CÁLCULO NUMÉRICO
COMPUTACIONAL.

Introdução à Computação em Pascal.

Tarcisio Praciano Pereira

Departamento de Matemática
Universidade Estadual Vale do Acaraú
Sobral, 5 de agosto de 2001tarcisio@e-math.ams.org -
fone 55 88 677 42 46

Tarcisio Praciano Pereira
PhD in Mathematics

CÁLCULO NUMÉRICO COMPUTACIONAL

Introdução à computação em Pascal

Publicações Eletrônicas
do
Laboratório de Matemática Computacional
UVA
Sobral - Ce

P496c

Pereira, Tarcisio Praciano
Cálculo Numérico Computacional: introdução à computação em Pascal.
Laboratório de Matemática Computacional - LMC
Sobral: UVA, 2001
133.p
Bibliografia
ISBN:85-87906-05-4

1 - Computação, 2 - Linguagem - Computação - Pascal
3 - Cálculo Numérico.
I. Título

CDD 515.1

Edições **UVA**
Universidade Estadual Vale do Acaraú- UVA
Av. Universidade, 850 - Betânia - CEP 62 040 370
Sobral - Ce

Reitor
Prof. José Teodoro Soares

Conselho Editorial
Prof. Evaristo Linhares Lima - Presidente
Prof$^{\underline{a}}$. Maria Norma Maia Soares - Secretária
Prof. Teobaldo Campos Mesquita
Prof. Modesto Siebra Coelho
Prof$^{\underline{a}}$. Zélia Leite Mendes

Coleção Textos Universitários

O autor,
Tarcisio Praciano Pereira,
Bacharel em Matemática pela UFC,
PhD em Matemática pela Universidade de Upsala - Suécia
Professor Titular da Universidade Federal do Rio Grande - RS,
Professor Titular da Universidade Estadual Vale do Acaraú- Ce

`Revisão ortográfica:` Prof. Vianey Mesquita
`Capa:` Harley Anderson Fernandes Rodrigues
`Digitação e diagramação` em LaTeX pelo autor.

Ao meu primeiro professor de aritmética, de filosofia,
de português, de inglês, de francês,
meu pai,
Gerardo Pereira.
À memória daquela que me ensinou a ler,
minha mãe,
Maria Antonieta Praciano Pereira.

Agradecimentos

Aos alunos que com suas críticas, me ajudaram a encontrar
o rumo ao escrever este texto.
Aos colegas de trabalho que leram e criticaram,
em particular, João Roberto Gerônimo que tem programas
incluidos aqui nestas páginas.
Ao prof. Vianey Mesquita pela cuidadosa revisão ortográfica,
sintática, léxica e semântica (mas os erros que ainda
assim ficaram, são meus).
Ao prof. Evaristo Linhares Lima, idealizador da coleção de *Textos Universitários*.
Ao prof. José Teodoro Soares,
Reitor da Universidade Estadual Vale do Acaraú,
que me colocou sobre os ombros a responsabilidade e a honra
de abrir com este livro a coleção
Textos Univeristários da UVA.

INTRODUÇÃO

A história, os objetivos e o plano do livro

O material contido neste livro foi testado em vários cursos de Cálculo Numérico na Universidade Estadual de Maringá, e na Universidade do Rio Grande. Presto minha sincera homenagem aos alunos que participaram desta experiência com clara consciência de participação e que, com suas perguntas críticas, muitas vezes me nortearam na determinação do melhor caminho. A primeira versão do livro tinha o objetivo extremamente modesto de fazer uma introdução ao Pascal, [7], para pessoas envolvidas com Matemática. Somente aos poucos se solidificou a idéia de escrever um curso de Cálculo Numérico, [8]. Este livro está contido num plano de dois livros independentes, mas que obviamente se completam:

- • o primeiro, este, é uma versão introdutória de Cálculo Numérico dedicado a iniciar o aluno em Pascal como uma primeira linguagem para escrever algoritmos que resolvam problemas de Cálculo, mas terminando com vida própria, num pequeno capítulo sobre *modelagem computacional de processos*, em que mostro como o material dos capítulos anteriores é aplicável num contexto mais amplo.
- •• o segundo representa um curso avançado de Cálculo Numérico voltado para soluções aproximadas de equações diferenciais. Não pressupõe diretamente que se tenha lido este, mas, naturalmente, conhecimento e habilidade com alguma linguagem de programação é um pré-requisito para sua leitura.

