{2} & 3 \\ \not{2} & \not{0} & \not{5} \\ 2 & \not{1} & 1 \end{bmatrix} = (-1)^{2+2} \times [(1 \times 1) - (2 \times 3)]$
  $= 1 \times -5 = -5$
- Linha 2 coluna 3
  $A = \begin{bmatrix} 1 & 2 & \not{3} \\ \not{2} & \not{0} & \not{5} \\ 2 & 1 & \not{1} \end{bmatrix} = (-1)^{2+3} \times [(1 \times 1) - (2 \times 2)]$
  $= -1 \times -3 = 3$
- Linha 3 coluna 1
  $A = \begin{bmatrix} \not{1} & 2 & 3 \\ \not{2} & 0 & 5 \\ \not{2} & \not{1} & \not{1} \end{bmatrix} = (-1)^{3+1} \times [(2 \times 5) - (0 \times 3)]$
  $= 1 \times 10 = 10$
- Linha 3 coluna 2
  $A = \begin{bmatrix} 1 & \not{2} & 3 \\ 2 & \not{0} & 5 \\ \not{2} & \not{1} & \not{1} \end{bmatrix} = (-1)^{3+2} \times [(1 \times 5) - (2 \times 3)]$
  $= -1 \times -1 = 1$
- Linha 3 coluna 3
  $A = \begin{bmatrix} 1 & 2 & \not{3} \\ 2 & 0 & \not{5} \\ \not{2} & \not{1} & \not{1} \end{bmatrix} = (-1)^{3+3} \times [(1 \times 0) - (2 \times 2)] = 1 \times -4 = -4$

A matriz dos cofatores é: $A = \begin{bmatrix} -5 & 8 & 2 \\ 1 & -5 & 3 \\ 10 & 1 & -4 \end{bmatrix}$

e a sua transposta: $A = \begin{bmatrix} -5 & 1 & 10 \\ 8 & -5 & 1 \\ 2 & 3 & -4 \end{bmatrix}$ que dividida por 17 dá

```
array([[-0.29411765,  0.05882353,  0.58823529],
       [ 0.47058824, -0.29411765,  0.05882353],
       [ 0.11764706,  0.17647059, -0.23529412]])

Para ter certeza:
>>> a
array([[-0.29411765,  0.05882353,  0.58823529],
       [ 0.47058824, -0.29411765,  0.05882353],
       [ 0.11764706,  0.17647059, -0.23529412]])
>>> b=np.array([[1,2,3],[2,0,5],[2,1,1]])
```

```
>>> b
array([[1, 2, 3],
       [2, 0, 5],
       [2, 1, 1]])
>>> print(np.dot(a,b))
[[ 9.99999990e-01 -1.00000001e-08 -1.00000001e-08]
 [-2.77555756e-17  1.00000001e+00 -1.80411242e-16]
 [ 0.00000000e+00  0.00000000e+00  1.00000001e+00]]
>>>
```

Não parece muito, mas se você olhar com cuidado verá que é a matriz identidade em sua modalidade *float*. Logo, cqd ou *qed*.

Usando LU Relembrando a decomposição LU, para um exemplo já feito em exercício passado:

$$A = L.U = \begin{pmatrix} 3 & 4 & -9 \\ 2 & 1 & 2 \\ 7 & -1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0.666 & 1 & 0 \\ 2.333 & 6.2 & 1 \end{pmatrix} . \begin{pmatrix} 3 & 4 & -9 \\ 0 & -1.666 & 8 \\ 0 & 0 & -27.6 \end{pmatrix}$$

A inversa pode ser calculada coluna a coluna, fazendo-se a multiplicação de L por um vetor coluna $\{d1, d2, d3\}$ obtendo como resultado a matriz coluna $\{1, 0, 0\}$. Depois, seguindo-se o método, multiplica-se a incógnita por U obtendo este resultado (Lembre-se do método LU). O resultado final deste sistema será a primeira coluna da matriz inversa procurada. Acompanhe: $L.\{d1, d2, d3\} = \{1, -0.6666, 1.79999\}$ Agora, deve-se calcular $U.x = d$ o que resulta $\{0.02173913, 0.08695652, -0.06521739\}$ que é a primeira coluna da matriz inversa procurada.

A segunda coluna é obtida fazendo-se $L.d = \{0, 1, 0\}$ e fica $d = \{0, 1, -6.2\}$ Agora $U.x = d$ e fica $x = \{0.03620637, 0.47828000, 0.22463768\}$ que é a segunda coluna da matriz inversa. Finalmente, a terceira coluna $L.d = \{0, 0, 1\}$ e fica $d = \{0.12319768, -0.17392000, -0.03623188\}$ e a matriz inversa é

```
array([[ 0.02173913,  0.03623188,  0.12318841],
       [ 0.08695652,  0.47826087, -0.17391304],
       [-0.06521739,  0.22463768, -0.03623188]])
```

o que pode ser confirmado multiplicando-se esta matriz por A o que resultará na matriz identidade.

Usando pacotes computacionais

APL Dada a matriz A, sua inversa é facilmente calculável. Acompanhe

```
      A←3 3⍴3 4 ¯9 2 1 2 7 ¯1 1
      ⌹A
 0.02173913043 0.03623188406  0.1231884058
 0.08695652174 0.4782608696  ¯0.1739130435
¯0.0652173913  0.2246376812  ¯0.03623188406
```

MATLAB A função é `Y=inv(X)`. Acompanhe

```
X=[1 0 2; -1 5 0; 0 3 -9]
```

```
Y=inv(X)
Y
    0.8824   -0.1176    0.1961
    0.1765    0.1765    0.0392
    0.0588    0.0588   -0.0980
```

Python-Numpy O pacote Numpy é o de computação numérica no Python.

```
>>> import numpy as np
>>> A=np.array([[1,0,2],[-1,5,0],[0,3,-9]])
>>> A
array([[ 1,  0,  2],
       [-1,  5,  0],
       [ 0,  3, -9]])
>>> B=np.linalg.inv(A)
>>> B
array([[ 0.88235294, -0.11764706,  0.19607843],
       [ 0.17647059,  0.17647059,  0.03921569],
       [ 0.05882353,  0.05882353, -0.09803922]])
```

Python-Sympy O pacote Sympy é o de computação simbólica.

```
>>> import sympy as sp
>>> a=sp.Matrix([[1,0,2],[-1,5,0],[0,3,-9]])
>>> a
Matrix([
[ 1, 0,  2],
[-1, 5,  0],
[ 0, 3, -9]])
>>> a.inv()
Matrix([
[15/17, -2/17, 10/51],
[ 3/17,  3/17,  2/51],
[ 1/17,  1/17, -5/51]])
>>> a.inv().n()
Matrix([
[ 0.882352941176471, -0.117647058823529,   0.196078431372549],
[ 0.176470588235294,  0.176470588235294,  0.0392156862745098],
[0.0588235294117647, 0.0588235294117647, -0.0980392156862745]])
```

Matrizes Inversíveis

Diz-se que uma matriz quadrada é inversível se existe uma matriz B com a propriedade de que $AB = BA = I$ onde I é a matriz identidade.

A matriz B inversa de A é única se existir. A matriz inversa de A é indicada por A^{-1}. Note-se que a relação acima é simétrica, pois se B é a inversa de A então A também é inversa de B.

Para que serve a matriz inversa ?

Inúmeras aplicações, mas vamos focar aqui na solução de sistemas lineares. Dado o sistema $Ax = B$, a matriz A é a dos coeficientes. Se você calcular

a inversa A^{-1}, poderá resolver inúmeras instâncias do mesmo sistema, sem outra preocupação, já que $A^{-1} \times B = x$. Vamos ver um exemplo do que se diz aqui. Seja um sistema, dado por

$$\left\{ \begin{array}{lll} 3x + 2y + z & = & \\ x - y + z & = & \\ 5x + z & = & \end{array} \right\}$$

Escrito na forma matricial, o sistema fica $Ax = B$, onde A é a matriz

$$\begin{pmatrix} 3 & 2 & 1 \\ 1 & -1 & 1 \\ 5 & 0 & 1 \end{pmatrix}$$

Calculando a matriz inversa desta, obtém-se: $\begin{pmatrix} -0.1 & -0.2 & 0.3 \\ 0.4 & -0.2 & -0.2 \\ 0.5 & 1 & -0.5 \end{pmatrix}$

Agora, a solução para a instância onde $B = \{10, 2, 8\}$ é obtida fazendo-se $A^{-1} \times B$ que dá os valores $x = 1, y = 2, z = 3$. Para outra instância cujo B seja $\{30, -1, 26\}$, ao fazer a mesma multiplicação, obtém-se $x = 5, y = 7, z = 1$ e assim por diante.

Um processo de cálculo

Este exemplo pede que você implemente o algoritmo de Gauss-Jordan para cálculo da matriz inversa e depois o aplique a uma matriz de ordem suficientemente alta para desencorajar o cálculo manual da mesma. Relembrando as etapas, você deve:

1. Escrever a matriz A e ao seu lado a matriz I.

2. Aplicar operações elementares com linhas a A visando transformá-la em I.

3. Aplicar rigorosamente as mesmas operações sobre a matriz ao lado.

4. Quando A tiver se transformado em I, a matriz que era originalmente I se transformou em A^{-1}.

5. Para conferir, faça a multiplicação matricial entre A e A^{-1}. Tem que dar I.

Vamos aplicar este processo ao sistema acima descrito. O objetivo é produzir um zero no coeficiente do x na segunda equação: Para isso:
$a[1][] = a[1][] + a[0] \times (-a[1][0]/a[0][0])$ onde a notação $a[1][]$ deve ser entendida como a aplicação da operaçao à toda a linha [1] da matriz a. Mas, atenção: isto não vale no Python. Lá, tem que ser por extenso.

Note-se também que o fator $(-a[1][0]/a[0][0])$ é o mesmo para toda a linha e portanto deve ser calculado antes e preservado durante todo o ciclo. O algoritmo vai chamar este fator de *pivot* e ele vai ser previamente calculado e guardado.

para calcular o zero no coeficiente de x na terceira linha, o comando é $a[2][] = a[2][] + a[0] \times (-a[2][0]/a[0][0])$.

Para a segunda coluna, começa-se achando o zero na primeira linha. O

comando agora é
$a[0][] = a[0][] + a[1][] \times (-a[0][1]/a[1][1])$ e depois
$a[2][] = a[2][] + a[1][] \times (-a[2][1]/a[1][1])$. A terceira coluna é
$a[0][] = a[0][] + a[2][] \times (-a[0][2]/a[2][2])$ e
$a[1][] = a[1][] + a[2][] \times (-a[1][2]/a[2][2])$.
Com estas operações, os zeros foram alcançados. Agora basta transformar os elementos da diagonal principal na unidade. As operações são:
$a[0][] = a[0][]/a[0][0]$, $a[1][] = a[1][]/a[1][1]$ e $a[2][] = a[2][]/a[2][2]$.

Note que não houve preocupação em determinar se a matriz A é inversível. Isto foi deixado para não complicar demais o algoritmo. Fica como desafio para os mais corajosos.

Eis o algoritmo:

```
import numpy as np
def cami(x):
  # x e a matriz de ordem ta de quem
  #      se quer obter a inversa
  ta=len(x[0])
  y=np.identity(ta,float) #tem 0, na dp tem 1
  i=0
  while(i<ta):
    j=0
    while(j<ta):
      if (j!=i):
        pivot=-x[j][i]/x[i][i]
        k=0
        while k<ta:
          x[j][k]=x[j][k]+x[i][k]*pivot
          y[j][k]=y[j][k]+y[i][k]*pivot
          k=k+1
      j=j+1
    alvo=x[i][i]
    k=0
    while k<ta:
      y[i][k]=y[i][k]/alvo
      x[i][k]=x[i][k]/alvo
      k=k+1
    i=i+1
  return y;
a=np.array([[2.1, 3.2],[4.3,5.4]],float)
b=cami(a)
print(b)
```

Um exemplo Para você testar seu algoritmo, use-o nesta matriz:

```
 59 57 44 42 65 23 47 66 78 71
 94 36 83 17 12 86  4 92 43 95
 12 61 18 10 94 45 94 81 30 76
  9 49 92 22 90 22 27  6 21 26
 31 45 96 26 26 75 59 39 47 98
 33 27  9 94 29  5 49 68  9 33
 61 81 17 18 61 74 16 37 34 45
```

```
 64 85 46 74 49 81 69 20 18 59
 17 54 66 65 94 54 76 54 30 76
 68 91 61 89  6 57 35 92 52 80
```

Aqui a resposta deverá ser

```
 .01666  .02144  .00355 -.00554 -.04328 -.03519 -.03828  .03334  .01739  .00614
 .01345  .01914  .00951 -.01430 -.08069 -.08932 -.06358  .04098  .05929  .04586
-.01299 -.01160  .00040  .02405  .04808  .05005  .03034 -.02125 -.04923 -.01706
-.00062 -.00404 -.01100 -.00480  .00165  .00610  .00527 -.00364  .01265  .00052
 .00908  .01192 -.00185 -.00722 -.03673 -.03242 -.02033  .01253  .03801  .00633
-.04554 -.05485 -.01058  .03782  .17440  .17987  .15199 -.08849 -.13871 -.07298
-.01941 -.03323  .00783  .02992  .10479  .11100  .07145 -.03744 -.10593 -.04658
-.02700 -.02597  .00479  .02832  .09002  .10399  .07709 -.05417 -.08656 -.03329
-.02273 -.05918 -.01267  .03639  .16686  .16698  .13858 -.08701 -.13690 -.06634
 .06722  .09440  .00508 -.08773 -.28924 -.31585 -.24089  .14264  .27209  .12086
```

Em um sistema linear, informa as respostas a determinados estímulos, ou seja funciona para entender as inter-relações de partes componentes. Sempre que um sistema linear for derivado de uma lei de conservação (massa, força, calor, momento, carga, quantidade de bits, ...). Neste caso, uma única equação pode ser escrita em cada ponto do sistema, resultando em um conjunto de equações que que define as propriedades do sistema. Em $A.x = B$, x são os níveis da propriedade sendo balanceada em cada parte do sistema. B contém os elementos do balanço que são independentes do comportamento do sistema – são constantes. Em muitos casos, são os estímulos externos. A matriz A contém os parâmetros que expressam como as partes do sistema interagem ou são acopladas. Como se viu, a solução de $A.x = B$ pode ser escrita como $x = A^{-1}.B$. Da multiplicação de matrizes, pode-se dizer: $x_1 = a_{11}^{-1}b_1 + a_{12}^{-1}b_2 + a_{13}^{-1}b_3$ (com formulações semelhantes para os demais x_i). Daqui vê-se que a matriz inversa, além de fornecer uma solução, mostra a resposta de uma parte do sistema a um estímulo unitário de qualquer outra parte do sistema. A conclusão é que a_{ij}^{-1} representa o valor de x_i causado por uma quantidade unitária de b_j.

9.8 Gauss-Jacobi

Este método é iterativo, ou seja, não exato, já que parte-se de um valor atribuído inicial e mediante buscas locais sucessivas, o mesmo converge para a resposta correta. A diferença entre o resultado sugerido e aquele real, é denominada erro, e pode ser tornada tão pequena quanto se queira. O método entretanto tem um senão, que é um critério de convergência bastante rígido. Se não obedecido o algoritmo não converge para a resposta correta, e portanto o método é inútil.

Considere $A.x = b$ um sistema de equações lineares de ordem n e cujo $det(A) \neq 0$. A maneira extensa de escrever tal sistema é

$$\begin{array}{rcrcrcl} a_{11}x_1 & + & a_{12}x_2 & +\ldots+ & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & +\ldots+ & a_{2n}x_n & = & b_2 \\ \ldots & & & & & & \\ a_{n1}x_1 & + & a_{n2}x_2 & +\ldots+ & a_{nn}x_n & = & b_n \end{array}$$

A matriz A dos coeficientes, pode ser decomposta em $A = L + D + R$, onde

L é a matriz inferior de A (o resto é zero)

D é a matriz contendo apenas a diagonal principal de A, o resto é zero.

R é a matriz superior de A (o resto é zero)

Veja por exemplo, se

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \therefore L = \begin{pmatrix} 0 & 0 & 0 \\ 4 & 0 & 0 \\ 7 & 8 & 0 \end{pmatrix},$$

e

$$D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 \end{pmatrix}, R = \begin{pmatrix} 0 & 2 & 3 \\ 0 & 0 & 6 \\ 0 & 0 & 0 \end{pmatrix}$$

Supondo $det(D) \neq 0$, pode-se escrever: $(L + D + R)x = b$ e daqui $Dx = -(L + R)x + b$. e $x = -D^{-1}(L + R)x + D^{-1}b$. Chamando $B = -D^{-1}(L + R)$ e $g = D^{-1}.b$ obtem-se a equação geral deste método que é

$$x = Bx + g$$

. Note que é uma equação recursiva (ou iterativa) já que o x aparece nos 2 lados da mesma. A idéia aqui é atribuir um valor a x e com ele calcular um novo x, que é jogado no lugar do x velho e gera um novo x e assim sucessivamente até haver a convergência esperada (de antemão). Como supusemos $det(D) \neq 0$ pode-se dividir cada equação do sistema pelo coeficiente da diagonal principal, resultando

$$A^* = L^* + I + R^*$$

Agora o processo iterativo pode ser definido como

$$x^{k+1} = -(L^* + R^*)x^k + b^*$$

onde os elementos de L^*, R^* e b^* são:

$l_ij^* = \frac{a_{ij}}{a_{ii}}$ se $i > j$ e zero senão

$d_{ij} = a_{ij}$ se $i = j$ e zero senão

$r_ij^* = \frac{a_{ij}}{a_{ii}}$ se $i < j$ e zero senão

$b_ij^* = \frac{b_i}{a_{ii}}$

Critérios de convergência

Depois de dividida a matriz pelo elemento da diagonal principal, deve-se ter o **critério das linhas** que diz

$$max_{1 \leq i \leq n} \sum_{j=1, j \neq i}^{n} |a_{ij}^*| < 1$$

OU então o **critério das colunas** que diz

$$max_{1 \leq j \leq n} \sum_{i=1, i \neq j}^{n} |a_{ij}^*| < 1$$

Note que qualquer um dos dois critérios sendo satisfeitos, isso garante que o sistema tem convergência.

Definindo uma matriz como estritamente diagonalmente dominante se

$$\sum_{j=1,j\neq i}^{n} |a_{ij}| < |a_{ii}|$$

e se A é estritamente diagonalmente dominante, então A satisfaz o critério das linhas, e o sistema converge. Então, um bom critério de convergência pode ser

$$max_{1\leq i\leq n} \sum_{j=1,j\neq i}^{n} |a_{ij}^{|}* < 1$$

Note que eventualmente, uma equação pode mudar de lugar, sem que o sistema se altere. Este fato pode e deve ser considerado na análise de convergência.

Exemplo

Seja resolver o sistema abaixo, usando o método de Jacobi

$$\begin{array}{rcrcrcr} 10x & + & 2y & + & z & = & 7 \\ x & + & 5y & + & z & = & -8 \\ 2x & + & 3y & + & 10z & = & 6 \end{array}$$

com $x^0 = (0.7, -1.6\ e\ 0.6)^t$ e $\epsilon < 0.11$.

Convergência ?

$|a_{12}| + |a_{13}| < |a_{11}|$? ou $|2| + |1| < |10|$? R: sim
$|a_{21}| + |a_{23}| < |a_{22}|$? ou $|1| + |1| < |5|$? R: sim
$|a_{31}| + |a_{32}| < |a_{33}|$? ou $|2| + |3| < |10|$? R: sim
logo, o sistema convergirá!