O plano deste livro se desenvolve ao longo de dois capítulos básicos e um de aplicações:

1. O primeiro capítulo trata de problemas elementares de aproximação, basicamente faz aquilo que deve ser feito num curso moderno de Cálculo I: cálculo aproximado da derivada, cálculo aproximado da integral.
2. O segundo capítulo já representa um vôo um pouco mais alto, pois trata da solução de equações da forma $f(x) = 0$ em que f é uma função “qualquer”.
3. O terceiro capítulo é uma introdução à *modelagem computacional de processos*, como aplicação direta do material contido

nos outros dois capítulos. É usado um pouco de estatística, que acho que o professor pode ensinar aos alunos informalmente, e servirá até mêsmo para reforçar a necessidade deste estudo se ele estiver na grade curricular.

O objetivo do livro é transmitir ao leitor os rudimentos de Pascal e quebrar as dificuldades iniciais com o aprendizado da linguagem. Espero que, posteriormente a esta *quebra de gelo*, o leitor se encontre em condições de prosseguir o seu aprendizado de Pascal sozinho ou, até mesmo, de uma outra linguagem de computador.

Não seria muito realístico supor que os alunos saibam Pascal em muitos casos até mesmo supor que saibam programar é irreal. Mas acho que é preciso *forçar a barra*, como tenho feito com os meus alunos de Cálculo Numérico, com resultados positivos: ao final do segundo mês de aula a maioria deles já sabe fazer programas e inclusive planejar um pequeno pacote fazendo uso de bibliotecas. Os programas crescem de nível de modo que, *se o aluno se empenhar em entendê-los*, ficará gradualmente no nível dos mesmos. Ao longo do primeiro capítulo, mantive a preocupação de explicar detalhadamente os programas, mas aos poucos vou deixando que o leitor comece a voar sozinho...obviamente, na companhia de um bom livro sobre Pascal e com a certeza de que sempre encontrará a sua volta algum *hacker*, que a menos de alguma *empáfia típica*, poderá lhe ser de muita ajuda.

Associo Pascal ao estudo da parte mais elementar de Cálculo Numérico que identifico como a *numerização das técnicas do Cálculo*: integral, derivada, máximos e mínimos, caso univariado e multivariado. Todos estes problemas podem ser resolvidos com uma técnica básica de programação, *varredura*, mais o acréscimo de algumas técnicas matemáticas auxiliares.

Assim, este livro pode tanto representar material para um curso de introdução à ciência da computação para estudantes das áreas de exatas ou tecnológicas como um curso mais elementar de Cálculo Númerico, ou ainda, espero, material auxiliar para ser usado pelo professor de Cálculo Diferencial e Integral que deseje motivar os seus alunos usando computação como um meio de *concretizar* os conceitos do Cálculo.

Este último ponto me parece de particular importância. Estou convencido de que o uso de computação no ensino de Matemática enriquece fortemente a experiência do aluno porque permite introduzir uma dinâmica que giz e quadro não conseguem mais gerar ante uma nova mentalidade gráfica que está presente em nós todos. Um exemplo é a comparação que pode ser feita de uma sequência de secantes que se aproximam de uma tangente, (ver o programa *Deriv01.pas*). É interessante observar a *surpresa* de alguns alunos que recém estudaram derivadas e não captaram o significado geométrico do coeficiente angular da reta tangente nem sua ligação com o coeficiente angular das secantes representados pelo quociente de diferenças. Eles quase instantâneamente passam a compreender o significado geométrico da derivada.

Neste livro, assumi o objetivo de aumentar sensivelmente o uso de métodos computacionais na disciplina. O objetivo é duplo:

1. Desenvolvimento tecnológico básico. Um país não se desenvolve se não possuir pesquisa básica, inclusive tecnológica. Nossos estudantes para isto tem que ser expostos às ferramentas básicas para produzir pacotes computacionais e algoritmos. Escolhi Pascal como linguagem para representar os algoritmos. Isto não significa, de minha parte, nenhuma indicação de que Pascal é a melhor linguagem para processamento numérico. Entretanto não há dúvida de que Pascal é uma ótima linguagem de apresentação didática para algoritmos sequênciais ou imperativos. Ela é usada freqüentemente como uma *pseudolinguagem* de programação. Existe uma outra linguagem, relativamente recente, *Python*, muito semelhante ao Pascal, porém bem mais poderosa do ponto de vista numérico, ver [11], com a vantagem de ser uma linguagem de domínio público com versões para LinuX e para Windows também.

2. Desenvolvimento científico avançado: Computação Algébrica. Um outro objetivo, num certo sentido *contraditório* com o anterior, é expor os alunos a uso direto de um programa de Computação Algébrica ou pelo menos a um programa de Cálculo numérico como **Maple**, **Derive**, **SciLab** e **MatLab**. **Derive** em particular não é uma linguagem de programação, é um poderoso pacote de Computação Algébrica feito em LISP e **SciLab** é um pacote de Cálculo Numérico com uma linguagem de programação associada, como também o são os outros.

A contradição se encontra em que um usuário de uma linguagem de computação algébrica **nunca** *será um construtor de uma tal linguagem.* Terá tecnologia, mas não a vai dominar.

Neste livro, me dedico ao primeiro objetivo, no outro, *Cálculo Numérico Avançado*, me entrego ao segundo.

Códigos de comunicação escrita.

A *oração escrita* tem que lançar mão de alguns códigos para transmitir os sentimentos que na oração falada são caracterizados por diferentes tons de voz ou gestos corporais. Este é um livro didático e, portanto, cheio da encenação típica da sala de aula. Basicamente o *itálico* é a minha grande marca de oração e geralmente quer indicar os aspectos vagos de alguma palavra ou sentença. Não tenho a menor presunção de conseguir ser completo ou enciclopédico, conseqüentemente, algumas vezes tenho que ser vago ou impreciso e sempre que o for conscientemente, o deixarei marcado com itálico ou reticências. O negrito se encontra em geral reservado para as palavras-chave do Pascal, mas há exceções óbvias a partir do contexto.

Chamo sua atenção para as observações. Elas são, em geral, *notas de rodapé* que ficariam muito grandes para serem assim utilizadas. Numa primeira leitura podem, ou até devem, ser ignoradas. Em geral são pequenos vôos filósoficos. Veja que o índice remissivo alfabético, ao final do livro, é uma forma de retornar às observações quando sentir necessidade delas.

Os programas.

Os programas usados neste livro se encontram disponíveis via *email* ou, se você preferir, poderá obtê-los nos enviando um disco flexível, com o porte pago, pelos Correios. Os programas são distribuídos num arquivo comprimido **pas.zip** junto com ele o programa **pkunzip.exe** para expandir o arquivo de volta a sua forma original (**pkunzip.exe** é um *shareware* que posso legamente distribuir, mas você deverá usá-lo com as devidas precauções). Tendo os programas em disco, não se esqueça de fazer-lhes um *backup* logo.

Vocabulário: 1 *Backup, compressão de arquivos, freeware, shareware.*

1. **Backup** *significa cópia de reserva. Os discos de computador ainda são muito frágeis e podem se danificar com relativa facilidade até mesmo por questões climáticas. Em razão disto é*

muito importante ter cópias de reserva de todo o material que se tenha em discos e naturalmente guardar tais cópias em locais diferentes livres de umidade, de alta temperatura e afastadas de campos magnéticos.

2. **Compressão de arquivos** *Os arquivos têm espaços em branco que podem ser eliminados sem nenhuma perda da informação neles contida, desde que se saiba fazê-lo. Os programas* **pkzip.exe, zip.exe, arj.exe** *sabem fazer isto muito bem. O programa* **pkzip. exe** *é um shareware e precisa de um programa chamado* **pkunzip.exe***, produzido pelo mesmo grupo, para restaurar o programa primitivo a partir do arquivo comprimido. Os outros são* freeware*. Todos estes programas podem ser obtidos em vários pontos da* rede*; ver por exemplo em* **http://www.cdrom.com** *que, apesar de ser um nó comercial, distribui programas de domínio público; veja também http://www.debian.org. Para os sistemas rodando LinuX, se tem o* **tar** *e o* **gzip***.*