Solução

Dividindo cada equação pelo elemento da diagonal principal, fica

$$\begin{array}{rcrcrcr} x & + & 0.2y & + & 0.1z & = & 0.7 \\ 0.2x & + & y & + & 0.2z & = & -1.6 \\ 0.2x & + & 0.3y & + & z & = & 0.6 \end{array}$$

Apenas para ilustrar o critério das linhas para testar convergência (desnecessário no caso, já que já vimos que o sistema converge, mas em tese...)
$|a_{12}^*| + |a_{13}^*| = |0.2| + |0.1| = 0.3$ e
$|a_{21}^*| + |a_{23}^*| = |0.2| + |0.2| = 0.4$ e
$|a_{31}^*| + |a_{32}^*| = |0.2| + |0.3| = 0.5$ e
$\Rightarrow\ max_{1\leq i\leq n}(0.3, 0.4\ 0.5) = 0.5 < 1$

Iterações

$x^{k+1} = -0.2y^k - 0.1z^k + 0.7$
$y^{k+1} = -0.2x^k - 0.2z^k - 1.6$
$z^{k+1} = -0.2x^k - 0.3y^k + 0.6$
Começando com $(0.7,\ -1.6,\ 0.6)$ obtemos uma primeira aproximação que é

(0.96, −1.86, 0.94) e jogando estes novos valores na fórmula iterativa, obtem-se:

$x \rightarrow 0.7,\ 0.96,\ 0.978,\ 0.9994,\ 0.9979$
$y \rightarrow -1.6,\ -1.86,\ -1.98,\ -1.9888,\ -1.9996$
$z \rightarrow 0.6,\ 0.94,\ 0.966,\ 0.9984,\ 0.9968$

O critério de parada pode ser
$max(x^{k+1} - x^k) < \epsilon$. No exemplo acima $0.0108 < 0.011$. Ou seja, neste caso, a solução procurada é $x = 0.9979$, $y = -1.9996$ e $z = 0.9978$. Salta aos olhos neste caso que se a tolerância for um pouco menor, a resposta exata será encontrada que é $(1, -2, 1)$. Basta lançar este resultado no sistema original que a exatidão será confirmada.

Solução Python

```
#jacobi
def jacobi(a,b):
 import numpy
 n=len(a) #numero de linhas de a
          # len(a[0])=colunas de a
 xvelho=numpy.zeros((n),float)
 erro=numpy.zeros((n),float)
 xnovo=numpy.zeros((n),float)
 f=numpy.zeros((n,len(a[0])),float)
 d=numpy.zeros((len(a[0])),float)
 tol=0.1
 for i in range(0,n):
  for j in range(0,n):
   if i==j:
    f[i,j]=0
   else:
    f[i,j]=-a[i,j]/a[i,i]
 for i in range(0,n):
  d[i]=b[i]/a[i,i]
 erromax=tol+1
 while erromax>tol:
  for i in range(0,n):
   xnovo[i]=0
   for k in range(0,n):
    xnovo[i]=xnovo[i]+f[i,k]*xvelho[k]
  for i in range(0,n):
   xnovo[i]=xnovo[i]+d[i]
  for i in range(0,n):
   erro[i]=abs(xnovo[i]-xvelho[i])
  erromax=erro[0]
  for i in range(1,n):
   if erro[i]>erromax:
    erromax=erro[i]
  for i in range(0,n):
   xvelho[i]=xnovo[i]
 return(xnovo)
```

```
import numpy
# para usar faça:
# a=numpy.array([[10,7,3],[1,5,4],[3,5,9]],float)
# b=numpy.array([22,33,44],float)
# print(jacobi(a,b))
```

9.9 Gauss-Seidl

Derivado do método de Gauss-Jacobi (portanto compartilhando com ele os critérios de convergência, a saber: o valor da diagonal principal de A deve ser igual ou maior do que a soma dos demais valores) este método acelera a convergência ao usar a cada interação os valores já calculados na própria interação, ou seja

$$x_1^{(k)} = \frac{1}{a_{11}}(b_1 - a_{12}x_2^{(k-1)} - a_{13}x_3^{(k-1)} - ... - a_{1n}x_n^{(k-1)})$$

$$x_2^{(k)} = \frac{1}{a_{22}}(b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k-1)} - ... - a_{2n}x_n^{(k-1)})$$

$$x_3^{(k)} = \frac{1}{a_{33}}(b_3 - a_{31}x_1^{(k)} - a_{32}x_2^{(k)} - ... - a_{3n}x_n^{(k-1)})$$

...

$$x_n^{(k)} = \frac{1}{a_{nn}}(b_n - a_{n1}x_1^{(k)} - a_{n2}x_2^{(k)} - ... - a_{n(n-1)}x_{(n-1)}^{(k)})$$

A condição de parada pode ser se $\forall|x_n^{(j)} - x_n^{(j-1)}| \leq \ erro$ então $x_n^{(j)}$ são as soluções do sistema. Usando a abordagem matricial pode-se escrever $Ax = B$ e sendo A é a matriz dos coeficientes e $A = D + I + S$, onde D é a matriz diagonal, I a matriz inferior e S a matriz superior e pode-se reescrever
$DX^{(k)} = B - IX^{(k)} - SX^{(k-1)}$ e
$(D + I)X^{(k)} = B - SX^{(k-1)}$
Multiplicando ambos os lados da equação pela inversa de $(D + I)$ fica $(D + I)^{-1}(D + I)X^{(k)} = (D + I)^{-1}B - (D + I)^{-1}SX^{(k-1)}$ e
$X^{(k)} = -(D + I)^{-1}SX^{(k-1)} + (D + I)^{-1}B$ e
$X^{(k)} = GX^{(k-1)} + F$
onde $G = -(D + I)^{-1}S$ e $F = (D + I)^{-1}B$

9.9.1 Critério de convergência

Continua valendo o critério das linhas[2], visto no método de Gauss-Jacobi.

Além deste, pode-se usar o critério de Sassenfeld Diz-se que o método de Gauss-seidl converge se

$$max_{q \leq i \leq n} \ \beta_i \ < \ 1$$

[2]aquele que manda que o elemento da diagonal principal seja maior que a soma dos demais em módulo

onde os $\beta_i$são calculados por recorrência:

$$\beta_i = \sum_{j=1}^{i-1} |a_{ij}^*|\beta_j + \sum_{j=i+1}^{n} |a_{ij}^*|$$

Acompanhe no exemplo: Seja o sistema

$$\begin{cases} 5x_1 + x_2 + x_3 = 5 \\ 3x_1 + 4x_2 + x_3 = 6 \\ 3x_1 + 3x_2 + 6x_3 = 0 \end{cases}$$

Aliás, antes de continuar, perceba que ele não atende o critério das linhas. Então, olha-se o critério de Sassenfeld

$$\begin{pmatrix} 5 & 1 & 1 & 5 \\ 3 & 4 & 1 & 6 \\ 3 & 3 & 6 & 0 \end{pmatrix}$$

dividindo cada linha pelo elemento da diagonal principal fica

$$\begin{pmatrix} 1 & 0.2 & 0.2 & 1 \\ 0.75 & 1 & 0.25 & 1.5 \\ 0.5 & 0.5 & 1 & 0 \end{pmatrix}$$

Daqui

$$\beta_1 = |0.2| + |0.2| = 0.4$$

$$\beta_2 = |0.75| \times 0.4 + 0.25 = 0.3 + 0.25 = 0.55$$

$$\beta_3 = |0.5| \times 0.4 + 0.5 \times 0.55 = 0.2 + 0.275 = 0.475$$

e daqui $\max(0.4, 0.55, 0.475) < 1$ ou $0.55 < 1$ e o critério de Sassenfeld está atendido. Neste exemplo cores aleatórias foram usadas para indicar de onde vem cada um dos valores aplicados na fórmula.

Mais um exemplo do critério de Sassenfeld, este estraído de [Rug88, pág. 174] Seja o sistema linear

$$\begin{cases} x_1 + 0.5x_2 - 0.1x_3 + 0.1x_4 = 0.2 \\ 0.2x_1 + x_2 - 0.2x_3 - 0.1x_4 = -2.6 \\ -0.1x_1 - 0.2x_2 + x_3 + 0.2x_4 = 1.0 \\ 0.1x_1 + 0.3x_2 + 0.2x_3 + x_4 = -2.5 \end{cases}$$

Temos
$\beta_1 = [0.5 + 0.1 + 0.1]/1 = 0.7$
$\beta_2 = [(0.2)(0.7) + 0.2 + 0.1]/1 = 0.44$
$\beta_3 = [(0.1)(0.7) + (0.2)(0.44) + 0.2]/1 = 0.358$
$\beta_4 = [(0.1)(0.7) + (0.3)(0.44) + (0.2)(0.358)]/1 = 0.2736$
E como $\beta_i < 1$ o sistema converge. Vale lembrar que o critério de Sassenfeld é suficiente, mas não é necessário.

Daqui, a solução programada em Python

```
def gaus_seid(x,b):
  ta=len(x[0])
  b=b.reshape((ta,1))
  D=np.zeros((ta,ta))
  S=np.zeros((ta,ta))
  I=np.zeros((ta,ta))
  F=np.zeros((ta,ta))
  G=np.zeros((ta,ta))
  for i in range(ta):
    for j in range(ta):
      if i<j:
        S[i][j]=x[i][j]
      elif i==j:
        D[i][j]=x[i][j]
      elif i>j:
        I[i][j]=x[i][j]
  F=D+I
  F=np.linalg.inv(F)   # OU cami(F)
  F=np.dot(F,b)  #   OU mm(F,b)
  F=np.around(F,decimals=6) # ser houver requisições
  G=-(D+I)
  G=np.linalg.inv(G) #  OU cami(G)
  G=np.dot(G,S)  #    OU mm(G,S)
  XN=np.zeros((ta,1),float)
  XV=np.ones((ta,1),float)
  while (0.00002<max(abs(XN-XV))):  # conforme requerido
    print('-----erro: ',max(abs(XN-XV)))
    XV=XN
    XN=mm(G,XV)+F
    print(XN)
a=np.array([[10.0,2,1],[1,5,1],[2,3,10]])
b=np.array([[7.0],[-8],[6]])
a=np.array([[42,9,21,4,3.0],[11,71,16,22,17],
   [19,24,89,10,31],[20,35,23,91,5],
   [12,30,25,14,82]])
b=np.array([[655.0],[1284],[1275],
   [1812],[1929]])
gaus_seid(a,b)
```

Por óbvio, as funções de matriz inversa e de multiplicação matricial podem ser as originais do pacote *numpy* ou ao contrário podem ser diretamente programadas conforme a matemática (quase) básica.

9.10 Relaxação

Este método atribui um valor inicial às variáveis e em seguida calcula o sistema, obtendo o que é chamado de resíduos. Agora, o maior resíduo é isolado (sinalizando qual das equações é responsável por ele) e dentro da equação, qual a variável de maior coeficiente. Esta variável passa a ser considerada a única responsável pelo erro. Portanto é recalculada e todo o ciclo recomeça.

O critério de parada pode ser quando todos os resíduos sejam menores que

um valor previamente estipulado.

No código Python em anexo, o valor inicial das varíaveis é sempre zero.

```
import numpy as np
def relax(a,b):
  xv=np.zeros(len(a),float)    # inicialização
  quem = 1.0
  while 1==1:
    re=np.zeros(len(a),float)
    for j in range(len(a)):
      for k in range(len(a)):
        re[j]=re[j]+a[j][k]*xv[k]
      re[j]=re[j]-b[j]
    re=list(re)
    quem=max(np.abs(re))
    if quem<0.00001:  # conforme requerido
      return
    for k in range(len(re)):
      if quem==abs(re[k]):
        ind=k
    novo=0
    for k in range(len(a)):
      if (k!=ind):
        novo=novo-a[ind][k]*xv[k]
    novo=novo+b[ind]
    xv[ind]=novo/a[ind][ind]
    print('novo xv=',xv)

a=np.array([[10.0,2,1],[1,-15,1],[2,3,10]])
b=np.array([7.0,32,6])
a=np.array([[42,9,21,4,3.0],[11,71,16,22,17],
            [19,24,89,10,31],
            [20,35,23,91,5],[12,30,25,14,82]])
b=np.array([655.0,1284,1275,1812,1929])
relax(a,b)
```

9.10.1 Comparação entre os métodos

Quanto à convergência, os métodos diretos por serem processos finitos sempre têm a convergência garantida. Já os métodos iterativos exigem algumas condições para a convergência.

Um aspecto importante é a esparsidade de A. Uma matriz é esparsa, quando formada em grande medida por zeros. É o caso de muitos problemas físicos (discretização de equações diferenciais, método dos elementos finitos, redes de potência, etc). Quando se aplicam os métodos diretos, muitos zeros deixam de sê-lo, aumentando a carga de memória e de processamento durante a solução. Além disso, pela impossibilidade de divisão por zero, exigem técnicas especiais para a escolha dos pivots e também para reduzir a expansão de A, mas nem sempre isso é possível. Nestes casos os métodos iterativos são a solução.

Outra questão é quanto aos erros de arredondamento. Aqui os métodos diretos são mais suscetíveis. Pode-se amenizar usando as técnicas de pivote-

amento. Já os métodos iterativos, quando tiverem convergência assegurada, podem ser refinados ao limite desejado.

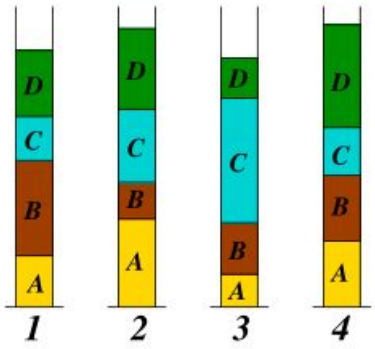
Problemas de Sistemas Lineares

De nada adianta estudar a maneira de resolver problemas lineares, se este estudo não for posto à prática, trabalhando com problemas minimamente reais tirados das engenharias clássicas, a saber: química, de produção, elétrica e civil.

1. Materiais Particulados Suponha 4 tipos de materiais particulados distribuídos em 4 provetas e em cada proveta os materais estão dispostos em camadas, não misturadas, de modo que é possível medir o volume de cada material em cada uma das provetas. Chamando os materiais de A, B, C e D e suas densidades de ρ_A, ρ_B, ρ_C e ρ_D estas são as incógnitas que queremos achar. Os dados disponíveis são:

incógnita	significado
m_1	massa da proveta 1
m_2	massa da proveta 2
m_3	massa da proveta 3
m_4	massa da proveta 4
v_{im}	volume material m na proveta i

Veja-se uma visualização do que se fala



Deve-se lembrar que a massa de material A na proveta 1 é $v_{1A} \times \rho_A$, e assim por diante para todos os materiais. Assim, a massa total na proveta 1 é

$$v_{1A}.\rho_A + v_{1B}.\rho_B + v_{1C}.\rho_C + v_{1D}.\rho_D = m_1$$

Considerando as 4 provetas tem-se 4 equações a 4 incógnitas. Monte e resolva o problema prático:

Para você fazer

m_1	m_2	m_3	m_4
47.96	52.87	47.46	42.60
v_{1A}	v_{1B}	v_{1C}	v_{1D}
10.8	5.5	2.2	8.7
v_{2A}	v_{2B}	v_{2C}	v_{2D}
8.1	3.1	10.2	9.1
v_{3A}	v_{3B}	v_{3C}	v_{3D}
7.8	2.0	8.6	9.2
v_{4A}	v_{4B}	v_{4C}	v_{4D}
4.0	7.2	9.8	4.9

Responda aqui:

ρ_A	ρ_B	ρ_C	ρ_D
3.1			

2. Transportando máquinas Extraído de [Fra07], pág. 166. Uma transportadora possui 5 tipos de caminhões, que são equipados para transportar cinco tipos diferentes de máquinas A, B, C, D e E. Na tabela a seguir, o que cada tipo de caminhão pode levar a carga plena. Se a linha (1), significando caminhão do tipo 1, tiver A=1, B=1, C=1, D=0, E=2 significa que quando totalmente carregado ele pode levar essas quantidades de cada máquina sem sobrar espaço.

Para você fazer

caminhão	máq A	máq B	máq C	máq D	máq E
1	2	5	1	8	3
2	1	7	0	9	6
3	3	2	7	9	9
4	1	9	4	1	7
5	7	5	0	5	7

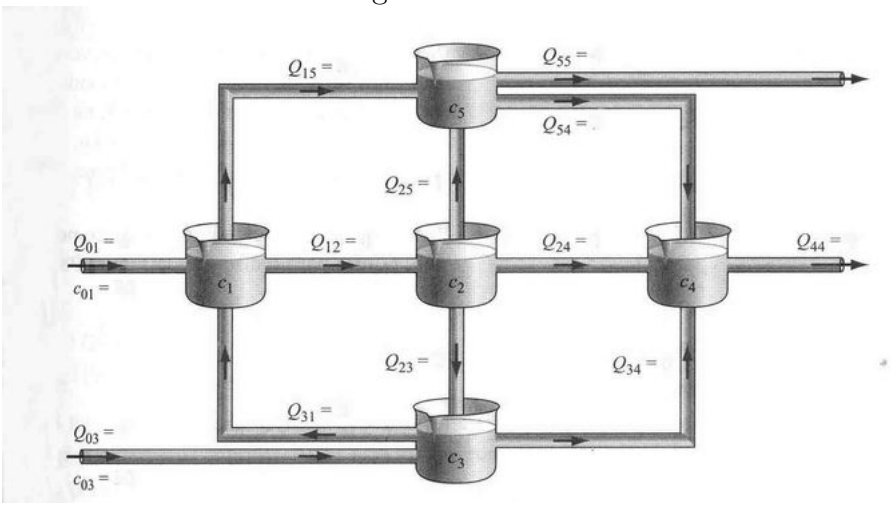
A pergunta é quantos caminhões de cada tipo deve-se usar para transportar exatamente as seguintes quantidades de cada tipo de máquina a carga plena.

maq.A	m.B	m.C	m.D	maq.E
164	368	166	438	412

Responda aqui:

c.t.1	c.t.2	c.t.3	c.t.4	c.t.5
16				

3. Reatores Extraído de [Cha13], pág. 227. Cinco reatores acoplados por tubos estão ilustrados na figura abaixo



Seja uma instância do problema na qual os valores são $Q_{15} = 5$, $Q_{55} = 4$, $Q_{54} = 2$, $Q_{25} = 1$, $Q_{01} = 6$, $Q_{12} = 4$, $Q_{24} = 1$, $Q_{44} = 9$, $Q_{23} = 2$, $Q_{34} = 6$, $Q_{31} = 3$, $Q_{03} = 7$, $C_{01} = 20$ e $C_{03} = 50$. A taxa de de fluxo de massa através de cada tubo é calculada como o produto de fluxo (Q) pela concentração (c). Em regime permanente ou seja, estacionário, o fluxo de massa para dentro e para fora de cada reator deve ser igual. Neste exemplo, os balanços de massa

podem ser escritos como
reator 1 $6 \times 20 + 3 \times r_3 = 5 \times r_1 + 4 \times r_1$
reator 2 $4 \times r_1 = r_2 + r_2 + 2 \times r_2 = 4 \times r_2$
reator 3 $2 \times r_2 + 7 \times 50 = 3 \times r_3 + 6 \times r_3$
reator 4 $r_2 + 2 \times r_5 + 6 \times r_3 = 9 \times r_4$
reator 5 $r_2 + 5 \times r_1 = 4 \times r_5 + 2 \times r_5$
Daqui:
Para o reator 1: $120 + 3.r_3 = 9.r_5$
Reator 2: $r_1 = r_2$
Reator 3: $2.r_2 + 350 = 9.r_3$
Reator 4: $r_2 + 2.r_5 + 6.r_3 = 9.r_4$
Reator 5: $r_2 + 5.r_1 = 6.r_5$
E daqui:
Reator 1: $9r_1 + 0r_2 - 3r_3 + 0r_4 + 0r_5 = 120$
Reator 2: $r_1 - r_2 + 0r_3 + 0r_4 + 0r_5 = 0$
Reator 3: $0r_1 + 2r_2 - 9r_3 + 0r_4 + 0r_5 = -350$
Reator 4: $0r_1 + 1r_2 + 6r_3 - 9r_4 + 2r_5 = 0$
Reator 5: $5r_1 + r_2 + 0r_3 + 0r_4 - 6r_5 = 0$
E resolvendo obtem-se $r_1 = 28$, $r_2 = 28$, $r_3 = 45.2$, $r_4 = 42.7$ e $r_5 = 42.6$ que é a resposta procurada. Simplificando e resolvendo, deve-se notar que o sistema é esparso (muitos zeros) o que eventualmente impede de usar o algoritmo ingênuo de Gauss. Se for o caso, use as ferramentas estudadas (APL, Maple, Freemat, pacote Numpy do Python ...)
Para você fazer

Q_{15}	Q_{55}	Q_{54}	Q_{25}	Q_{01}
4	12	9	2	6

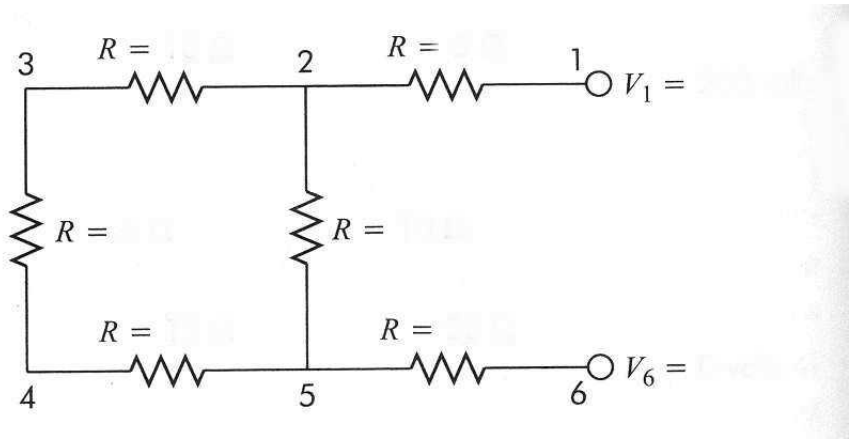
Q_{12}	Q_{24}	Q_{44}	Q_{23}	Q_{34}
14	3	20	16	18

Q_{03}	Q_{31}	c_{01}	c_{03}	xxx
15	10	30	20	xxx

Responda aqui: (concentração de cada reator)

c.r.1	c.r.2	c.r.3	c.r.4	c.r.5
20.23				

4. Circuito Elétrico Extraído de [Cha13], pág. 224. A lei das tensões especifica que a soma algébrica das quedas de tensão (ou das diferenças de potencial) em qualquer caminho fechado deve ser igual a zero. Usando também a lei de Ohm ($V = ri$) obtém-se um sistema de equações algébricas lineares simultâneas porque os vários laços do circuito estão interconectados.