3. **Freeware** *Há dois grandes sistemas de produção independente de programas:* **freeware***, programas livres, e* **shareware***, programas de uso em concessão . Os programas que os autores classificam como* freeware *são também chamados de* programas de domínio público*. Seus autores os deixam livres para uso de qualquer pessoa em geral, com a restrição de que eles não sejam usados para fins comerciais. Este livro foi inteiramente feito com programas de domínio público, com excessão dos programas em Pascal que foram produzidos e testados com um pacote comercial: Turbo Pascal. Todos os programas que produzi e que se encontram citados neste livro são de domínio público, portanto* freeware*.*

4. **Shareware** *É uma outra forma de produzir programas em forma independente das grandes empresas. Os autores que produzem programas classificados como tal fazem a distribuição de seus programas pela* rede *e os usuários podem experimentá-los durante algum tempo, devendo, posteriormente, se os considerarem úteis, pagar uma taxa de uso diretamente ao autor. É o caso dos programas* **pkzip, pkunzip** *que estou distribuindo, legalmente, junto com os meus programas.*

No arquivo **pas.zip**, você poderá encontrar todos os programas que se encontram listados no livro e até algumas variantes deles. Todos os programas foram testados várias, vezes inclusive por alguns alunos. Mas ainda podem ter ficado erros, pelo que me desculpo.

Os programas em Pascal foram feitos e testados com TurboPascal da Borland. Todos os programas podem ser usados e modificados livremente, espero, apenas, que seja feita a devida citação.

Este livro foi composto com o processador de textos TEX produzido por Donald Knuth, junto com a implementação para micros EmTEX de Eberhard Mattes, que faz a interface do TEX via LaTEX. LaTEX foi produzido por Leslie Lamport. Foi usada também a "casca" TEXShell que gerencia os módulos do EmTEX, produzida por Jürgen Schlegelmilch na versão para DOS. Os gráficos foram traduzidos para TEX com auxílio do programa **bm2font** produzido inicialmente por Friedhelm Sowa e modificado por várias outras pessoas. Todos estes programas são de domínio público e podem ser obtidos em algum dos CTAN, Comprehensive TeX Archive NetWork, que se encontram em diversos nós da *rede*, ver por exemplo:

ftp ftp.dante.de /pub/tex/systems/msdos/emtex

ftp ftp.univie.ac.at /pub/...

www http://www.dante.de; http://www.univie.ac.at

e-email Onde você pode obter os meus programas em Pascal:

tarcisio@e-math.ams.org

Você pode acessar estes nós usando os métodos do "www" e os programas que se comunicam com ele. Sou grato aos autores dos programas de domínio público ou de shareware usados neste livro.

Tarcisio Praciano Pereira
e-email tarcisio@e-math.ams.org
Sobral, 5 de agosto de 2001.

Capítulo 1

Sucessões, somas e limites

Introdução

Os programas deste capítulo partem todos de um mesmo programa inicial e vão se tornando aos poucos mais sofisticados. Variantes deles serão estudadas para construir sucessões, calcular valores de funções, determinar raízes de funções. Todos estes problemas se resolvem com uma mesma técnica computacional: *varredura*. Vou apresentar-lhe os problemas num processo crescente de dificuldades, entremeados de exercícios, para que você adquira o domínio dos métodos.

Começarei com um programa bem simples, exatamente para mostrar como se fazem programas. Ele irá aos poucos se tornando mais complexo. Na primeira seção, discutirei a *estrutura* de um programa em Pascal.

Você vai precisar de consultar um livro sobre Pascal. Vá à biblioteca e escolha aquele que melhor sintonizar com o seu estilo de leitura. Neles há muito mais informação sobre esta linguagem de programação do que será necessário para acompanhar este livro. O que você deve fazer é complementar, com o livro de Pascal, exatamente aquilo que for necessário para entender o conteúdo deste.

1.1 Estrutura de um programa em Pascal

1.1.1 O ambiente integrado do TurboPascal

Este livro não é um livro *sobre Pascal*, apenas *usa Pascal*. Estou supondo que você já adquiriu um livro *sobre Pascal* e que o irá usar em companhia deste. TurboPascal lhe oferece um *ambiente integrado* que é muito rico de opções e *efeitos especiais*, por exemplo algumas técnicas para fazer *debug*.

Vocabulário: 2 *Bug, debug.*

Bug *É uma palavra inglesa para "pulga" que também é usada na gíria para insetos em geral ou probleminhas. Pegou na gíria* *de computação para representar pequenos erros, pequenos mas chatos.*

Debug *É uma palavra da gíria do pessoal de computação para representar o trabalho de catar os pequenos erros de grandes programas, a limpeza dos "insetos" que ficam sobretudo em programas com centenas ou milhares de linhas... evite fazer programas que sejam maiores que a tela do micro... e você certamente nunca vai necessitar de nenhuma técnica especial para* dedetizar *os seus programas...*

Mas o *ambiente integrado* do TurboPascal é um ambiente rico e vale a pena dominar-lhe as possibilidades. O primeiro exercício irá induzí-lo a este aprendizado. Por exemplo, duas teclas são *quentíssimas* dentre as *teclas de atalho* do TurboPascal, *hot keys*: F1 e CTR-L F1, ambas lhe fornecem informações ao vivo, *on line*, aprenda a usá-las. O seu livro sobre Pascal deve lhe ensinar tudo isto, desde que você tenha adquirido o livro certo...sobre TurboPascal 5.5 em diante...

Vocabulário: 3 *Ambiente integrado, o ratinho, apontadores.*

Ambiente Integrado *Nas linguagens de programação modernas o programador trabalha com várias ferramentas que o auxiliam a programar. É isto que se chama de um* ambiente integrado *de programação. A tendência é de que os programas fiquem até mesmo independentes de uma determinada linguagem de programação . Já existem ambientes integrados que produzem pedaços de programa a partir de uma seleção de tópicos feita pelo usuário usando o ratinho. As linguagens* Delphi *e* Maple*, por exemplo, produzem pedaços de programa automaticamente, Maple lhe permite escolher entre duas outras linguagens.*

o ratinho *Do inglês,* mouse=rato*, é um instrumento que serve para apontar objetos na tela. Faz parte de uma classe de instrumentos chamados* apontadores*, cuja família cresce muito hoje, tornando mais fácil a arte de programar computadores porque permitem aumentar a interação física do usuário com o computador.*

1.1.2 O primeiro programa

Para resolver problemas precisamos encontrar uma seqüência de operações elementares que conduzam à solução esperada: um algoritmo.

Observação: 1 *Celularização das operações Se nos permitirmos filosofar um pouco, a idéia das operações elementares é bem natural. A matéria se subdivide em partículas elementares, ou células ou moléculas. De modo análogo, as* tarefas *se organizam em tarefas mais simples ou mais elementares. Quando conseguimos descobrir os* elementos *de um processo, o problema que este processo representa se encontra resolvido. Nada há de estranho que a economia se celularize; faz parte da busca de organização. O estranho é o capitalismo procurar isolar o homem e definí-lo como supérfluo...*

Não há um *método* para especificar como resolver problemas. Um matemático polonês que viveu na Inglaterra, Polya, escreveu um livro famoso em que tenta, com vários exemplos, mostrar como fazer algoritmos. É um livro interessante, mas, obviamente, não consegue *resolver isto*[1].

Vocabulário: 4 *Programa, algoritmo.*

Algoritmo *é uma listagem de operações com o objetivo de resolver um problema específico. É como o* algoritmo da divisão euclidiana*, que especifica que você deve ir separando no dividendo tantas casas quantas tiver o divisor e ir procurando o quociente máximo entre o divisor e estas duas casas e suas diferenças sucessivas. Um algoritmo se compõe de uma sucessão de operações elementares que no conjunto resolvem um dado problema. Criou-se um hábito, entre os professores de programação : falar em "algoritmo" como sendo uma tradução em Português dos programas em Pascal, que depois devem ser traduzidos para Pascal. A idéia é boa em princípio, porque induz a uma maior generalidade, mas tem o defeito de afastar, inicialmente, o estudante do computador enquanto ele aprende a fazer algoritmos. Vou evitar isto aqui.*

Programas *são algoritmos escritos numa determinada linguagem de programação . Neste livro todos os algoritmos serão escritos em Pascal diretamente.*

O primeiro programa em Pascal:

[1] O livro de Polya se intitula "How to solve it".