Seja uma instância do problema onde: $R_{12} = 5\Omega$, $R_{23} = 10\Omega$, $R_{34} = 5\Omega$, $R_{45} = 15\Omega$, $R_{25} = 10\Omega$ e $R_{56} = 20\Omega$ e $V_1 = 200V$ e $V_6 = 0V$.

Assumindo os sentidos positivos como $3 \to 2$, $4 \to 3$, $5 \to 4$, $6 \to 5$, $5 \to 2$ e $1 \to 2$ (se o resultado der negativo, basta inverter o sentido). Aplicando a lei de Kirchhoff tem-se

nodo 2: $i_{12} + i_{52} + i_{32} = 0$

nodo 5: $i_{65} - i_{52} - i_{54} = 0$

nodo 3: $i_{43} - i_{32} = 0$

nodo 4: $i_{54} - i_{43} = 0$

Aplicando a lei das tensões a cada um dos laços, tem-se

$-i_{54}.R_{54} - i_{43}.R_{43} - i_{32}.R_{32} + i_{52}.R_{52} = 0$

$-i_{65}.R_{65} - i_{52}.R_{52} + i_{12}.R_{12} - 200 = 0$

substituindo as resistências e trazendo as constantes para o lado direito, fica:

$-15.i_{54} - 5.i_{43} - 10.i_{32} + 10.i_{52} = 0$

$-20.i_{65} - 10.i_{52} + 5.i_{12} = 200$

Agora as 6 correntes desconhecidas podem ser representadas na forma matricial como

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 10 & -10 & 0 & -15 & -5 \\ 5 & -10 & 0 & -20 & 0 & 0 \end{bmatrix} \begin{Bmatrix} i_{12} \\ i_{52} \\ i_{32} \\ i_{65} \\ i_{54} \\ i_{43} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 200 \end{Bmatrix}$$

Resolvendo este sistema encontra-se a solução: 6.15, -4.6, -1.53, -6.15, -1.53 e -1.53.

Para você fazer

R_{12}	R_{23}	R_{34}	R_{45}
35	35	20	5
R_{25}	R_{56}	V_1	V_6
30	25	150	0

Responda aqui:

i_{12}	i_{52}	i_{32}	i_{65}	i_{54}	i_{43}
1.88					

5. Cascalho na construção civil Extraído de [Cha13], pág. 252. Um engenheiro civil envolvido em uma construção precisa de areia (A), cascalho fino (F) e cascalho grosso (G) para um projeto de uma obra. Existem 3 minas de onde esses materiais podem ser obtidos. A composição de cada mina é dada. Quantos metros cúbicos devem ser minerados de cada mina para as

necessidades do engenheiro ?

Seja um exemplo: $A = 4800m^3$, $F = 5800m^3$ e $G = 5700m^3$, com as seguintes proporções de cada mina:

	Areia	C. fino	C. grosso
Mina1	52	30	18
Mina2	20	50	30
Mina3	25	20	55

Para este caso, as respostas são: $M_1 = 4005m^3$, $M_2 = 7131m^3$ e $M_3 = 5162m^3$.

Para você fazer

Suponha que as necessidades da obra são:

Areia	C. fino	C. grosso
8697	9834	7903

E as proporções para cada mina são (em percentagem):

Mina	Areia	C.fino	C.grosso
1	37	39	24
2	34	33	33
3	28	39	33

Responda aqui a quantidade de material a ser extraído em cada mina em m^3.

$m^3M.1$	$m^3M.2$	$m^3M.3$
9119		

Problemas adaptados de: CHAPRA, Steven. **Métodos Numéricos aplicados com MATLAB**. Bookman. São Paulo, 2013 e FRANCO, Neide. **Cálculo Numérico**. Pearson, São Paulo, 2007.

Respostas

1) 0.40
2) 9
3) 5.14
4) -0.62
5) 9402

Sistemas Lineares na prática: Tomografia Computadorizada

Para encerrar a discussão de sistemas lineares com uma aplicação robusta e real, vai-se trabalhar com a reconstituição de imagens que é feita pela máquina de TCA (tomografia computadorizada axial).

Começa-se a discussão pela presença muito comum de um sistema de m equações a n incógnitas estudando-se agora o que acontece com os diversos valores de m e n.

Quando $m = n$, tem-se o caso estudado até aqui nas aulas anteriores, já que se as m equações são linearmente independentes (nenhuma é múltipla de outra ou combinação linear de outras) o sistema tem uma única solução.

Se $m < n$ o sistema só admite solução se forem arbitrados valores **ad-hoc** para $m - n$ incógnitas.

O caso que vai ser estudado agora é $m > n$ que é o caso mais comum na realidade. Em uma rápida análise, comecemos com um sistema $m = n$ e acrescentemos ao sistema uma nova equação linearmente independente.

Para simplificar usando um caso real suponhamos 3 equações e 3 incógnitas: o sistema tem uma solução x_1, x_2 e x_3. Ao acrescentar a quarta equação o

sistema deixa de ter solução exata. Mas, se pegarmos 3 dessas 4 equações (por exemplo as equações 1, 2 e 4) ele voltará a ter solução. Vamos chamá-la de x_1, x_2 e x_4. Se pegarmos as equações 1, 3 e 4, nova solução. Ao resolver o sistema com as 4 equações quer-se o ponto do espaço que minimiza a distância (mínimos quadrados) das soluções de 4 equações tomadas 3 a 3: a saber 1,2,3; 1,2,4; 1,3,4 e 2,3,4.

Esta estratégia faz todo o sentido, já que na vida real não existem soluções exatas, por diversas razões, sendo a principal a omnipresença do ruído, aqui entendido como erro (de leitura, de processamento, de transcriçao, do universo contra mim...).

Não vamos trabalhar com este algoritmo ele é bastante sofisticado, mas vão-se usar os pacotes à nossa disposição (APL, Freemat, Matlab, Python, ...). Neste último não se usa `numpy.linalg.solve` já que este exige matrizes quadradas, mas `numpy.linalg.lstsq`. Este exercício pede que você resolva um sistema com 170 equações a 64 incógnitas.

Tomografia Computadorizada Axial Trata-se de uma técnica que usa um computador e uma máquina de Raio X para reconstituir imagens de maneira transversal de um corpo. Originalmente na pesquisa do seu inventor, o programador inglês Godfrey Hounsfield, era apenas o cérebro humano já que este – por estar dentro de um capacete de matéria óssea: o crânio – é impermeável ao Raio-X exceto nos casos de rachaduras ou quebra. Mais tarde a técnica deu tão certo que acabou sendo adaptada a outras partes do corpo, mesmo de animais e também em alfândegas, presídios e similares para examinar o interior de qualquer coisa. A propósito até onde sei, o Hounsfield foi o único não médico a ganhar o Nobel de Medicina até hoje.

Feixes de raio X são disparados sobre o corpo em análise e ao invés de serem registrados em fotografias são entregues a um computador (a você...) para processamento. A máquina toda gira ao redor do corpo tirando muitos tomogramas do mesmo objeto. A variação angular é que vai permitir a reconstituição posterior.

O coeficiente de atenuação linear média μ_t de cada pixel é comparado com o coeficiente da água, μ_a, definindo o número CT:

$CT = 1000(\mu_t - \mu_a)/\mu_a$.

A água é utilizada como referência porque seu coeficiente de atenuação é similar ao dos tecidos moles, e é um material fácil de obter para calibrar os aparelhos. O coeficiente 1000 é utilizado para obter números inteiros.

O número CT, ou coeficiente de Hounsfield, é definido como -1000 para o ar e 0 para a água.

Para os tecidos em geral, ele depende da energia do feixe empregado. Por exemplo, para 80 keV, se o coeficiente de atenuação linear típico de ossos é de 0,38 cm^{-1}, e da água 0,19 cm^{-1}, o número CT dos ossos é de +1000. Pode ser ainda maior para ossos corticais. Estes valores também variam de aparelho para aparelho, já que os coeficientes dependem da distribuição de energia do feixe. A radiação observada, I, está relacionada com a radiação na fonte, I_0, por: $I = I_0 e^{-\mu x}$.

Tecido	CT	Tecido	CT
Ar	-1000	Sangue	35:55
Pulmão	-900:-400	Coágulo	80
Gordura	-110:-65	Músculo	40:60
Água	0	Fígado	50:85
Rim	30	Osso	130:250

Por convenção, altos valores de CT são imageados como branco, e baixos como preto. Como o olho humano não pode distinguir os milhares de coeficientes, utilizamos a técnica de janelas (windowing), para graficar somente os valores em uma certa faixa.

Aquisição de dados Na tomografia computadorizada mais comum, um tubo com um feixe de cerca de 0,6 mm de diâmetro gira em torno do paciente, e emerge do paciente sobre um detector com aproximadamente 700 sensores, que convertem a intensidade em uma corrente. Cada pulso de raio-X dura 2 a 3 ms, completando uma volta em cerca de 1 s. Cada 360° gera 300 somas.

Cada vez que o tubo emite um pulso, cada detector mede o logaritmo da intensidade que recebe. Este valor representa a soma de todos os números CT dos voxeis atravessados pelo raio, completando uma projeção. Cada voxel é atravessado pelo feixe em diferentes direções, durante a rotação do anel. O número CT de cada voxel está portanto representado em várias somas. (Estas últimas informações foram retiradas em http://www.if.ufrgs.br/ast/med/imagens/imagens.htm por Kepler de Souza Oliveira Filho).

Algoritmo de reconstrução Vamos pensar em uma sessão transversal de um corpo humano. Imagine também um conjunto de emissores de raio-X, colocados paralelos em uma estrutura rígida que gira em um eixo estabelecido no centro do corpo humano. No lado oposto (ultrapassado o corpo) estão os detectores do raio-X.

À medida em que a máquina gira, inumeros conjuntos de Raios-X vão sendo tomados. Para uma determinada posição angular (digamos horizontal) tem-se um conjunto de x valores de atenuação. O valor x é a quantidade de emissores e detetores, e a atenuação é a diminuição da potência do Raio-X emitido numa ponta e recebido na outra após passar pelas estruturas do corpo que está sendo estudado.

Por exemplo, na horizontal, os raios inferiores, tem que atravessar a coluna (supondo que o paciente esteja deitado de costas), ao passo que os raios superiores ou não atravessam nada, ou apenas alguma gordura (isso para os mais gordinhos). Quando a estrutura estiver tirando raios X na vertical, a coluna do paciente atenuará os raios do meio do conjunto.

Os detectores não tiram fotografias, ao invés eles mandam sinais elétricos proporcionais a potência do Raio-X recebido diretamente para um computador. Este, de posse da informação referente ao ângulo de tomada e dos valores de todos os sensores, vai reconstituindo do corpo que está sendo "cortado".

A técnica mais simples de reconstrução é a chamada "back projection", e todas as demais que vieram depois são melhoramentos e variações desta.

Sua formulação matemática é bem compacta:

$$P(\theta, t) = \int_R f(x, y)\delta(x \sin\theta - y \cos\theta - t)dxdy$$

Essa fórmula pode ser lida como: P é a atenuação medida no sensor t, posicionado em um ângulo θ em relação ao eixo horizontal. x, y são as coorde-

nadas dos pontos da imagem e δ é um filtro que tem a função de só considerar os pontos que estão no caminho do fluxo t de Raios X.

Neste exercício imagine uma tomografia de um único plano, de um corpo quadrado de 8×8 binário (valendo 0 ou 1 apenas). A cada tomograma há 10 feixes, e depois há um giro de 5^o graus no sentido horário. Começando em 5^o e terminando em 85^o, são 17 rotações. Então são $17 \times 10 = 170$ equações e 64 variáveis.

Você vai receber 2 arquivos: o primeiro de nome `Fk37COEF.myd` que está publicado em `algoritmovivo.com`, pasta "atividades"e que contém os coeficientes do sistema linear que deverá usar para reconstituir a imagem. Estes coeficientes formam uma matriz de 170 linhas e 64 colunas, contendo números reais entre 0 e 1. Esta matriz é a mesma para todos pois ela está associada a este exercício. Trata-se da matriz A em $Ax = B$.

Além deste arquivo você deve recuperar o arquivo

`FK300001.myd`

que contém a coluna de termos independente do seu exercício (o B em $Ax = B$) que é único para você.

Depois de resolver o sistema linear (usando o método dos mínimos quadrados), responda o que tem nos pixels das linhas 1..4 (primeira é a superior) na coluna 8 (a última).

Para testar seus programas, confira os arquivos `Fk37EXEn.myd` no mesmo local: o primeiro tem apenas as 2 primeiras linhas iguais a 1, as demais são zero. O segundo é a negação do primeiro. O terceiro é aleatório, e as posições pedidas (últimas colunas, primeira à quarta linha) são: 1 1 0 1. Finalmente o exemplo 4 traz apenas as colunas 3, 4 e 5 valendo 1. As demais valem zero.

Leitura de dados em Python Para ter acesso aos dados, você vai precisar baixar os arquivos publicados em seu computador e processar um programa Python semelhante a este:

```
import numpy as np
def leitcoef():
    coef=np.zeros((680,16),float)
    ref=open("f:/n/k37/fk37coef.myd","r")
    lin=ref.readline()
    i=0
    while (0!=len(lin)):
        lin=lin.split()
        for j in range(0,16):
            coef[i,j]=float(lin[j])
        i=i+1
        lin=ref.readline()
    coef.resize([170,64])
    return(coef)
a=leitcoef()
print(a)
```

Ao final deste trecho `a` será uma matriz contendo os coeficientes A da matriz de solução deste problema.

A seguir, voce vai precisar ler seus coeficientes particulares. Faça

```
def leiti():
    ti=np.zeros((170),float)
    ref=open("x:/xxx/fk3700nn.myd","r")
    lin=ref.readline()
    i=0
    while (0!=len(lin)):
        ti[i]=float(lin)
        i=i+1
        lin=ref.readline()
    return(ti)
b=leiti()
print(b)
```

Finalmente, a solução do sistema linear usa a funçao `lstsq` do pacote linalg de numpy. Acompanhe

```
c=np.linalg.lstsq(a,b,rcond=None)
ima=np.zeros((8,8),float)
k=0
xx=c[0]
for i in range(0,8):
    for j in range(0,8):
        ima[i,j]=abs(float("%8.3f" % xx[k]))
        k=k+1
print(ima)
```

Responda aqui:

L1 C8	L2 C8	Ł3 C8	Ł4 C8

Para saber mais Consulte:

DEWDNEY, A.K. **The Turing Omnibus**. New York, Freeman Co. 1989.
Kepler de Souza Oliveira Filho. In http://www.if.ufrgs.br/ast/ med/imagens/ imagens.htm
GUSTAVSON, Katarina. **The Role of Linear Algebra in Computed Tomography** In
https://raysforexcellence.squarespace.com/.

Capítulo 10

Interpolação

10.1 Polinomial

Suponha que você está estudando um fenômeno físico qualquer e ocorre uma destas duas situações:

- Não se tem idéia da expressão analítica (fórmula) que descreve o fenômeno, e o que se tem são apenas pares de valores levantados *in loco* (no campo).
- A fórmula é conhecida, mas é complicada, eventualmente descontínua o que inviabiliza derivadas e integrais a ela associadas.

Uma proposta de tratamento para esta classe de problemas é procurar um polinômio que se aproxime tanto quanto possível (e desejado) da função desconhecida original. Agora, todo o tratamento analítico se dá sobre o polinômio aproximado e não sobre a função original (que lembrando, ou é desconhecida ou é muito complexa). No primeiro caso viabiliza-se o tratamento e no segundo troca-se precisão por simplicidade no manuseio do problema.

Algumas vantagens deste polinômio:

- polinômios são facilmente computáveis;
- suas derivadas e suas integrais continuam sendo polinômios;
- é fácil (e já é conhecido) achar suas raízes;

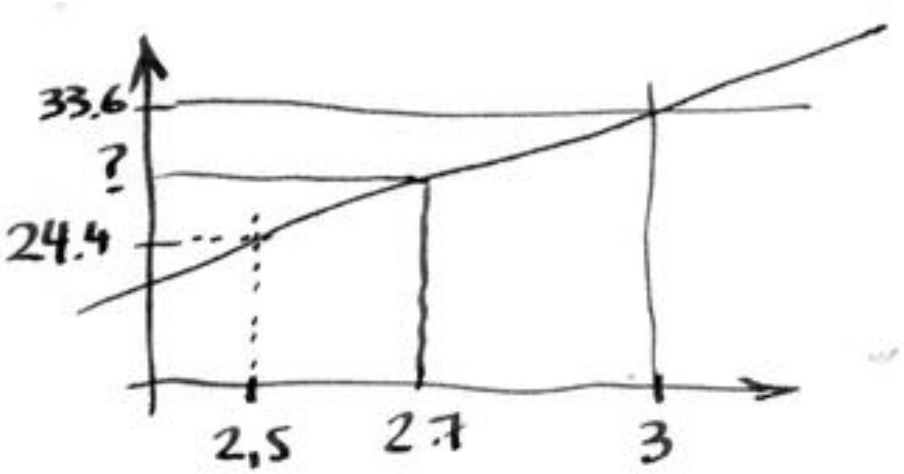
Um exemplo

Suponha um sistema hidráulico para transporte de água. Suas perdas supostamente são proporcionais à vazão do sistema. Em campo, levantaram-se os seguintes dados

vazão em m^3/h	perdas em %
1.5	10
2	16.5
2.5	24.2
3	33.6
3.5	44
4	55.6

A questão que se poderia levantar aqui é **Qual seria a perda do sistema**

em uma vazão de $2.7m^3/h$ **?** Desenhando isto num gráfico poder-se-ia ter



Por semelhança de triângulos pode-se escrever

$$\frac{\Delta y}{\Delta x} = \frac{33.6 - 24.4}{3 - 2.5} = \frac{y - 24.4}{2.7 - 2.5}$$

e daqui $y = 28.08$.