Exemplo: 1

```
1  Program Pas00;
2  Begin
3    Write(' O primeiro programa ');
4  End.
```

Neste exemplo, o programa tem 4 linhas. As linhas estão numeradas apenas para facilitar nossa comunicação com você. Pascal não aceita que isto seja feito. Não há o conceito "linha de programa" em Pascal e sim o de comando. Na verdade este programa tem 4 *comandos*. Cada comando é terminado com ";" sendo que ao final tem se um "." Pascal exige que os programas terminem com um "ponto final". O primeiro *comando*, "Program Pas00;", é desnecessário. É apenas uma regra de arrumação iniciar todos os programas dando-lhes um nome. Este programa apenas escreve na tela a frase "O primeiro programa". Mas se você "rodar" este programa, não vai ver a frase impressa na tela, porque logo que o programa terminar de rodar, o *ambiente integrado* retoma o seu espaço e tira o resultado do programa da tela. Com "Alt-F5" você pode ver o que houver na tela por traz do *ambiente integrado*.

Tente rodar o programa e usar "Alt-F5".

Vocabulário: 5 *Ctrl-F1, F1, Alt-F5*

Ctrl-F1 *significa que você deve apertar simultaneamente as teclas "Ctrl" e "F1".Chama o "help" específico.*

Alt-F5 *significa que você deve apertar simultaneamente as teclas "Alt" e "F5". Alterna as telas.*

1.1.3 Variáveis num programa

As linguagens modernas de processamento exigem que se lhes diga tudo a respeito das "variáveis" que se desejam usar. Umas são mais exigentes, como TurboPascal, outras são mais flexíveis, como *Python*.

Vocabulário: 6 *Memória, ponteiro, variável.*

1. **Memória** *é um segmento, conjunto seqüencial, de endereços numa placa do computador onde o sistema operacional pode*

armazenar dados. Chama-se ainda `RAM` *ou memória volátil, porque os dados desaparecem ao se desligar o computador. Com auxílio de ponteiros, nos discos, é possível armazenar largas quantidades de dados em forma não seqüencial. Dados* on line *podem ser memorizados em variáveis e algumas linguagens de processamento permitem que se armazenem as variáveis em arquivos de disco. TurboPascal não oferece esta possibilidade, diretamente.*

2. **ponteiro** *É um tipo de dado, em Pascal, que liga, de forma mais direta, o nome de uma variável a um endereço de memória.*

3. **Variáveis** *são* dados estruturados *em Pascal. As variáveis representam uma abstração computacional, a linguagem de programação, no presente caso Pascal, associa a uma palavra, que tem significado para o programador, um segmento da memória onde irá armazenar dados associados a esta palavra. O programador usa a palavra para chamar e operar sobre os dados e Pascal traduz para o sistema operacional o endereço, assim como os códigos de máquina que correspondam às operações que devam ser efetuadas. As variáveis ganham um endereço de memória quando definidas e podem assim ser recuperadas, modificadas, durante o processamento, como também ocupam espaço uma regra que deve ser observada é a de economizar variáveis. Entretanto, no início, o conselho que lhe dou é quase o inverso: use bastante variáveis, para aprender a usá-las corretamente. Mas, à frente, voltarei a alertá-lo para esta economia de variáveis e inclusive mostrar-lhe os meios de evitar o uso excessivo delas.*

   A palavra usada para identificar *uma variável, se chama em Pascal* identificador, identifier *em inglês.*

   Uma sentença *típica em Pascal é a seguinte:*

   $$x := x + 1;$$

   que significa: "a variável x *deve passar a guardar o valor de* x *acrescido de uma unidade". Não se trata de uma equação matemática.*

Pascal é muito estrito com esta regra de descrição de variáveis, e isto o torna uma linguagem muito didática: quem desejar programar em Pascal tem que ser arrumado e saber fazer um trabalho planejado. É preciso dizer quais são as variáveis a serem utilizadas, deve-se indicar se elas irão guardar números, letras, pares ordenados, conjuntos etc... O programa simplesmente não roda se isto não for feito!

Mas em geral não precisamos de um vocabulário tão envolvido para trabalhar com programação; há muita coisa que os "pacotes" decidem e fazem sem que precisemos nos envolver com detalhes. A regra é simples: temos que aprender a escrever a lógica dos nossos algoritmos com uma linguagem que se assemelhe muito à que usamos para nossas comunicações pessoais. Mas, como disse no começo, nem sequer precisamos conhecer um exemplo particular de linguagem porque alguns *ambientes integrados* escrevem programas; pelo menos programas simples eles já sabem escrever.