Como consideraram-se 2 pontos a aproximação obtida é linear. A regra da interpolação é

para n pontos $\Rightarrow$ polinômio grau $n-1$

Neste caso o polinômio linear (uma reta) é

$$P_1 = a_0 + a_1 x$$

onde a_0 é o coeficiente linear da reta e a_1 é o angular. Aplicando-se esta abordagem a dois pontos, tem-se
$a_0 + a_1 x_1 = y_1$
$a_0 + a_1 x_2 = y_2$
Ou na forma matricial

$$\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \end{bmatrix} . \begin{Bmatrix} a_0 \\ a_1 \end{Bmatrix} = \begin{Bmatrix} y_1 \\ y_2 \end{Bmatrix}$$

Que pode ser resolvido por qualquer dos métodos já conhecidos. Aqui vai-se usar o método de escalonamento de Gauss. Por exemplo:

$$\begin{bmatrix} 1 & 2.5 \\ 1 & 3 \end{bmatrix} . \begin{Bmatrix} a_0 \\ a_1 \end{Bmatrix} = \begin{Bmatrix} 24.4 \\ 33.6 \end{Bmatrix}$$

o que dará $a_0 = -21.6$ e $a_1 = 18.4$. Daqui, $P_1(2.7) = 28.08$.

Ao refazer o mesmo exemplo para 3 pontos, vai-se criar agora um polinômio de grau 2 (3-1=2) no formato

$$P_2(x) = a_2 x^2 + a_1 x + a_0$$

e o sistema fica

$$\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{bmatrix} . \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix} = \begin{Bmatrix} y_1 \\ y_2 \\ y_3 \end{Bmatrix}$$

```
def intpol(a,y):
    import numpy as np
    n=len(a)
    x=np.zeros((n),float)
    passo=0
    while passo<n-1:
        i=passo+1
        while i<n:
            pivot=a[i,passo]/a[passo,passo]
            j=0
            while j<n:
                a[i,j]=a[i,j]-pivot*a[passo,j]
                j=j+1
            y[i]=y[i]-pivot*y[passo]
            i=i+1
        passo=passo+1
    x[n-1]=y[n-1]/a[n-1,n-1]
    i=n-1
    while i>=0:
        x[i]=y[i]
        j=i+1
        while j<n:
            x[i]=x[i]-a[i,j]*x[j]
            j=j+1
        x[i]=x[i]/a[i,i]
        i=i-1
    xint = float(input('Qual o valor a interpolar ? '))
    yint=0.0
    i=0
    while i<n:
        yint=yint+(x[i]*xint**(i))
        i=i+1
    print('y interpolado ',yint)

import numpy as np
a=np.array([[1,49,0,0,0],[1,75,0,0,0],[1,101,0,0,0],
          [1,127,0,0,0],[1,153,0,0,0]],float)
b=np.array([289.6, 776, 1946.5, 4736.3, 11348.5],float)
i=0
while i<5:                    #importante: este trecho
    j=2                       # e' opcional e serve
    while j<5:                # para preencher x^2,
        a[i,j]=a[i,1]**j      # x^3 e x^4 que ficaram
        j=j+1                 # zerados lá na entrada
    i=i+1                     # de dados... veja la'
intpol(a,b)
```

10.2 Lagrange

Começa-se definindo um polinômio a ser usado nos pontos $x_0, x_1, ...x_n$ ($n+1$ pontos):

$$\begin{aligned} P_0(x) = &\quad (x-x_1)(x-x_2)...(x-x_n) \\ P_1(x) = &\quad (x-x_0)(x-x_2)...(x-x_n) \\ &... \\ P_n(x) = &\quad (x-x_0)(x-x_1)...(x-x_{n-1}) \end{aligned}$$

Note que no $P_k(x)$ o termo $x - x_k$ é excluído do produto.

Vai-se definir agora uma família de polinômios (conhecidos como de Lagrange)

$$\ell_k(x) = \frac{(x-x_0)...(x-x_{k-1})(x-x_{k+1})}{(x_k-x_0)...(x_k-x_{k-1})(x_k-x_{k+1})}$$

Note que esta expressão apresenta o seguinte comportamento

$$\ell_k(x_j) = \begin{cases} 0 \; se \; k \neq j \\ 1 \; se \; k = j \end{cases}$$

Um parênteses, para entender o resultado acima:
seja

$$\ell_k(x_j) = \frac{(x_j-x_0)...(x_j-x_{k-1})(x_j-x_{k+1})}{(x_k-x_0)...(x_k-x_{k-1})(x_k-x_{k+1})}$$

o que acontece se $k = 2$ e $j = 3$?

$$\ell_2(x_3) = \frac{(x_3-x_0)...(x_3-x_1)(x_3-x_3)}{(x_2-x_0)...(x_2-x_1)(x_2-x_3)}$$

note que o $(x_3 - x_3)$ no denominador vale 0, logo a fração toda se anula.
O que acontece agora se $k = 3$ e $j = 3$?

$$\ell_3(x_3) = \frac{(x_3-x_0)...(x_3-x_2)(x_3-x_4)}{(x_3-x_0)...(x_3-x_1)(x_3-x_4)}$$

veja que agora o numerador e o denominador são absolutamente iguais, o que simplificando, dá 1. Fecha o parênteses.

Então, supondo um fenômeno do qual se conhecem pares de valores $x_0, f_0 = f(x_0)$, $x_1, f_1 = f(x_1)$, ... $x_n, f_n = f(x_n)$, de uma função genérica (possivelmente desconhecida) $y = f(x)$, o polinômio

$$P_n(x) = \sum_{k=0}^{n} f_k \ell_k(x)$$

é chamado de **Fórmula de Lagrange do polinômio de interpolação**. Note que ele é de grau no máximo n e satisfaz

$$P_n(x_k) = f(x_k) \; para \; k = 0, 1, 2, ...n$$

Exemplo

Sejam 3 pontos:

x	$f(x)$
-1	15
0	8
3	-1

E seja descobrir quanto a função vale no ponto $x = 1$. Para isso, deve-se achar

o polinômio de interpolação de Lagrange. Tem-se
$x_0 = -1,\ f(x_0) = 15$
$x_1 = 0,\ f(x_1) = 8$
$x_2 = 3,\ f(x_2) = -1$
$n = 2$ e daqui $P_2(x) = \sum_{k=0}^{2}\ f_k \ell_k(x)$.

Determinando os $\ell_k(x),\ k = 0, 1, 2$:

$$\ell_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} = \frac{(x - 0)(x - 3)}{(-1 - 0)(-1 - 3)} = \frac{x^2 - 3x}{4}$$

$$\ell_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} = \frac{(x + 1)(x - 3)}{(0 + 1)(0 - 3)} = \frac{x^2 - 2x - 3}{-3}$$

$$\ell_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} = \frac{(x + 1)(x - 0)}{(3 + 1)(3 - 0)} = \frac{x^2 + x}{12}$$

e

$$P_2(x) = f_0\ell_0(x) + f_1\ell_1(x) + f_2\ell_2(x) =$$

$$15\left[\frac{x^2 - 3x}{4}\right] + 8\left[\frac{x^2 - 2x - 3}{-3}\right] + (-1)\left[\frac{x^2 + x}{12}\right]$$

cozinhando os dados e agrupando fica:

$$P_2(x) = x^2 - 6x + 8$$

e daqui $f(1) \cong P_2(1) = 3$

Esquema prático

Se for escrever um programa de computador para obter o valor da função em um ponto através do polinômio de interpolação pode-se usar um esquema prático o qual acha este ponto sem determinar a expressão do polinômio:

$$P_n(x) = \sum_{i=0}^{n} y_i . \prod_{j=0, j\neq i}^{n} \frac{(x - x_j)}{(x_i - x_j)}$$

Escrevendo esta fórmula em termos mais amigáveis:

$$P(x) = y_0 . \frac{\frac{Prod_x}{Dif_o}}{Prod_0} + y_1 . \frac{\frac{Prod_x}{Dif_1}}{Prod_1} +$$

$$y_2 . \frac{\frac{Prod_x}{Dif_2}}{Prod_2} + ... + y_n . \frac{\frac{Prod_x}{Dif_n}}{Prod_n}$$

Uma ótima atividade aqui é refazer o exemplo acima $\{x_0 = -1,\ f(x_0) = 15;$ $x_1 = 0,\ f(x_1) = 8;\ x_2 = 3,\ f(x_2) = -1$ e $n = 2\}$ usando o esquema prático. O resultado tem que ser o mesmo, e ficará evidente que agora há muito menos trabalho.

A seguir, mais um exemplo, seja a função $f(x)$ conhecida nos pontos:

i	x_i	$f(x_i) = y_i$
0	0.00	1.000
1	0.10	2.001
2	0.30	4.081
3	0.60	8.296

Determinar o valor para $f(0.20)$ aplicando o esquema prático de Lagrange

	$x_0 =$	$x_1 =$	$x_2 =$	$x_3 =$	$\prod$
-	0.00	0.10	0.30	0.60	
	$D_0 =$	$D_1 =$	$D_2 =$	$D_3 =$	$P_x =$
$x = 0.20$	0.20	0.10	-0.10	-0.40	$P_x = 0.0008$
$x_0 = 0.00$	-	-0.10	-0.30	-0.60	$P_0 = -0.018$
$x_1 = 0.10$	0.10	-	-0.20	-0.50	$P_1 = 0.01$
$x_2 = 0.30$	0.30	0.20	-	-0.30	$P_2 = -0.018$
$x_3 = 0.60$	0.60	0.50	0.30	-	$P_3 = 0.09$

e agora

$$P_x(0.2) = (1.000 \times \frac{\frac{0.0008}{0.20}}{-0.018}) + (2.001 \times \frac{\frac{0.0008}{0.10}}{0.01}) +$$

$$(4.081 \times \frac{\frac{0.0008}{-0.10}}{-0.018}) + (8.296 \times \frac{\frac{0.0008}{-0.40}}{0.09}) = 3.008$$

```
def lagra(x,y):
    xinterp=float(input("entre valor interp."))
    n=len(x)
    dif=[0]*n
    prod=[0]*n
    j=0
    while j<n:
        dif[j]=xinterp-x[j]
        j=j+1
    prodx=1
    j=0
    while j<n:
        prodx=prodx*dif[j]
        j=j+1
    i=0
    while i<n:
        prod[i]=1
        j=0
        while j<n:
            if i!=j:
                prod[i]=prod[i]*(x[i]-x[j])
            j=j+1
        i=i+1
    yinterp=0
    i=0
    while i<n:
        yinterp=yinterp+y[i]*
                    ((prodx/dif[i])/prod[i])
```

```
        i=i+1
    print('vetor diferencas ',dif)
    print('prodx ',prodx)
    print('vetor produtos ',prod)
    print('y interpolado ',yinterp)

#a=[-1.0,0,3]
#b=[15.0,8,-1]
a=[0.0,0.1,0.3,0.6]
b=[1.0,2.001,4.081,8.290]
lagra(a,b)
```

Uma implementação mais genérica pode ser

```
def lagrange(a,b,x):
#a=x, b=f(x), x=a interpolar
  n=len(a)
  dif=[0.0]*n
  pro=[0.0]*n
  for i in range(n):
    dif[i]=x-a[i]
  prodx=1
  for i in range(n):
    prodx=prodx*dif[i]
  for i in range(n):
    pro[i]=1
    for j in range(n):
      if (j!=i):
        pro[i]=pro[i]*
         (a[i]-a[j])
  y=0
  for i in range(n):
    y=y+b[i]*((prodx/dif[i])/
       pro[i])
  return y
print(lagrange([-1.0,0,2,3],
      [4.0,1,1,16],4))
>>> 26810.129812286617
```

10.3 Newton

Aqui vai-se estudar uma técnica que evita um inconveniente do método de Lagrange, a saber: sempre que se vai passar de um polinômio de grau p (construído com $p+1$ pontos) para um de grau $p+1$ (com $p+2$ pontos) lá, o trabalho tem que ser praticamente refeito do zero. O ideal seria passar de $p+1$ para $p+2$ pontos apenas acrescentando um termo ao polinômio. Para isso pode-se usar este algoritmo, mas antes se precisa definir a noção de diferença dividida de uma função.

Tabela de diferenças divididas Dada um conjunto de x e $f(x)$ (de ordem n) como até agora visto, construa-se uma tabela de n linhas por $n+1$ colunas

onde

- a primeira coluna contém os pontos x_k
- a segunda coluna contém $f(x)_k$ nos pontos equivalentes
- as colunas seguintes contém as diferenças divididas: trata-se de uma fração onde o numerador é a diferença entre os valores da coluna anterior e da linha atual e da próxima. O denominador é a diferença (sempre na coluna 1) entre os extremos dos pontos envolvidos.

Analiticamente $f[x_0, x] = \frac{f[x]-f[x_0]}{x-x_0}$ e $f[x_0, x_1, x] = \frac{f[x_0,x]-f[x_0,x_1]}{x-x_1}$ e $f[x_0, x_1, x_2, x] = \frac{f[x_0,x_1,x_2,x]-f[x_0,x_1,x_2]}{x-x_2}$ Da primeira equação, $f(x) = f[x_0] + (x - x_0)f[x_0, x]$ e daqui recursivamente, extendendo a expressão chega-se a $P_n(x) = f[x_0] + (x - x_0)f[x_0, x_1]+(x-x_0)(x-x_1)f[x_0, x_1, x_2]+...+(x-x_0)...(x-x_{n-1})f[x_0, x_1, ..., x_n] = \{...\}$.

Acompanhe o exemplo
Seja um conjunto de pontos x_i e os valores da função a eles associados $f(x_i)$.

x	-2	-1	0	1	2
$f(x)$	-2	29	30	31	62

Note que os valores calculados estão em cores diferentes para que se possa perceber o mecanismo de construção desta tabela: (se a sua impressão atual é monocromática, localize no lugar de sempre, o arquivo .PDF equivalente)

x_i	$f(x_i)$				
-2	-2	$\frac{29-(-2)}{(-1)-(-2)} = 31$	$\frac{1-31}{0-(-2)} = -15$	$\frac{0-(-15)}{1-(-2)} = 5$	$\frac{5-5}{2-(-2)} = 0$
-1	29	$\frac{30-29}{0-(-1)} = 1$	$\frac{1-1}{1-(-1)} = 0$	$\frac{15-0}{2-(-1)} = 5$	
0	30	$\frac{31-30}{1-0} = 1$	$\frac{31-1}{2-0} = 15$		
1	31	$\frac{62-31}{2-1} = 31$			
2	62				

A partir da tabela pode-se construir o polinômio interpolador. Note que as cores foram mantidas para que se possa perceber o mecanismo usado. Veja que apenas as primeiras linha e coluna é que são usadas na construção do polinômio

$$P(x) = -2 + [(x-(-2))] \times 31 + [(x-(-2)) \times (x-(-1))] \times -15$$
$$+[(x-(-2)) \times (x-(-1)) \times (x-0)] \times 5$$
$$+[(x-(-2)) \times (x-(-1)) \times (x-0) \times (x-1)] \times 0$$

Esquema numérico

Existe um esquema prático para calcular o valor de um polinômio de interpolação em um novo ponto (não tabelado) sem precisar determinar a expressão do polinômio. A idéia é ir colocando os termos comuns em evidência. Acompanhe um exemplo de grau 3:
$f[x_0, x_1, x_2, x_3] = \alpha_0$,
$f[x_0, x_1, x_2] + (x - x_2)\alpha_0 = \alpha_1$
$f[x_0, x_1] + (x - x_1)\alpha_1 = \alpha_2$
$f[x_0] + (x - x_0)\alpha_2 = \alpha_3 = P_3(x)$

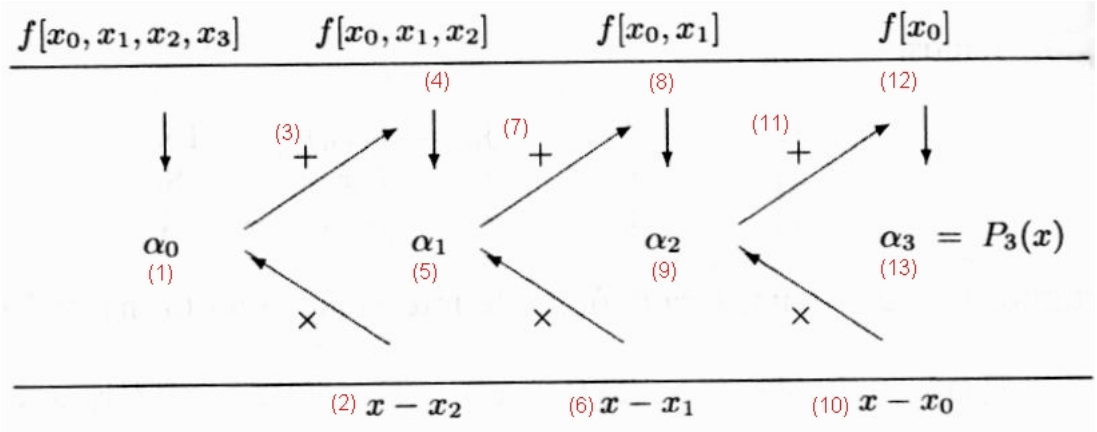
Vamos fazer um exemplo completo de grau=3. Seja a seguinte tabela:

$x =$	-2	1	2	3
$f(x)$	-26	-2	2	14

Construindo a tabela de diferenças divididas, fica:

	$f[x_0]$	$f[x_0, x_1]$	$f[x_0, x_1, x_2]$	$f[x_0, x_1, x_2, x_3]$
2	-26	8	-1	1 $(= \alpha_0)$
1	-2	4	4	
2	2	12		
3	14			

Suponhamos querer calcular $f(5)$. Portanto agora $x = 5$. Acompanhe cada etapa no desenho a seguir



Acompanhe cada passo
(1) Recupera-se $\alpha_0 = f[x_0, x_1, x_2, x_3]$ diretamente na tabela de diferenças divididas e ele vale 1.
(2) Calcula-se $x - x_2$ e $5 - 2 = 3$ que é devidamente multiplicado por α_0 que aqui vale 1, e daqui $3 \times 1 = 3$
(3) Somado com o valor de
(4) $f[x_0, x_1, x_2]$ que vale -1 (da tabela de diferenças) fica
(5) $\alpha_1 = 2$
(6) Calculando $x - x_1$ tem-se $5 - 1 = 4$ que multiplicado por $\alpha_1 = 2$ fica $4 \times 2 = 8$
(7) Somado a
(8) $f[x_0, x_1] = 8$ fica $8 + 8 = 16$
(9) e $\alpha_2 = 16$
(10) Agora, calcula-se $x - x_0$ e $5 - (-2) = 7$
(11) Multiplicado por α_2 fica $7 \times 16 = 112$
(12) Somado com $f[x_0]$ que é -26 fica
(13) $112 + (-26) = 86$ que é igual a α_3 e também (e principalmente) igual a $f(5) = 86$.

```
def polnew(a,b):
  t=len(a)
  tb=np.zeros((t,t+1),float)
  for i in range(t):
    tb[i][0]=a[i]
    tb[i][1]=b[i]
  j=2
  while j<t+1:
    for i in range(t-j+1):
      tb[i][j]=(tb[i+1][j-1]-tb[i][j-1])
         /(tb[i+j-1][0]-tb[i][0])
    j=j+1
  print(tb)
```

Para conferir

Suponha o mesmo enunciado para $x = -3, -2, 2, 4, 5$ e $f(x) = 185, 51, -5, 87, 289$. Os valores esperados para a resposta são

```
-3 185 -134 24 -2 1
-2  51  -14 10  6 0
 2  -5   46 52  0 0
 4  87  202  0  0 0
 5 289    0  0  0 0
```

O valor a interpolar é 3 e o resultado interpolado é 11. O coeficiente de x^3 é -3 e a sequência $\alpha_1, \alpha_2, ...$ é -3, 21, -29, 11. Note que o último alpha é obrigatoriamente igual ao valor da função no ponto interpolado (=11).

Se quiser mandar o sympy calcular e imprimir o polinômio de interpolação, atente para o código. Note que aqui enfrentei uma difícil decisão: gerar o polinômio $as - is$ pronto para ser copiado e colado numa janela Python – e neste caso com excesso de parênteses – ou ao contrário gerar uma visão mais *humana* do polinômio, sem tantos e possivelmente irrelevantes parênteses. Usei neste caso os [] que todos usamos lá nos longínquos bancos escolares do ensino fundamental. Esta segunda estratégia é mais fácil de ser humanamente entendida, mas precisa – antes de passar para o python ter os [] trocados por (). Vamos a ela

```
def polin_lagra_gen_pts(a,b):
  import numpy as np
  import sympy as sp
  sp.init_printing(use_unicode=True)
  name="x=sp.symbols('x')"
  exec(name)
  name="P=sp.symbols('P',cls=sp.Function)"
  exec(name)
  for i in range(len(a)):
    name="L"+str(i)+"=sp.symbols('L"+str(i)+"',cls=sp.Function)"
    exec(name)
  for i in range(len(a)):
```

```
    name='L'+str(i)+'=('
    for j in range(len(a)):
      if i!=j:
        name=name+'(x-a['+str(j)+'])'
        if j!=len(a)-1:
          name=name+'*'
    if i==len(a)-1:
      name=name[0:len(name)-1]+')/('
    else:
      name=name+')/('
    for j in range(len(a)):
      if i!=j:
        name=name+'(a['+str(i)+']-a['+str(j)+'])'
        if j!=len(a)-1:
          name=name+'*'
    if i==len(a)-1:
      name=name[0:len(name)-1]+')'
    else:
      name=name+')'
    print(name)
    exec(name)
    name='sp.pprint(L'+str(i)+')'
    exec(name)
  name='P='
  for i in range(len(b)):
    name=name+'(b['+str(i)+']*L'+str(i)+')+'
  name=name[0:len(name)-1]
  print(name)
  exec(name)
  for i in range(len(a)):
    print('---------------L'+str(i)+'-----------')
    name='sp.pprint(sp.expand(L'+str(i)+'))'
    exec(name)
  print('---------------P---------------')
  name="sp.pprint(sp.expand(P))"
  exec(name)
  print('--------P no formato para fazer a interpolacao -----')
  name='print(sp.expand(P))'
  exec(name)
polin_lagra_gen_pts([2.0,3,4,6,9],[41.0,115,215,325,-995])
```

10.4 Newton-Gregory

Este algoritmo parte da fórmula de Newton e imagina que os pontos estudados estão igualmente espaçados e este fato leva a importante simplificação.

Relembrando, a forma de Newton para o polinômio interpolador era

$$P_n(x) = f[x_0] + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2] + ...$$

$$+(x - x_0)(x - x_1)...(x - x_{n-1})f[x_0, x_1, ..., x_n]$$

Ordem	Operador		
0	$f[x_i]=$	$f(x_i)=$	$\Delta^0 f(x_i)$
1	$f[x_i, x_{i+1}]=$	$\frac{f(x_{i+1}-f(x_i)}{x_{i+1}-x_i}=$	$\frac{\Delta^1 f(x_i)}{h}$
2	$f[x_i, x_{i+1}, x_{i+2}]=$	...	$\frac{\Delta^2 f(x_i)}{2h^2}$
3	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]=$	...	$\frac{\Delta^3 f(x_i)}{6h^3}$
...	...		
n	$f[x_i, x_{i+1}, ..., x_{i+n}]=$	...	$\frac{\Delta^n f(x_i)}{n!h^n}$

A hipótese simplificadora aqui é os pontos são igualmente espaçados, por h e fica
$x_1 - x_0 = x_2 - x_1 = ... = x_n - x_{n-1} = h$, e daqui
$x_i = x_{i-1} + h$ e portanto $x_i = x_0 + i \times h$

Agora, podemos calcular as diferenças ordinárias de $f(x)$ sobre os pontos $x_0, x_1, ..., x_n$ como segue:

- a primeira coluna é constituída pelos pontos $x_i,\ i = 0, 1, ..., n$
- a segunda coluna contém os valores de $f(x)$ nos mesmos pontos
- as colunas 3,4,... contém as diferenças ordinárias que são as diferenças ordinárias consecutivas e de ordem imediatamente inferior.

Acompanhe o exemplo: Suponha a função tabelada:

x	-2	-1	0	1	2
$f(x)$	-2	29	30	31	62

Seguindo o esquema acima descrito tem-se

x_k	$\Delta^0 f(x_k)$	$\Delta^1 f(x_k)$	$\Delta^2 f(x_k)$	$\Delta^3 f(x_k)$	$\Delta^4 f(x_k)$
-2	-2	29-(-2)=31	1-31=-30	0-(-30)=30	30-30=0
-1	29	30-29=1	1-1=0	30-0=30	
0	30	31-30=1	31-1=30		
1	31	62-31=31			
2	62				

O operador de diferenças divididas $f[x_i]$ pode ser relacionado com o operador de diferenças finitas Δ assim:

Com este resultado, o polinômio interpolador de Newton-Gregory (para pontos igualmente espaçados) é

$$p_n(x) = f(x_0) + (x - x_0)\frac{\Delta^1 f(x_0)}{h} + (x - x_0)(x - x_1)\frac{\Delta^2 f(x_0)}{h^2 2!} + ... +$$

$$(x - x_0)(x - x_1)...(x - x_{n-1})\frac{\Delta^n f(x_0)}{h^n n!}$$

Ou mais sucintamente

$$p_n(x) = \Delta^0 f(x_0) + \sum_{i=1}^{n} \left\{ \prod_{j=0}^{i-1} (x - x_j) \frac{\Delta^i f(x_0)}{h^i i!} \right\}$$

Ou seja,

$$P(x) = -2 + (x-(-2)\frac{29}{1} + (x-(-2))(x-(-1))\frac{-30}{1^2.2!} + (x-(-2))(x-(-1))(x-0)\frac{30}{1^3.3!}$$

ou

$$P(x) = -2 + 29(x+2) - 15(x+2)(x+1) + 5x(x+2)(x+1)$$

Vamos executar um exemplo: Seja

x	-1	0	1	2	3
$f(x)$	2	1	2	5	10

Montando a tabela, fica

-1	2	-1	2	0	0
0	1	1	2	0	
1	2	3	2		
2	5	5			
3	10				

Antes de continuar, atente que $\Delta^0 f(x_0) = 2$, $\Delta^1 f(x_0) = -1$ $\Delta^2 f(x_0) = 2$, $\Delta^3 f(x_0) = 0$ e $\Delta^4 f(x_0) = 0$.
Tem-se então

$$p_4(x) = \Delta^0 f(x_0) + (x - x_0)\frac{\Delta^1 f(x_0)}{h^1 1!} + (x - x_0)(x - x_1)\frac{\Delta^2 f(x_0)}{h^2 2!} + (x - x_0)$$

$$(x - x_1)(x - x_2)\frac{\Delta^3 f(x_0)}{h^3 3!} + (x - x_0)(x - x_1)(x - x_2)(x - x_3)\frac{\Delta^4 f(x_0)}{h^4 4!}$$

Como Δ^3 e Δ^4 são zero, o polinômio terá grau 2. Como h é a distância entre os pontos da tabela, tem-se $h = 1$ e

$$p_2(x) = 2 + (x - (-1))\frac{-1}{1^1 1!} + (x - (-1))(x - 0)\frac{2}{1^2 2!}$$

$$p_2(x) = 2 + (x + 1)(-1) + (x + 1)(x)$$

ou $p_2(x) = x^2 + 1$ e $p_2(0.5) = 1.25$

```
# newton-gregory
import numpy as np
def ngreg(a,b):
  t=len(a)
  r=np.zeros((t,t+1),float)
  for i in range(t):
    r[i][0]=a[i]
    r[i][1]=b[i]
  j=2
  while j<=t:
    i=t-j
    while i>=0:
      r[i][j]=r[i+1][j-1]-r[i][j-1]
      i=i-1
    j=j+1
  return r
print(ngreg([-1,0,1,2,3],[2,1,2,5,10]))
```

Aplicações práticas de interpolação

Aqui aparecem exercícios práticos de problemas reais onde a interpolação é uma necessidade. Use a técnica (interpolação polinomial, mínimos quadrados,

algoritmo de Lagrange, ou pacotes de programação (APL, Matlab, Freemat, Python, ...) que quiser e a seu critério. O único critério de correção será o resultado numérico correto.

1. e 2. Um ônibus sai de uma cidade colada a uma rodovia federal, e nela vai-se considerar o marco zero da viagem. Ao longo do tempo obtiveram-se os seguintes valores. O tempo está dado em minutos

tempo	61	72	85	92	108	114
posição	86	95	103	114	121	134

Ache a posição esperada do ônibus para o tempo de 88 usando uma reta e uma parábola. Quando for escolher os pontos a usar, (e a desprezar), elimine os mais afastados do ponto procurado.

3. Um paraquedista realizou 4 saltos, pulando de alturas distintas em cada salto. Foi testada a precisão de seus saltos em relação a um alvo de raio 5m de acordo com a altura. A distância apresentada na tabela a seguir é relativa à circunferência.

salto	altura (m)	distância ao alvo (m)
1	1475	99
2	1152	62
3	833	59
4	807	33

Se o saltador for absolutamente regular e previsível, a que distância ele cairia se saltasse de 1005 metros ?

4. Conhecendo-se o diâmetro e a resistividade de um fio cilíndrico, verificaram-se as seguintes resistências de acordo com o comprimento.

comp (m)	2090	3720	4180	5900
resist (ohms)	13	18	21	22

Usando o algoritmo de Lagrange e considerando os 4 pontos, determine qual a resistência provável para um fio de comprimento igual a 2752 metros.

5. Um procedimento que pode e deve ser feito é obter um ponto em processo de interpolação e compará-lo com o resultado real desse ponto, desde que disponível. Este procedimento qualifica o processo de interpolação (para este problema e neste entorno). No problema a seguir, dados 5 pontos, você deve interpolar os 3 pontos internos considerando em cada caso os 4 pontos restantes. Use o método de Lagrange. Seja a tabela contendo 5 pontos

	1	2	3	4	5
x	2360	3540	4270	5900	6410
$f(x)$	11	16	21	21	23

Informe qual o maior erro cometido na interpolação dos 3 pontos interiores da distribuição acima vis-a-vis os pontos reais.

6. Seja uma distribuição qualquer dada pela coleção de pontos

x	2360	3270	4720	5810
$f(x)$	119	131	145	157

Seja o valor de x a interpolar igual a 2099

Resolva este problema usando Lagrange e usando Mínimos Quadrados e relate na resposta a diferença $Lagrange - Minimosquadrados$.

7. Suponha uma distribuição na qual há abundância de dados, como em $x =$

```
262.3
271.7
281.1
292.8
313.9
335.0
351.4
414.6
440.4
468.5
505.9
513.0
541.1
585.6
609.0
758.9
777.7
808.1
843.2
1061.1
1440.4
1498.9
1513.0
1653.5
1698.0
1742.5
1934.6
1990.8
2002.5
2016.6
2018.9
2025.9
2072.8
2084.5
2089.2
2180.5
2393.7
2536.6
2576.4
2752.1
```

Com os correspondentes $f(x)$ iguais a

```
1267.0
1274.1
1290.5
1313.9
1323.2
1330.3
1335.0
1346.7
```

```
1372.4
1395.9
1400.5
1433.3
1438.0
1466.1
1468.5
1498.9
1597.3
1658.2
1684.0
1698.0
1766.0
1794.1
1836.2
1866.7
1908.8
1941.6
1962.7
1976.8
1986.1
2033.0
2187.6
2187.6
2250.8
2283.6
2349.2
2363.2
2440.5
2569.4
2599.8
2660.7
```

Sendo que o ponto a interpolar é aquele de $x = 661.3$

Descreva aqui o que você deveria fazer sobre estes dados para torná-los "manuseáveis" pelos algoritmos de interpolação.

Agora, aplique a estratégia que você descreveu sobre o problema acima e calcule o valor correspondente a $f(x)$:

Responda os números pedidos aqui:

1	2	3
4	5	6

Respostas
1) 107.71
2) 108.31
3) 110.47
4) 12.92
5) 4.90
6) -11.15
7) a gosto do freguês

Interpolação - Prática

Estes exercícios estão baseados no material da Unesp - Bauru, Faculdade de Ciências, dos autores Cherri, A; Vianna, A; Balbo, A e Baptista, E. Disponível em maio/19 em `wwwp.fc.unesp.br/~adriana/Numerico/Interpolacao.pdf`

Exercícios

1. Calcule um valor aproximado para cosseno de 0.9 utilizando a fórmula de Lagrange para os pontos -1, 0, 1 e 2. Sabe-se que cos(-1)=0.5403023059; cos(0)=1; cos(1)=0.5403023059 e cos(2)=-0.4161468365. Note que este exercício é apenas didático (já que a função cosseno é conhecida com a aproximação desejada em qualquer ponto do seu domínio). Aproveite para comparar o resultado achado na interpolação vis-a-vis o resultado dado pela função cosseno original. Lembre que a unidade de ângulo aqui é o radiano.

2. Considere a função $y = f(x)$ definida pela tabela

x	-2	0	1	2	3
2.1	2.8	-3	-1.7	2.5	

Usando a interpolação polinomial de 5 pontos, ou então o algoritmo de Lagrange (tanto faz) calcule um valor aproximado para 0.215

3. As densidades do sódio para três temperaturas são dadas a seguir:

i	0	1	2
Temperatura T_i	94.4^oC	205.6^oC	371.2^oC
Densidade ρ_i	$928kg/m^3$	$906kg/m^3$	$863kg/m^3$

Utilizando a Fórmula da Interpolação de Lagrange, estime o valor aproximado da densidade para T= 243^oC

4. Um paraquedista realizou seis saltos, saltando de altura distinta a cada salto. Testou-se a precisão de seus saltos em relação a um alvo de 5m de raio, de acordo com a altura. A distância apresentada na tabela abaixo é relativa à circunferência.

alt	1508m	1247m	1011m	754m	531m
dist. do alvo	34m	27m	16m	12m	9m

Levando em consideração dos dados acima, a que provável distância do alvo o paraquedista cairia se saltasse de uma altura de 950 metros ?

5. Um veículo de fabricação nacional, após vários testes, apresentou os resultados a seguir, quando se analisou o consumo de combustível de acordo com a velocidade média imposta ao veículo. Os testes foram realizados em rodovia em operação normal de tráfego, numa distância de 76 km.

velocidade (km/h)	56	67	85	96	119
consumo (km/l)	14.45	13.67	13.54	12.58	11.71

Verifique o consumo aproximado no caso da velocidade a usar seja de 83 km/h.

6. Seja $f(x)$ dada na forma tabular

x	0.22	0.37	0.41	0.53	0.63	0.75
$f(x)$	0.15	0.24	0.27	0.35	0.36	0.37

Obtenha $f(0.49)$ usando um polinômio de grau 3. Lembre que para obter um polinômio de grau 3 usam-se 4 pontos. Como há 6 pontos, você deve sacrificar 2 deles. Em casos reais, use a estratégia que fizer mais sentido para você, mas neste exercício despreze os 2 extremos.

7. Dada a tabela

x	0	0.1	0.2	0.3	0.4	0.5
$f(x)$	1	1.1052	1.2214	1.3499	1.4918	1.6487

Obtenha x, tal que $e^x = 2.259$

8. Construa uma tabela para a função $f(x) = sen(x)$ usando os pontos 0.8, 0.9, 1.0, 1.1, 1.2 e 1.3. Estime o valor de $sen(1.178)$ usando um polinômio de 3^o grau. Como só podem ser usados 4 pontos, desconsidere os extremos acima.

9. Durante três dias consecutivos foi tomada a temperatura em ^{o}C em Curitiba, por 4 vezes no período das 6 às 12 horas. Determine, usando todos os dados abaixo, a média das temperaturas nos três dias às 09h00.

dia	06h00	08h00	10h00	12h00
1	16.4	19.4	21.1	22
2	17	18.6	22.7	23.8
3	18	20.2	20.8	23.6

10. Um automóvel percorreu 160 km numa rodovia que liga Biguaçu a Joinville. Gastou nesse trajeto 2 horas e 20 minutos. A tabela abaixo dá o tempo gasto e a distância percorrida entre as 2 cidades.

tempo	0	8	32	64	93	120	140
distância	0	10	2761	104	144	160	

Determine:

1. Qual foi aproximadamente à distância percorrida pelo automóvel nos primeiros 45 minutos de viagem, considerando apenas os quatro primeiros pontos da tabela?

2. Quantos minutos o automóvel gastou para chegar à metade do caminho?

11. Construa a tabela de $log_{10}(x)$, usando 6 pontos igualmente espaçados de tal forma que $x_0 = 2$ e $x_5 = 3$. Usando a interpolação, determine o valor aproximada de x tal que $log_{10}(x) = 0.418$.

12. Na tabela abaixo está assinalado o número de habitantes (fictícios) de Londrina nos censos de 1950, 1960, 1970 e 1980.

Ano	1950	1960	1970	1980
Habitantes	111371	212567	504997	972409

Determine o número aproximado de habitantes de Londrina em 1977.

13. Seja a função

$$\frac{2sen^2x}{x+1}$$

Utilizando apenas os dados da tabela abaixo

i	x_i	$sen(x_i)$
0	0	0.00
1	$\pi/6$	0.5
2	$\pi/4$	0.7071067812
3	$\pi/3$	0.8660254038
4	$\pi/2$	1.0

Determine agora:

1. $f(\frac{\pi}{5})$

2. $f(\frac{11\pi}{21})$

14. Use os valores de $e^{0.0}$, $e^{0.2}$ e $e^{0.4}$ para determinar o valor aproximado de $e^{0.14}$.

1	2	3	4	5	6	7	8
9	10.1	10.2	11	12	13.1	13.1	14

Respostas

1) 0.6
2) 1.1
3) 897.4
4) 14.3
5) 13.6
6) 0.3
7) 0.7
8) 0.9
9) 20.5
10) 39.2 e 70.5
11) 2.6
12) 814779
13) 0.6 e 1.0
14) 1.1

Capítulo 11

Ajustes de curvas

Ao estudar um fenômeno, pode-se propor um modelo matemático. Nele, definem-se

- variável resposta (ou variável dependente) – é aquela que é respondida pelo modelo, ou seja, é a variável de saída do modelo.
- variável explicativa (ou independente) – que é aquela exigida pelo modelo ou seja é a variável de entrada.

Para estudar melhor o fenômeno, deve-se construir um gráfico $x0y$ conhecido como gráfico de dispersão, para verificar visualmente como os dados se distribuem. É bom lembrar que a decisão entre quem é entrada e quem é saída é uma decisão puramente arbitrária. A matemática "lava as mãos" nessa hora. Para ela tanto faz se *o cachorro abana o rabo ou ao contrário é o rabo que abana o cachorro.*

Este procedimento é parecido com a interpolação, mas lá havia a preocupação de analisar os dados desconhecidos apenas entre os limites obtidos na experimentação (senão seria extrapolação). Aqui não: no ajuste de curvas busca-se a expressão de uma curva que se extende ao infinito no sentido matemático. Ruggiero [Rug88] diz textualmente *"para a extrapolação (pesquisa fora dos intervalos conhecidos) indica-se o ajuste de curvas, já que a interpolação não é recomendada aqui".*

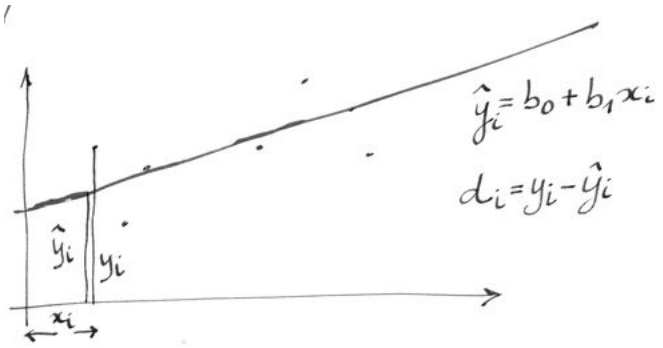
Dito de outra maneira, na interpolação privilegia-se uma visão local (próxima aos pontos coletados) enquanto no ajuste de curvas o enfoque é uma visão mais global do fenômeno enm estudo.

Isto fica evidente no seguinte fato: se o ajuste de curvas se fizer com p pontos gerando um polinômio de grau $p-1$ este polinômio é para ser idêntico ao obtido pela interpolação.

Falando sobre o ajuste de curvas Chapra [Cha13, pág. 321] diz *"Quando os dados exibirem grau significativo de erro ou "ruido" a estratégia será encontrar uma única curva quer represente a tendência geral dos dados. Como cada ponto individual pode estar incorreto, não será feito nenhum esforço para passar a curva por todos os pontos. Em vez disso, a curva é escolhida para seguir o padrão dos pontos considerados como um grupo. Uma abordagem dessa natureza é chamada de regressão por mínimos quadrados.*

11.0.1 Regressão linear: mínimos quadrados

Veja-se um exemplo de um gráfico com alguns pontos:



Cada ponto no gráfico tem as suas coordenadas x e y e buscou-se uma reta ($y = b_0 + b_1.x$) que busque reproduzir o comportamento da distribuição dos pontos examinados. Cada ponto tem sua ordenada y real (y_i) e a ordenada correspondente $\hat{y}_i = b_0 + b_1 x_i$.

Para cada ponto i, pode-se calcular a distância entre y_i e $\hat{y}_i$ através da expressão $d+i = y_i - \hat{y}_i$. A soma das distâncias de todos os pontos levantados é

$$D = \sum_{j=1}^{n} d_i$$

D é a somatória das diferenças entre o valor real de y e aquele dado pela reta $y = b_0 + b_i.x$. Como D pode ser para mais ou para menos, o somatório simples pode mascarar os erros e não serve. Uma possível solução seria tomar o módulo ($|d_i|$) que resolveria o problema do sinal, mas criaria outro: devido a suas decontinuidades o módulo não tem uma boa derivada.

Resta tomar o quadrado das diferenças (d_i^2), que resolve os 2 problemas. Daqui, vem o nome do método que o dos "mínimos quadrados".

Então,

$$D = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

. Substituindo d_i^2 por $(y_i - \hat{y}_i)^2$, chamado D de $D(b_0, b_1)$ que é a reta determinada por b_o e b_1 e substituindo $\hat{y}_i$ por $b_0 - b_1 x_i$ fica

$$D(b_0, b_1) = \sum_{i=1}^{n} (y_i - b_0 - b_1 x_i)^2$$

Derivando esta equação em relação a b_0 fica:

$$\frac{\partial D}{\partial b_0} = -2 \sum_{i=1}^{n} (y_i - b_0 - b_1.x_i)$$

Derivando a mesma equação em relação a b_1 fica:

$$\frac{\partial D}{\partial b_1} = -2 \sum_{i=1}^{n} (y_i - b_0 - b_1.x_i).x_i$$

Os valores de b_0 e b_1 em que a função $D(b_0, b_1)$ apresenta um ponto de mínimo são obtidos – as usual – igualando-se as derivadas a zero. Fica: $-2 \sum_{i=1}^{n} (y_i - b_0 - b_1.x_i) = 0$ e $-2 \sum_{i=1}^{n} (y_i - b_0 - b_1.x_i).x_i = 0$. Daí, $\sum y_i - \sum b_0 - \sum b_1.x_i = 0$ e $\sum x_i y_i - \sum b_0 x_i - \sum b_1.x_i^2 = 0$. Chamando $\sum$ de S fica:

$Sy_i - Sb_0 - Sb_1 x_i = 0$ e
$Sy_i x_i - Sb_0 x_i - Sb_1 x_i^2 = 0$. Reescrevendo
$Sy_i = nb_0 + Sx_i b_1$ e
$Sy_i x_i = Sx_i.b_0 + Sx_i^2 b_1$ que é um sistema linear. Na sua forma matricial

$$\begin{bmatrix} n & Sx_i \\ Sx_i & Sx_i^2 \end{bmatrix} \cdot \begin{Bmatrix} b_0 \\ b_1 \end{Bmatrix} = \begin{Bmatrix} Sy_i \\ Sy_i x_i \end{Bmatrix}$$

O objetivo final é obter b_0 e b_1 que, lembrando são os coeficientes da reta procurada ($\hat{y}_i = b_o + b_1 x_i$). A solução deste sistema pelo método de escalonamento de Gauss é

$$b_1 = \frac{n.\sum x_i y_i - \sum x_i \sum y_i}{n.\sum x_i^2 - (\sum x_i)^2}$$

e

$$b_0 = \frac{\sum y_i - (\sum x_i).b_1}{n}$$

Obtidos estes parâmetros pode-se escrever a "menor" reta como

$$Y = b_0 + b_1 X$$

Seja um exemplo:

i	1	2	3	4	5
x_i	1.3	3.4	5.1	6.8	8.0
y_i	2.0	5.2	3.8	6.1	5.8

Calculando:
$n = 5$
$\sum x_i = ... = 24.6$
$\sum x_i^2 = (1.3)^2 + (3.4)^2 + ... = 149.50$
$\sum y_i = 22.9$
$\sum x_i y_i = (1.3 \times 2.0) + (3.4 \times 5.2) + ... = 127.54$

$$b_1 = \frac{5 \times 127.54 - 24.6 \times 22.9}{5 \times 149.50 - 24.6^2} = 0.522$$

e

$$b_0 = \frac{22.9 - (24.6 \times 0.522)}{5} = 2.01$$

e daqui

$$Y = 2.01 + 0.522X$$

```
# minquad
def minquad(x,y):
    import numpy as np
    n=len(x)
    sy=0
    sx=0
    sx2=0
    sxy=0
    sy2=0
    i=0
    while i<n:
       sx=sx+x[i]
```

```
        sy=sy+y[i]
        sx2=sx2+(x[i]**2)
        sxy=sxy+(x[i]*y[i])
        sy2=sy2+y[i]**2
        i=i+1
    print('sy ',sy)
    print('sx ',sx)
    print('sx2 ',sx2)
    print('sxy ',sxy)
    print('n ',n)
    print('sy2 ',sy2)
    b1=((n*sxy)-(sy*sx))/((sx2*n)-(sx**2))
    b0=(sy-(sx*b1))/n
    print('coefs da linha ',b0,'  ',b1)
    num=(sxy-((sx*sy)/n))**2
    den=((sx2-((sx**2)/n))*(sy2-((sy**2)/n)))
    print("coef de determinação = ",num/den)
    while (1==1):
        xx=float(input("Informe x (-1 para fim)"))
        if xx==-1:
            return
        yy=b0+(b1*xx)
        print("y = ",yy)
#para chamar faca: x=[1.3,3.4,5.1,6.8,8.0] e
# y=[2.0, 5.2, 3.8, 6.1, 5.8]
#depois: minquad(x,y) e informe quantos x_i
# quiser, seguido de -1
```

Coeficiente de determinação

mede a qualidade do ajuste linear, isto é quão próxima é a reta determinada em relação ao conjunto de pontos. É um valor (R^2) que varia entre zero e um ($0 \leq R^2 \leq 1$). Quanto mais perto de 1, melhor o ajuste. A fórmula é

$$R^2 = \frac{[\sum xy - ((\sum x \sum y)/n)]^2}{[\sum x^2 - ((\sum(x))^2/n)].[\sum y^2 - ((\sum(y))^2/n)]}$$

no exemplo acima, o número que faltou calcular é $\sum y^2 = 116.33$ e fica:

$$R^2 = \frac{[127.54 - 24.6 \times 22.9/5]^2}{149.5 - 24.6^2/5] \times [116.33 - 22.9^2/5]}$$

ou $R^2 = 0.679$.

11.0.2 Regressão múltipla

Quando há apenas 2 variáveis em análise diz-se "regressão simples"e quando há mais do que 2 variáveis envolvidas, diz-se "regressão múltipla".

Na fórmula original tem-se

$$y = Xb + \epsilon$$

, que deve ser lido como o parâmetro matricial b é o que ajusta os y e os x observados.

A fórmula de cálculo de b é $b = (X'X) -1X'y$, onde X' é a matriz transposta de X é X^{-1} é a matriz inversa de X.

Acompanhe o exemplo

Suponhamos querer estudar o comportamento do consumo de bens em uma determinada família, sujeita a dois parâmetros:

- renda e
- taxa de juros

Dos estudos de econometria, sabe-se que o consumo aumenta em função da renda e diminui em função da taxa de juros.

O trabalho começa pelo levantamento de dados prévios e obteve-se a seguinte tabela:

medida	consumo y	renda x_1	tx juros x_2
1	122	139	11,5%
2	114	126	12,0%
3	86	90	10,5%
4	134	144	9,0%
5	146	163	10,0%
6	107	136	12,0%
7	68	61	10,5%
8	117	62	8,0%
9	71	41	10,0%
10	98	120	11,5%

A matriz X do nosso problema é:

```
1  139  0.115
1  126  0.12
1   90  0.105
1  144  0.09
1  163  0.1
1  136  0.12
1   61  0.105
1   62  0.08
1   41  0.1
1  120  0.115
```

Note que a primeira coluna sempre é formada de unidades e as demais colunas são as variáveis em estudo (no caso: renda e taxa de juros).

Tomando a matriz transposta fica-se com

```
  1     1    1    1   1    1    1    1   1    1
139   126   90  144 163  136   61   62  41  120
0.11 0.12 0.10 0.09 0.1 0.12 0.10 0.08 0.1 0.11
```

Multiplicando uma pela outra, tem-se

```
  10      1082     1.05
1082    132744   115.4
   1.05    115.4   0.1118
```

E a matriz inversa desta é

```
7.225436859   0.00095983853 -68.85039418
0.00095983853 0.00007350542  -0.084886905
-68.85039418 -0.08488690598 743.1919753
```

Agora, multiplicando a inversa por X' fica:

```
-.5 -.91  .08  1.16   .49 -.90 .05  1.7  .3 -.5
 .0  .00 -.00   .00   .00  .0 -.00  -.0 -.0  .0
4.8 9.63 1.54 -14.1 -8.36 8.7 4.0 -14.6 1.9 6.4
```

Agora multiplicando esta matriz pela matriz coluna de y (consumo) que é (devidamente horizontalizada para economizar papel)

```
122  114   86  134  146  107   68  117   71   98
```

Fica

```
  148.5220194
    0.6135933813
-1034.40784
```

Interpretação

Neste problema, o consumo é

$$Consumo =$$

$$148,52 + 0.6135 \times Renda - 1.034.41 \times TxJuros$$

Ou seja, tirando a parte do Consumo que não é influenciada pela Taxa de Juros, o incremento de $ 1 na Renda causa um incremento esperado de $ 0,6136 no Consumo; além disso, o incremento de 1 ponto percentual (0,01) na Taxa de Juros causa um decréscimo esperado de $ 10,3441 no Consumo.

Observação importante: No Python, ao gerar as matrizes, não esquecer de pedir campos float, porque senão ele pode resolver usando inteiros e vai achar respostas erradas...

Capítulo 12

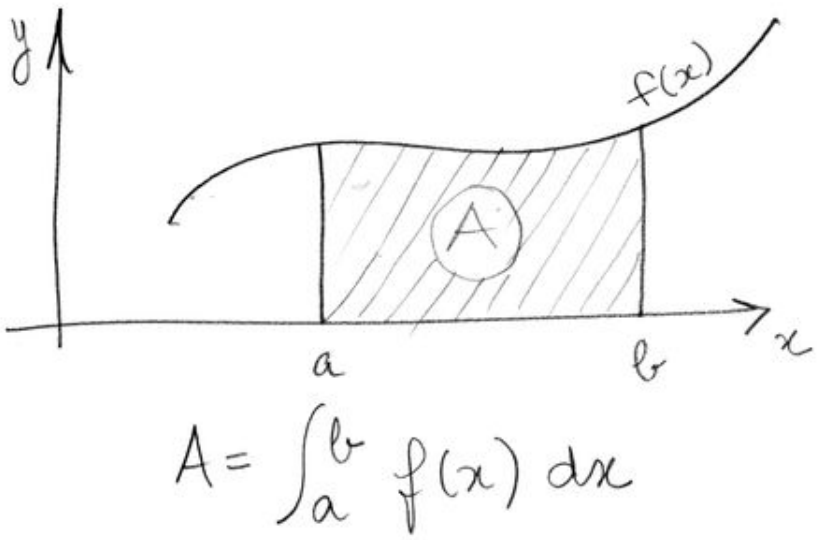
Integração

12.1 Integração Numérica

Se $f(x)$ é contínua em $[a, b]$ e sua primitiva $F(x)$ é conhecida, então

$$\int_a^b f(x)dx = F(a) - F(b)$$

A interpretação geométrica da integral é



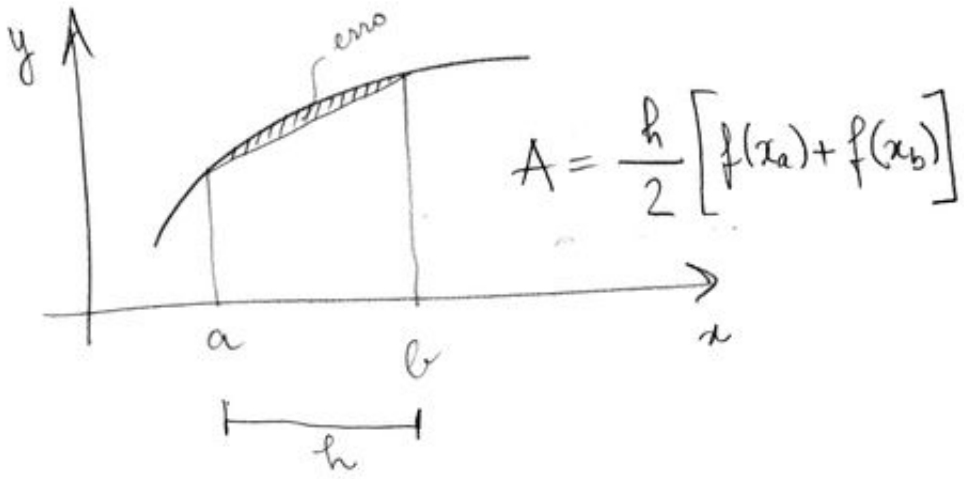
Esta análise pode ter algum problema quando:

- $F(x)$ não é conhecida, ou
- $F(x)$ é muito complexa, ou
- só se tem pares de pontos com seus valores

Nestes casos, pode-se lançar mão de métodos numéricos que são conhecidos como **quadraturas**.

Primeira abordagem: usando uma reta

Conhecido como método dos Trapézios, ele sugere obter a reta que passa por $f(a)$ e $f(b)$ como uma aproximação para a área investigada. Neste caso, são usados apenas 2 pontos (que como se sabe determinam uma reta).



Vejamos um exemplo, relembrando um pouco de cálculo: Se $f(x) = ln(x)$ então $f'(x) = \frac{1}{x}$. Daqui, se $f(x) = \frac{1}{x}$ então $\int f(x) = F(x) = ln|x| + C$. Seja calcular

$$I = \int_3^{3.6} \frac{dx}{x}$$

1. Pela regra do trapézio $I = \frac{h}{2}(y_0 + y_1)$ como $y = \frac{1}{x}$ então $y_0 = \frac{1}{x_0} = \frac{1}{3} = 0.3333333333$ e $y_1 = \frac{1}{x_1} = \frac{1}{3.6} = 0.2777777778$ e $h = x_1 - x_0 = 3.6 - 3 = 0.6$ e daí

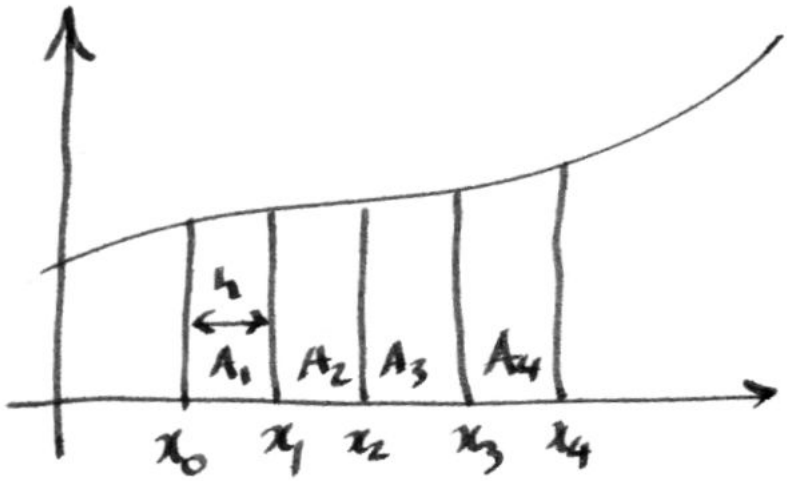
$$I = \frac{0.6}{2}(\frac{1}{3} + \frac{1}{3.6}) = 0.18333$$

2. Pelo cálculo integral

$$\int_3^{3.6} = \frac{1}{x}dx = ln(3.6) - ln(3) = 0.18232$$

12.2 Trapézio

Ao invés de usar um único trapézio, a idéia agora é dividir o intervalo $[a, b]$ em sub-intervalos e aplicar a mesma regra a cada sub-intervalo.



Agora

$$A_1 = \frac{h}{2}[f(x_0) + f(x_1)]$$

,

$$A_2 = \frac{h}{2}[f(x_1) + f(x_2)]...$$

e

$$A_n = \frac{h}{2}[f(x_{n-1}) + f(x_n)]$$

ou juntando tudo

$$A = \frac{h}{2}[f(x_0) + 2f(x_1) + ... + 2f(x_{n-1}) + f(x_n)]$$

Usando o mesmo exemplo acima $\int_3^{3.6} \frac{dx}{x}$ e usando 6 intervalos

i	x_i	y_i
0	3	0.333
1	3.1	0.322
2	3.2	0.312
3	3.3	0.303
4	3.4	0.294
5	3.5	0.285
6	3.6	0.277

$A = 0.182350$ Daqui obtem-se a primeira função:

```
def f(x):
  return 1/x
def inttrap(a,b,ntrap):
  h=(b-a)/ntrap
  area=f(a)
  i=1
  while (i<ntrap):
    a=a+h
    area=area+2*f(a)
    i=i+1
```

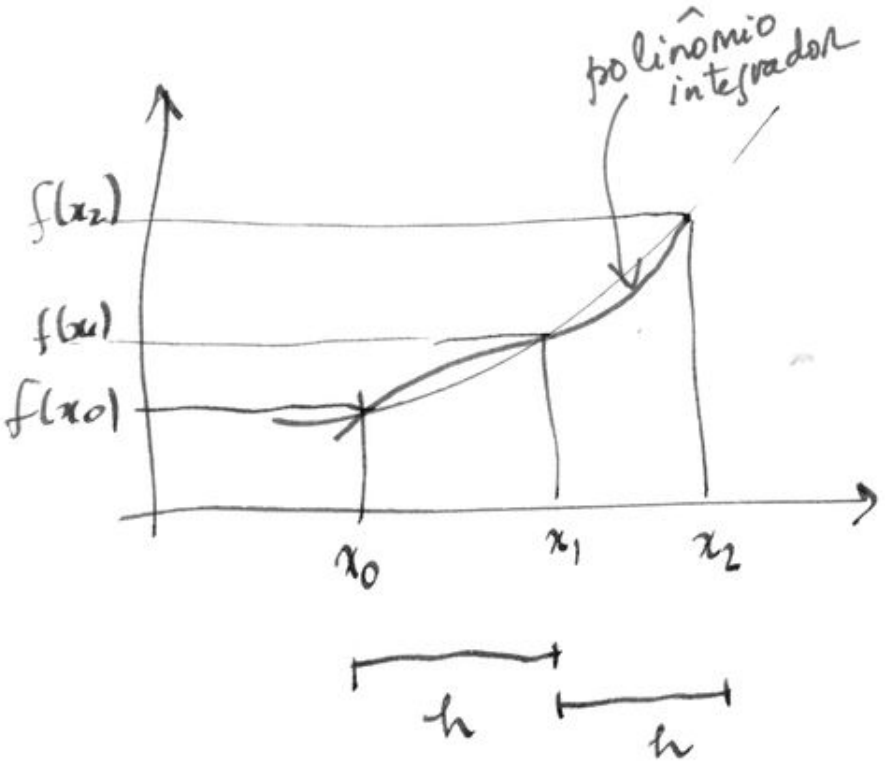
```
  a=a+h
  area=area+f(a)
  return area*h/2
print(inttrap(3,3.6,6))
```

A mesma função em C++:

```
#include <iostream>
using namespace std;
double fobj(double a)
   return 1/a;

int main()
   double a,b,area,h;
   int i,ntrap;
   cout << "Informe o limite inferior " << endl;
   cin >> a;
   cout << "Informe o limite superior " << endl;
   cin >> b;
   cout << "Informe o número de trapezios " << endl;
   cin >> ntrap;
   h = (b-a) / ntrap;
   area = fobj(a);
   cout << a << fobj(a);
   for (i=1;i<ntrap;i++)
      a=a+h;
      area=area+2*fobj(a);
      cout<< a << " - " << fobj(a) << endl;

   a=a+h;
   area=area+fobj(a);
   area=area*h/2;
   cout<<"Area = " << area << endl;
   return(0);
```

12.3 Simpson

Este método é chamado de Regra $\frac{1}{3}$ de Simpson



Agora

$$A = \frac{h}{3}[f(x_0) + 4.f(x_1) + f(x_2)]$$

Melhorando: vários intervalos

A generalização da Regra de $\frac{1}{3}$ de Simpson exige um número par de intervalos. Sua fórmula:

$$A = \frac{h}{3}[f(x_0) + 4.f(x_1) + 2.f(x_2) + ... + 2.f(x_{n-1}) + f(x_n)]$$

Função em Python

```
def f(x):
  return 1/x
def intsimpg(a,b,ntrap):
  h=(b-a)/ntrap
  if (0!=ntrap%2):
    print('ntrap nao e par, saindo')
    return
  area=f(a)
  i=1
  while i<ntrap:
    a=a+h
    if (0==i%2):
      area=area+2*f(a)
    else:
      area=area+4*f(a)
```

```
    i=i+1
  a=a+h
  area=area+f(a)
  area=area*h/3
  return(area)
print(intsimpg(3,3.6,6))
```

Uma observação final

Embora não faça parte do curriculum oficial da disciplina, caso o pacote de computação simbólica do Python (o sympy) tenha sido estudado, ele pode ser usado para resolver integrações. No exemplo acima, ter-se-ia que fazer:

```
# para comparar com a integração via sympy
import sympy as sp
import numpy as np
sp.init_session()
y=x*E**x
t=integrate(y,x) #----> para ver a integral OU
t=integrate(y,(x,5,6))
t.n() #--> para ver o valor numérico
# a solução original 1423.8941270517853 (c/20)
# o sympy deu        1423.49133105337
```

12.3.1 Uma lista de exercícios

Aplicações práticas de Integração numérica

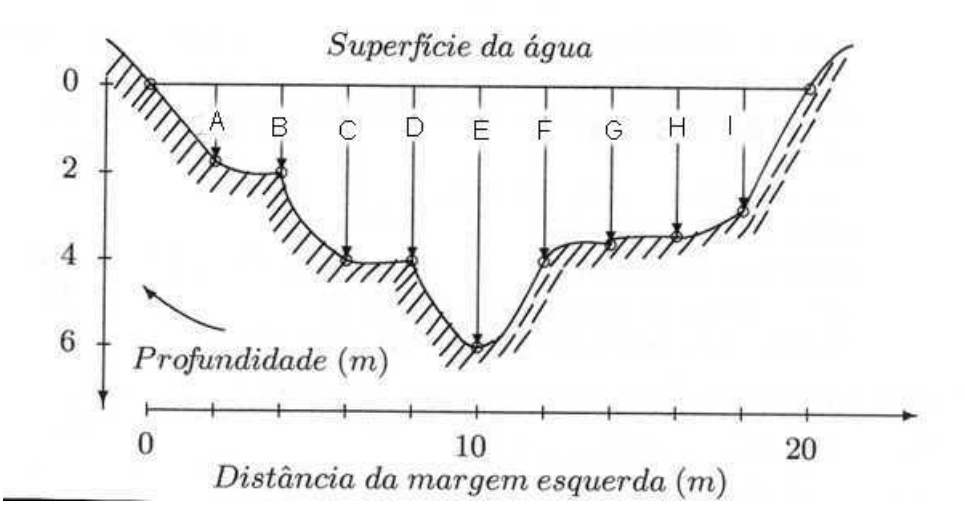
Neste exercício você deverá calcular diversas integrais de forma numérica usando os algoritmos do cálculo diferencial e integral (quando disponíveis) e também usando retas (regra do trapézio), vários trapézios e também usando polinômios do segundo grau (ou a regra de 1/3 de Simpson). Se quiser use o pacote `SymPy` do Python para responder (ou conferir a resposta) de cada exercício.

1. Calcule

$$\int_a^b xe^{x/2}dx$$

sabendo que a=9 e b=14.

2. A determinação da área da sessão reta de rios e lagos é importante em projetos de prevenção de enchentes (para o cálculo de vazão da água) e nos projetos de reservatórios (para o cálculo do volume total de água). A menos que dispositivos tipo sonar sejam usados na obtenção do perfil do fundo de rios e lagos, o engenheiro deve trabalhar com valores de profundidade obtidos em pontos discretos da superfície. Um exemplo típico de seção reta de um rio é mostrado na figura a seguir



Use os seguintes valores de profundidade: A=1.4, B=1.7, C=4.2, D=4.3, E=5.6, F=4.5, G=3.7, H=3.6, I=2.5

Use uma fórmula de quadratura sobre pontos igualmente espaçados de h para calcular a área da seção reta do rio. [Adaptado de Fra07, pág. 371]

3. Do velocímetro de um automóvel foram obtidas as seguintes leituras de velocidade instantânea

t(min)	0	5	10	15	20	25	30	35	40
v (km/h)	22	26	32	37	40	45	49	53	62

Calcule a distância em quilômetros, percorrida pelo automóvel usando a regra do trapézio generalizada.

4. calcule a mesma distância do exercício anterior pela regra de 1/3 generalizada de Simpson. Compare os 2 valores obtidos.

5. Para a duplicação de uma avenida, um estudo de engenharia de transportes necessita do cálculo do número total de veículos que passam por ela em um período de 24 horas. Um engenheiro (você) vai ao local várias vezes durante um período de 24 horas e conta o número total de carros, por minuto, que passam pela avenida. Os dados são

Hora	Carros/minuto
0:00	4
2:00	1
4:00	0
5:00	2
6:00	5
7:00	11
8:00	23
9:00	15
10:30	5
11:30	12
12:30	17
14:00	6
16:00	8
17:00	21
18:00	21
19:00	11
20:00	14
21:00	11
22:00	5
23:00	7
00:00	4

Usando uma fórmula de quadratura de trapézios generalizada, estime o número de carros que passam pela avenida a cada 24 horas.

6. A função de Debye é encontrada em Termodinâmica estatística no cálculo do calor específico da água a volume constante de certas substâncias. A função é expressa por

$$D(x) = \frac{3}{x^3}\int_0^x \frac{y^3}{e^y - 1}dy$$

Ache $D(x)$ com erro relativo $< 10^{-2}$ para $x = 3.9$

7. Um paraquedista pulando de um avião a 6000m de altura adquire uma velocidade em queda livre em função do tempo que é dada pela equação

$$v(t) = \frac{gm}{c}(1 - e^{-\frac{c}{m}t})$$

onde g é a aceleração da gravidade (igual a $9.8m/s^2$), m é a massa do paraquedista (igual a 68) e c é o coeficiente de arrasto (igual a $12.5kg/s$) e t é o tempo em *seg* a partir do início da queda. Sabendo que o espaço percorrido pelo paraquedista entre os instantes de tempo a e b é dado por

$$s = \int_a^b v(t)dt$$

calcule a altura em relação ao chão em que o paraquedista se encontra no instante de tempo t= 15

responda aqui:

1	2	3	4
5	6	7	
			xxx xxx xxx xxx

Respostas:
1) 25058.97
2) 59.10
3) 27.000
4) 26.944
5) 624960
6) 0.190
7) 2519.580

Capítulo 13

Bibliografia

Alm21 ALMEIDA, Paulo; PIMENTEL, Andrey; DELGADO, Armando; GOMES, David; SOUZA, Giovanni; DERENIEVICZ, Guilherme; MADRUGA, Leôncio; BELLON, Olga; PORTELA, Paulo e BEPPLER, Tamy. **Notas de Aula de Cálculo Numérico**. UFPR, Curitiba, 2021.

Are08 ARENALES, Selma e DAREZZO, Artur. **Cálculo Numérico - aprendizagem com apoio de software**. Thomson, São Paulo, 2008.

Bar87 BARROSO, Leônidas et alli. **Cálculo Numérico (com aplicações)**. Harbra, São Paulo, 1987.

Cha13 CHAPRA, Steven C. **Métodos Numéricos aplicados com MATLAB para engenheiros e cientistas**. Bookman, Porto Alegre, 2013.

Fra07 FRANCO, Neide. **Cálculo Numérico**. Pearson, São Paulo, 2007.

Jus18 JUSTO, Dagoberto et alli. **Cálculo Numérico - um livro colaborativo - versão Python**. Disponível em www.ufrgs.br/numerico. s/e, s/c, 2018.

Kaw17 KAW, A e KALU, E. **Numerical Methods with applications**. Disponível em nm.mathforcollege.com. s/e, s/c, 2017.

Pac77 PACITTI, Tércio e ATKINSON, Cyril. **Programação e Métodos Computacionais**. Vol 2. LTC, Rio de Janeiro, 1977.

Pet18 PETERS, Sérgio e SZEREMETA, Júlio. **Cálculo Numérico computacional**. UFSC, Florianópolis, 2018.

Rug88 RUGGIERO, Márcia e LOPES, Vera. **Cálculo Numérico - aspectos teóricos e computacionais**. Pearson, São Paulo, 1988.

San07 SANCHES, Ivonildo e FURLAN, Diógenes. **Métodos Numéricos**. UFPR, Curitiba, 2007.

Índice Remissivo

e^x, 26

achando derivada, 100
adin falcoff, 38
adição, 47
ajuste de curvas, 163
aleatório, 62
alo mundo, 42
ambientes, 33
apagão, 43
apl, 38, 44
aplx, 38
arange, 59
array, 58
arredondamento, 28
aspas, 44
atribuição, 43

bibliografia, 179
binary128, 22
binary16, 22
binary32, 22
binary64, 22
bissecção, 91
break, 53
broadcasting, 39

C++, 44
c++, 35
C++: integração, 172
claude shanon, 12
codeblocks, 35
coeficiente de determinação, 166
começo, 57
complemento para 2, 20
concatenação, 62
condicional, 51
consulta rápida, 37
continue, 54
conversão de bases, 11
cordas, 95
critério das linhas, 130
critério de sassenfeld, 130

default, 57
derivada, 65
derivada automática, 70, 100
desempenho, 58
desvio padrão, 64
determinante, 61
determinante eficiente, 117
dialog apl, 38
diff, 66
distbelief, 42
distribuição binomial, 63
distribuição normal, 63
divisão inteira, 47
divisão real, 47
dois pontos, 52
dsolve, 67

else, 52
engano, 30
engenharia elétrica, 37
equação, 65
erro, 25
escalonamento, 107
expand, 66
expressão aritmética, 47

factor, 66
falsa posição, 95
fatiamento, 57
final, 57
float, 45, 46, 50
flutuante, 20
for, 55
fortran, 34
fracionário, 15
freelab, 38
freeware, 41

gauss-jacobi, 126
gauss-jordan, 112

Gauss-Jordan - matriz inversa, 124
gauss-seidl, 130
gedit, 37
gregory, 153

hexadecimal, 12
histograma, 81

IEEE754, 21
if, 51
incremento, 57
indentação, 52
input, 50
int, 46, 50
integral, 65, 66
integrate, 66
integração, 169
integração numérica, 169
inteiro, 19
inteiro em C++, 20
interpolação, 143
interpolação polinomial, 143
Ipython, 37
iteração linear, 96

jacobi, 126
java, 44

keneth iverson, 38

lagramge: esquema prático, 147
lagrange, 146
licença estudantil, 37
limite, 65
linsolve, 67
loop, 53
LU, 113
LU, condição de existência, 114
LU, processo de obtenção, 114

mainstream, 41
mais, 47
mantissa, 21
maple, 37
matlab, 38
matplotlib, 36
matriz, 58
matriz inversa, 118
menos, 47
multiplataforma, 41
multiplicação, 47
murphy, 30
método de gauss, 107
módulo, 47

naive, 107
napl, 38
nars apl, 38
NC da UFPR, 34
negativo, 20
newton, 149
newton-gregory, 153
newton-raphson, 99
nonlinsolve, 67
normalização, 22
numpy, 36, 45
número complexo, 109
números em computadores, 19

octave, 38
operação elementar com linhas, 107
ordem das incógnitas, 111
overflow, 21

palavra reservada, 45
parênteses, 47
pascal, 44
pdf, 52
PHP, 44
pie, 80
pivoteamento parcial, 109
pizza, 80
polarização, 21
precisão ilimitada, 46
propagação de erro, 29
PYPI, 41, 75
python, 36
python2, 49
python3, 49
Python: bissecção, 92
Python: calculo de e^x, 27
Python: cordas, 96
Python: falsa posição, 96
Python: gauss (escalonamento), 108
Python: gauss-jordan, 113
Python: gauss-Seidl, 131
Python: integração, 171
Python: interpolação, 144
Python: inversa, 123
Python: jacobi, 129
Python: lagrange, 148
Python: leitura de arquivo, 141
Python: LU, 115
Python: LU alternativo, 118
Python: matriz inversa, 125

Python: mínimos quadrados, 165
Python: newton, 152
Python: newton-gregory, 155
Python: newton-raphson, 99, 101
Python: relaxação, 133
Python: simpson, 173
pythonanywhere, 37

raízes, 89
real, 20
regressão múltipla, 166
resmungar, 58
resto, 47

secante, 103
seidl, 130
short circuit, 58
simplify, 65
simpson, 173
sistemas lineares, 105
solve, 61
str, 46, 50
sublinha, 47
subpacotes, 62
subtração, 47
symbols, 64
sympify, 65
sympy, 36, 64

tautologia, 54
tela azul da morte, 43
tensorflow, 40
texnsorflow, 42
tiobe, 40, 41
tipo, 45
tolerância à falha, 30
toolbox, 35
trapézio, 170
truncamento, 28

ufunc, 39
underflow, 21
unicode, 68

variável, 43
vezes, 47

web, 37
while, 53
winpython, 36

zeros, 89

Esta página fica em branco

Resumo de Cálculo Numérico

Lembre-se: Python,Python+Sympy, APL, Matlab e Octave, C ou C++, Maple e outros.

1. Conversão de bases

Decimal: 10 símbolos: 0,1,2,3,4,5,6,7,8 e 9. Base 10, Exemplo: $347 = 3\times10^2+4\times10^1+7\times10^0 = 300+40+7 = 347$

Binário: 2 símbolos: 0 e 1. Base 2, Exemplo $11010_2 = 1\times2^4+1\times2^3+0\times2^2+1\times2^1+0\times2^0 = 16+8+0+2+0_2 = 26_{10}$

Hexadecimal: 16 símbolos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E e F. Base 16, Exemplo: $2CA_{16} = 2\times16^2+12\times16^1+10\times16^0 = 512+192+10 = 714_{10}$

de → para	processo
2 → 10	potências crescentes de 2 e soma tudo.
2 → 16	Cada 4 bits formam um dígito hexadecimal. Os valores de 0 a 9 ficam e depois A=10, B=11, C=12, D=13, E=14 e F=15
10 → 2	Divisões sucessivas por 2 até o resultado ser 0. O número é construído com os restos de trás para a frente
10 → 16	Divisões sucessivas por 16 até o resultado ser 0. O número é construído com os restos de trás para a frente
16 → 2	Cada dígito hexadecimal dá origem a 4 bits: 0=0000, 1=0001, 2=0010, 3=0011, 4=0100, 5=0101, 6=0110, 7=0111, 8=1000, 9=1001, A=1010, B=1011, C=1100, D=1101, E=1110 e F=1111
16 → 10	potências crescentes de 16 e soma tudo

Na conversão de números fracionários, usa-se a mesma regra, mas com as potências -1, -2, -3, ... Por exemplo 101.111_2 é igual a: a parte inteira é 5_{10}, e a parte fracionária é $1\times2^{-1}+1\times2^{-2}+1\times2^{-3} = 0.5+0.25+0.125 = 0.875_{10}$. Então: $101.111_2 = 5.875_{10}$.

1.1 Potências de 2

2^0	1		
2^1	2	1 b	
2^2	4		
2^3	8		
2^4	16		
2^5	32		
2^6	64		
2^7	128		
2^8	256	1 B	10^0 B
2^9	512		
2^{10}	1024	1 KB	$\approx 10^3$
2^{12}	4096		
2^{16}	65536		
2^{20}	1048576	1 MB	$\approx 10^6$
2^{30}		1 GB	$\approx 10^9$
2^{40}		1 TB	$\approx 10^{12}$
2^{50}	peta	1 PB	$\approx 10^{15}$
2^{60}	exa	1 EB	$\approx 10^{18}$
2^{70}	zeta	1 ZB	$\approx 10^{21}$
2^{80}	yotta	1 YB	$\approx 10^{24}$

2. Representação interna

2.1 Inteiros: usam a metade da capacidade para representar positivos e a outra metade (1. bit=1) para negativos. Para representar os negativos usa-se o complemento para 2. Para fazer isto:

- troque 1 por 0 e vice versa, Aqui o primeiro bit já vira 1 (negativo)
- some 1 ao resultado

Acompanhe, a representação de -6 em 4 bits: 6 positivo é 0110. Trocando, fica 1001. Somando 1 fica 1010, que é -6.

2.2 Flutuantes: Tem-se o padrão IEEE754 que estabelece:
Binary16: 1=sinal, 5=expoente e 10=mantissa
Binary32: 1=sinal, 8=expoente e 23=mantissa
Binary64: 1=sinal, 11=expoente e 52=mantissa
Binary128: 1=sinal, 15=expoente e 112=mantissa
Otimizações da norma:

- Polarização do expoente. Em 16, por exemplo ao invés de -15..15 vira 0..31
- Despreza-se o primeiro bit da mantissa (é sempre 1)
- Exceto quando o número todo é zero

3. Erros

Acurácia: viés (desvio) de erro.
Precisão: concentração no valor correto.
Quando se representa um valor em ponto flutuante: registra-se o mais próximo do número real. Por exemplo:

```
>>> 2**2 == 4
True
>>> ((5**0.5)**2) == 5
False
>>> '%2.24f'%(((5**0.5)**2)-5)
'0.000000000000000888178420'
```

4. Python

Para importar os pacotes importantes:

```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
```

Para criar variáveis

```
a = 1 # inteiro
b = 2.0 # flutuante
c = 'ola' # caracter
d = True # or False
e = np.array([[1,2,...],[...]],float)
f = np.zeros((3,2),float)
```

Para imprimir

```
print('txt',var,'txt',...)
```

Para ler valores

```
a = int(input('msg'))
```

Operações aritméticas: +, −, ∗, /, // (divisão inteira), % (resto), *abs* (módulo), ∗∗ (potência)... Operações lógicas> == (igual), ! = (diferente), >, <, >=, <=, *in* (pertence)... e os conectivos *and* (e), *or* (ou) e *not* (não).

Condicional:

```
if <condição>:
  A  # executado se True
[else:
  B  # executado se False ] (opcional)
C # executado independ. da condição
```

Repetição

```
while <condição>:
  A # executando enquanto condição é True
C # executado independ. da condição

for # é o grande interador de Python
exemplo: for i in range(10): #0,1,...,9
```

Definição de função:

```
def nome-função(parâmetros):
  A # comandos de definição
  return resultado
```

Listas:

```
a = [] # lista vazia
a.append(2) # agora a é [2]
b = [1,3,8]
len(b) é 3
```

fatiamento:

```
[começo:final:incremento]
```

Sympy

Sympy: variáveis simbólicas

```
x,y=sp.symbols('x y')
f,g=sp.symbols('f g',cls=sp.Function)
f=x**3+2*x**2+sp.exp(x)
g=sp.diff(f,x) # derivada de f em rel a x
print(g.subs(x,2)) # expressao com x=2
print(g.subs(x,2).n()) # valor numérico
```

Matplotlib

```
def pla():
  def f(x):
    return 4*math.cos(x)
  def g(x):
    return math.exp(x)
  vi,vf = 0,2
  x1=np.arange(vi,vf,((vf-vi)/100))
  y1=[0.0]*100  # idem para y2
  for i in range(100):
    y1[i]=f(x1[i]) # idem y2[i]=g(x1[i])
  plt.plot(x1,y1,'b-')
  plt.plot(x1,y2,'r-')
  plt.grid()
  plt.show()
pla()
```

5. Resolução de equações

5.1 Bissecção

Etapas:
1) arbitrar $[a,b]$ que compreenda a raiz.
2) arbitrar o erro ε.
3) Calcular $x_m = (a+b)/2$.
4) Se $f(a).f(x_m) < 0$ o novo intervalo é $[a,x_m]$, senão $[x_m,b]$
5) enquanto $\varepsilon < |a-b|$ voltar a 3.
6) A resposta é x_m

5.2 Falsa posição

O procedimento é similar à bissecção, mas

$$x_m = \frac{a.f(b)-b.f(a)}{f(b)-f(a)}$$

5.3 Iteração linear

Transforma $f(x)$ em sistema formado por $y = x$ e $y = g(x)$. A partir de x_0 (inicial) aproxima-se fazendo $x_{k+1} = g(x_k)$. Diversas alternativas de $g(x)$ podem ser escolhidas, devendo-se estudar a convergência em função de $g'(x)$

5.4 Newton-Raphson

Considera a derivada de $f(x)$ como a inclinação da tangente. Passos:
1) Derivar $f(x)$.
2) Arbitrar um valor x_0.
3) Arbitrar erro ε.
4) Calcular $f(x_0)$ e $f'(x_0)$.
5) novo $x = x_0 - \frac{f(x_0)}{f'(x_0)}$
6) Se $|f(x)| > \varepsilon$ retorne a 4.

5.5 Secante

Alternativa para não precisar calcular $f'(x)$ no método de Newton-Raphson. A derivada é substituída por $f'(x) = \frac{f(x_k)-f(x_{k-1})}{x_k-x_{k-1}}$ onde x_k e x_{k-1} são duas aproximações para a raiz. A função de interação fica $xk+1 = \frac{x_{k-1}\cdot f(x_k)-x_k\cdot f(x_{k-1})}{x(x_k)-f(x_{k-1})}$

5.6 Misto

Aplicação em sequência do método de Newton-raphson e da falsa posição, nesta ordem. Agora $x_n = \frac{x_n^{NR}+x_n^{FP}}{2}$. Garante uma convergência mais rápida.

6. Sistemas Lineares

Suponha-se o sistema $Ax = B$

6.1 Escalonamento de Gauss

Usando operações elementares em A (troca de linhas, multiplicação de linha por constante e troca de uma linha por expressões de linhas) deve-se obter uma nova matriz A (equivalente à original) onde o triângulo inferior esteja zerado. Por retrossubstituição, obtem-se x. Por exemplo, Seja $A = \begin{vmatrix} 3 & -1 & 1 \\ 1 & 2 & -1 \\ 4 & -2 & -1 \end{vmatrix}$ e $B = (11, 6, -5)$. Após rodar o algoritmo, $A = \begin{vmatrix} 3 & -1 & 1 \\ 0 & 2.33 & -1.33 \\ 0 & 0 & -2.71 \end{vmatrix}$ e $B = (11, 2.33, -19)$ e com isso $x = (3, 5, 7)$.

6.1.1 Pivoteamento parcial e total

No pivoteamento parcial as linhas são trocadas de lugar com vistas a escolher o elemento a zerar entre aqueles que maior valor absoluto. Algebricamente a operação é inócua, mas computacionalmente tende a diminuir os erros de truncamento/arredondamento.

No pivoteamento total, linhas são trocadas (como acima) mas também as colunas para escolher o elemento de maior valor absoluto entre todos os candidatos. Ao trocar as colunas, precisa-se guardar a informação de qual é o x_k modificado.

6.2 Gauss-Jordan

Similar ao método de eliminação de Gauss, mas o método prossegue até que A tenha sido transformada na matriz I (identidade). Neste ponto tem-se $x = B$.

6.3 Fatoração LU

A matriz A deve ser transformada em um produto de duas outras matrizes, $L.U = A$, sendo U a A original, mas devidamente escalonada (zeros no triângulo inferior) e L que contém a unidade na diagonal principal, zeros acima desta e os pivots usados para obter U abaixo da diagonal principal.

Se começa com o sistema $Ax = B$. Calcula-se L e U, e fica $L.Ux = B$, separando fica $Ux = y$ e finalmente $Ly = B$. Ou seja, primeiro se calcula y em $Ly = B$ e depois se calcula x em $Ux = y$. Vejamos um exemplo Seja $A = \begin{vmatrix} 3 & -4 & 1 \\ 1 & 2 & 2 \\ 4 & 0 & -3 \end{vmatrix}$

Agora,

$$L = \begin{vmatrix} 1 & 0 & 0 \\ 0.33 & 1 & 0 \\ 1.33 & 1.6 & 1 \end{vmatrix} \text{ e}$$

$U = \begin{vmatrix} 3 & -4 & 1 \\ 0 & 3.33 & 1.66 \\ 0 & 0 & -7 \end{vmatrix}$. Daqui $y = (9, 0, -14)$ e finalmente $x = (1, -1, 2)$.

6.4 Via matriz inversa

Dada uma matriz A pode existir uma matriz A^{-1} tal que $A.A^{-1} = I$ onde I é a matriz identidade. (Funciona como na aritmética. Dado um número x pode haver seu inverso $1/x$ tal que $x.1/x = 1$). Uma matriz A só tem inversa se o seu determinante é diferente de zero. Há muitas maneiras de calcular a inversa, mas pode-se usar o método de Gauss-Jordan, escrevendo $A|I$ e aplicando as operações de matrizes para transformar A em I e aplicando rigorosamente as mesmas operações no lado direito da matriz que começou com I. Ao final do processo aqui estará A^{-1}. Esquematicamente $A|I \Rightarrow I|A^{-1}$. Todos os pacotes de software estudados têm como calcular diretamente (e otimizadamente) a inversa.

A razão deste cálculo é a solução do sistema $Ax = B$ ou $x = \frac{B}{A}$ ou ainda $x = B.A^{-1}$. Uma vez calculada A^{-1} (que é mais ou menos fixa para um dado sistema ou fenômeno), múltiplas encarnações de x (uma para cada vetor B diferente) podem ser facilmente calculadas bastando multiplicar A^{-1} por B. Por exemplo, seja $A = \begin{vmatrix} 3 & 3 & 3 \\ 1 & 2 & 3 \\ 5 & 2 & 1 \end{vmatrix}$ então sua inversa é $A^{-1} = \begin{vmatrix} -0.66 & 0.5 & 0.5 \\ 2.33 & -2 & -1 \\ -1.33 & 1.5 & 0.5 \end{vmatrix}$ para resolver o sistema onde $B = (0, -4, 6)$, basta fazer $A^{-1}.B$ que dá $x = (1, 2, -3)$ que é a solução. Observação importante: note que a multiplicação matricial NÃO é comutativa, e em geral $X.Y \neq Y.X$.

6.5 Gauss-Jacobi

Método interativo que pode convergir para a solução. O critério de convergência é o de A ser diagonalmente dominante. Em palavras, o elemento da diagonal principal (em módulo) deve ser maior que a soma de todos os outros elementos da linha (em módulo).

Primeiro divide-se A e B pelo elemento da diagonal principal de A. Agora, cada variável (da d.p.) é isolada em um lado da equação, passando-se todo o resto para o outro lado. A primeira aproximação da raiz é B dividido. Daí, recalculam-se novas aproximações, a partir das equações acima, usando a aproximação anterior. Exemplo: Seja o sistema $A = \begin{vmatrix} 20 & 1 & 1 \\ 0 & 4 & 2 \\ 2 & 2 & 8 \end{vmatrix}$ com $B = 19, 5, -18$.
Dividindo pelo elemento da diagonal principal fica $A = \left[\begin{array}{ccc|c} 1 & 0.05 & 0.05 & 0.95 \\ 0 & 1 & 0.25 & 1.25 \\ 0.25 & 0.25 & 1 & -2.25 \end{array}\right]$
as equações de convergência são:
$x = 0.95 - (0.05y + 0.05z)$,
$y = 1.25 - 0.25z$ e
$z = -(2.25 + 0.25x + 0.25z)$. A primeira aproximação é $0.95, 1.25e - 2.25$ e em 9 iterações, a resposta é $x = 0.9999, 1.9999, -2.9999$ ou claramente $x = (1, 2, -3)$.

6.6 Gauss-Seidl

Derivado de Gauss-Jacobi (compartilhando o mesmo critério de convergência) acelera pois usa os valores da iteração atual e não os da anterior. Transforma A em $A = I + D + S$, onde I=inferior, D=diagonal e S=superior (o resto é zero).
Primeiro, se faz $G = -(D + I)^{-1}.S$ e
$F = (D + I)^{-1}.B$ e depois $X^{(k)} = GX^{(k-1)} + F$

6.7 Relaxação

Atribui zeros as variáveis e calcula o sistem, obtendo os resíduos. O maior resíduo é isolado, e nesta equação qual a variável de maior coeficiente. Ela é considerada a única responsável pelo erro. É recalculada e o ciclo recomeça. No mesmo sistema acima calculado, começando com $0, 0, 0$, o primeiro resíduo seria $0.95, 0, 0$. O segundo $0.95, 0, -2.487$. O terceiro seria $0.95, 1.871, -2.487$. O processo segue e em 9 interações converge para $1.0004, 1.9999e - 2.998$ claramente $1, 2, -3$

7. Interpolação

Converte uma função desconhecida ou complexa em um polinômio *mais bem comportado*.

7.1 Linear e polinomial

Na interpolação linear obtem-se uma reta $a_0 + a_1x$, onde a_0 é o coeficiente linear e a_1 o angular. Aplicando-se esta reta a dois pontos, $(x_1, y_1$ e $x_2, y_2)$, tem-se $\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \end{bmatrix} . \begin{Bmatrix} a_0 \\ a_1 \end{Bmatrix} = \begin{Bmatrix} y_1 \\ y_2 \end{Bmatrix}$ usando-se qualquer mecanismo de solução de sistemas lineares, obtem-se a_0 e a_1 e portanto a reta buscada.

Na interpolação polinomial, (parábola com 3 pontos) o sistema fica
$\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{bmatrix} . \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix} = \begin{Bmatrix} y_1 \\ y_2 \\ y_3 \end{Bmatrix}$
e o polinômio é $P(x) = a_2x^2 + a_1x + a_0$

Para graus superiores, basta generalizar a expressão acima.

7.2 Lagrange

Acompanhe: seja a função $f(x)$ conhecida nos pontos:

i	x_i	$f(x_i) = y_i$
0	0.00	1.000
1	0.10	2.001
2	0.30	4.081
3	0.60	8.296

Determinar o valor para $f(0.20)$ aplicando o esquema prático de Lagrange

$$P_n(x) = \sum_{i=0}^{n} y_i . \prod_{j=0, j\neq i}^{n} \frac{(x - x_j)}{(x_i - x_j)}$$

ou em termos mais amigáveis:

$$P(x) = y_0 . \frac{\frac{Prod_x}{Dif_o}}{Prod_0} + y_1 . \frac{\frac{Prod_x}{Dif_1}}{Prod_1} +$$

$$y_2 . \frac{\frac{Prod_x}{Dif_2}}{Prod_2} + ... + y_n . \frac{\frac{Prod_x}{Dif_n}}{Prod_n}$$

	$x_0 =$	$x_1 =$	$x_2 =$	$x_3 =$	$\prod$
-	0.00	0.10	0.30	0.60	
	$D_0 =$	$D_1 =$	$D_2 =$	$D_3 =$	$P_x =$
$x = 0.20$	0.20	0.10	−0.10	−0.40	$P_x = 0.0008$
$x_0 = 0.00$	-	-0.10	-0.30	-0.60	$P_0 = -0.018$
$x_1 = 0.10$	0.10	-	-0.20	-0.50	$P_1 = 0.01$
$x_2 = 0.30$	0.30	0.20	-	-0.30	$P_2 = -0.018$
$x_3 = 0.60$	0.60	0.50	0.30	-	$P_3 = 0.09$

e agora

$$P_x(0.2) = (1.000 \times \frac{0.0008}{-0.018}) + (2.001 \times \frac{0.0008}{0.01}) +$$

$$(4.081 \times \frac{0.0008}{-0.018}) + (8.296 \times \frac{0.0008}{0.09}) = 3.008$$

7.3 Newton

Evita o inconveniente de ter que recalcular tudo quando se acrescenta um ponto a mais. Analiticamente $f[x_0, x] = \frac{f[x]-f[x_0]}{x-x_0}$ e $f[x_0, x_1, x] = \frac{f[x_0,x]-f[x_0,x_1]}{x-x_1}$ e $f[x_0, x_1, x_2, x] = \frac{f[x_0,x_1,x_2,x]-f[x_0,x_1,x_2]}{x-x_2}$ Da primeira equação, $f(x) = f[x_0] + (x - x_0)f[x_0, x]$ e daqui recursivamente, extendendo a expressão chega-se a $P_n(x) = f[x_0] + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2] + ... + (x - x_0)...(x - x_{n-1})f[x_0, x_1, ..., x_n] = \{...\}$. Acompanhe o exemplo
Seja um conjunto de pontos x_i e os valores da função a eles associados $f(x_i)$.

x	-2	-1	0	1	2
$f(x)$	-2	29	30	31	62

Note que os valores calculados estão em cores diferentes para que se possa perceber o mecanismo de construção desta tabela:

x_i	$f(x_i)$				
-2	-2	$\frac{29-(-2)}{(-1)-(-2)} = 31$	$\frac{1-31}{0-(-2)} = -15$	$\frac{0-(-15)}{1-(-2)} = 5$	$\frac{5-5}{2-(-2)} = 0$
-1	29	$\frac{30-29}{0-(-1)} = 1$	$\frac{1-1}{1-(-1)} = 0$	$\frac{15-0}{2-(-1)} = 5$	
0	30	$\frac{31-30}{1-0} = 1$	$\frac{31-1}{2-0} = 15$		
1	31	$\frac{62-31}{2-1} = 31$			
2	62				

A partir da tabela pode-se construir o polinômio interpolador. Note que as cores foram mantidas para que se possa perceber o mecanismo usado. Veja que apenas as primeiras linha e coluna é que são usadas na construção do polinômio

$$P(x) = -2 + [(x - (-2))] \times 31 + [(x - (-2)) \times (x - (-1))] \times -15$$
$$+[(x - (-2)) \times (x - (-1)) \times (x - 0)] \times 5$$
$$+[(x - (-2)) \times (x - (-1)) \times (x - 0) \times (x - 1)] \times 0$$

7.4 Newton-Gregory

Exige que as distâncias entre os pontos seja um valor constante, denotado como h. O polinômio é

$$p_n(x) = f(x_0) + (x - x_0)\frac{\Delta^1 f(x_0)}{h} +$$

$$(x - x_0)(x - x_1)\frac{\Delta^2 f(x_0)}{h^2 2!} + ... +$$

$$(x - x_0)(x - x_1)...(x - x_{n-1})\frac{\Delta^n f(x_0)}{h^n n!}$$

Suponha a função tabelada:

x	-2	-1	0	1	2
$f(x)$	-2	29	30	31	62

Seguindo o esquema acima descrito tem-se

x_k	$\Delta^0 f(x_k)$	$\Delta^1 f(x_k)$	$\Delta^2 f(x_k)$	$\Delta^3 f(x_k)$	$\Delta^4 f(x_k)$
-2	-2	29-(-2)=31	1-31=-30	0-(-30)=30	30-30=0
-1	29	30-29=1	1-1=0	30-0=30	
0	30	31-30=1	31-1=30		
1	31	62-31=31			
2	62				

Ou seja, $P(x) = -2 + (x - (-2)\frac{29}{1} + (x - (-2))(x - (-1))\frac{-30}{1^2.2!} + (x - (-2))(x - (-1))(x - 0)\frac{30}{1^3.3!}$ ou

$$P(x) = -2 + 29(x+2) - 15(x+2)(x+1) + 5x(x+2)(x+1)$$

8. Ajuste de curvas

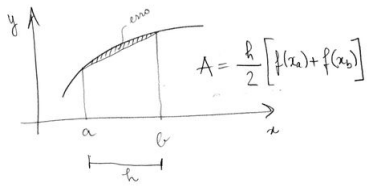
Uma reta pode ser ajustada com quantos pontos você quiser, resolvendo $\begin{bmatrix} n & sx_i \\ Sx_i & Sx_i^2 \end{bmatrix} . \begin{Bmatrix} b_0 \\ b_1 \end{Bmatrix} = \begin{Bmatrix} Sy_i \\ Sy_i x_i \end{Bmatrix}$.
A reta será $Y = b_0 + b_1X$.

9. Integração

Se $f(x)$ é contínua em $[a, b]$ e sua primitiva $F(x)$ é conhecida então

$$\int_a^b f(x)dx = F(a) - F(b)$$

9.1 Trapézio



A fórmula aqui é $I = \frac{h}{2}[f(x_0) + f(x_1)]$
Seja um exemplo. Se $f(x) = \frac{1}{x}$, então $\int \frac{1}{x} = ln|x| + C$. Para Calcular $\int_3^{3.6} \frac{dx}{x}$. Tem-se $I = \frac{h}{2}(y_0 + y_1)$ e como $y = \frac{1}{x}$ então $y_0 = \frac{1}{3} = 0.333...$ e $y_1 = \frac{1}{3.6} = 0.277...$ e $h = x_1 - x_0 = 3.6 - 3 = 0.6$ e daí $I = \frac{0.6}{2}(\frac{1}{3} + \frac{1}{3.6} = 0.1833...$

9.2 Múltiplos trapézios

A idéia agora é usar muitos trapézios, dividindo o intervalo $[a, b]$ em sub-intervalos e aplicar a mesma regra a cada intervalo. Fica:
$A_1 = \frac{h}{2}[f(x_0) + f(x_1)]$
$A_2 = \frac{h}{2}[f(x_1) + f(x_2)]$
...
$A_n = \frac{h}{2}[f(x_{n-1}) + f(x_n)]$, ou juntando tudo
$A = \frac{h}{2}[f(x_0) + 2f(x_1) + 2... + 2f(x_{n-1}) + f(x_n)]$.
Usando 6 intervalos, o mesmo cálculo acima dará 0.1823

9.3 1/3 de Simpson

com 1 intervalo: $A = \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)]$
Usando vários (um número par de) intervalos, fica $A = \frac{h}{3}[f(x_0) + 4f(x_1) + 2f(x_2) + 4... + 2f(x_{n-1}) + f(x_n)]$