Exemplo: 2 *A rigidez da definição das variáveis.*

Pascal não aceita que num certo momento **x** *seja um número inteiro para logo em seguida representar um número racional, (o inverso pode acontecer porque um número inteiro também é racional). Se você tentar fazer alguma coisa deste tipo, o* compilador *pára o trabalho de análise do programa, quando encontrar a primeira discrepância, vai lhe dizer isto:* erro de tipo de dados, *type mismatch*, variável não corresponde ao tipo indicado.

Aliás, o *compilador* desenvolve um *monólogo* com você, o que representa uma importante ajuda neste aprendizado. Não espere muito dele, entretanto!

Vejamos um exemplo. Abaixo você tem um *vocabulário* em que descrevo o sentido de algumas palavras ou de comandos do Pascal. Em seguida, vem o segundo programa. Talvez você devesse ler imediatamente o programa e depois retornar ao *vocabulário*.

Vocabulário: 7 Begin/End., Compilador, Integer, Program, Read, ReadLn, Var, WriteLn, ;,:=

Há dez "palavras" acima, com as quais vou montar o segundo programa em Pascal. Pascal tem algumas palavras *básicas com que a linguagem lhe é entregue para uso. Depois você verá que pode criar também as suas próprias palavras: os seus próprios comandos.*

1. **"Begin"** *junto com* **"End."** *marcam o início e o fim do algoritmo. Observe o ponto final que faz parte da palavra.*

2. **Compilador** *é a parte do "sistema Pascal", ou do* ambiente integrado *que analisa o programa para verificar a sua correção em relação às regras do Pascal, portanto uma análise sintática. Se tudo estiver* sintaticamente correto, *ele cria um* código *executável. Alguns compiladores chegam a encontrar alguns erros de lógica, mas esta habilidade ainda se encontra nos seus primeiros passos. Em geral o* compilador *está preparado para agir conversacionalmente com você, lhe indicando os erros cometidos. O compilador que estou utilizando está integrado no sistema TurboPascal. Por esta razão , se diz que você tem a sua disposição um* ambiente integrado *em que se encontram presentes o compilador, um editor de texto e uma série de outros programas que lhe poderão servir de auxílio.*

3. **Integer** *é uma palavra reservada do Pascal para criar variáveis do tipo inteiro.*

4. **Program** *marca o início do texto e pode ser seguida do nome do programa. A linha deve terminar com ";". Olhe o texto do programa que vem a seguir.*

5. **ReadLn** *faz o programa parar e esperar que você lhe forneça alguma informação pelo teclado. Provoca uma mudança de linha também.*

6. **Read** *não provoca mudança de linha e deve ser evitado agora no começo porque ele exige uma limpeza na memória do teclado. O uso de* **Read**, *em Pascal, provoca erros inesperados. Evite usar este comando enquanto não tiver uma experiência maior.*

7. **Var** *é uma palavra reservada do Pascal indicando uma região do programa em que as variáveis serão definidas.*

8. **WriteLn** *escreve* alguma coisa *numa linha nova. Escreve aquilo que você mandar escrever...*

9. **;** *Todos os comandos devem terminar com* **";"** .

10. **:=** *é o comando do Pascal para fazer uma atribuição . Atribuição é uma palavra de "computês" que significa que você vai dar um valor a uma certa variável. Abaixo você encontrará a expressão :*

$$contador := contador + 1$$

que significa: "contador" deve ser aumentado de uma unidade e em seguida este novo valor deve ser colocado na "variável contador". Veja que as operações foram efetuadas de traz para frente: primeiro foi somado 1 ao valor de contador*, depois foi atribuído a* contador *este novo valor. Não se trata de uma equação matemática.*

Não se esqueça, num programa em Pascal as linhas não são numeradas. Estou enumerando as linhas, abaixo, para facilitar as referências posteriores. Se você numerar as linhas, Pascal indicará um erro, possivelmente *unknown identifier* : um identificador de variável desconhecido.

Exemplo: 3

```
1  Program Pas01;
2  Var   contador : Integer;
3  Begin
4.1  contador := 0;
4.2  WriteLn(contador);
4.3  contador := contador + 1;
4.4  WriteLn(contador);
4.5  ReadLn;
5  End.
```

1. *O título "Program Pas01;"*

   é o título do programa. Não é obrigatório.

2. *Definição de variáveis, Var contador : Integer;*

   Declaração da variável **contador** *como "inteiro".*

3. *Onde começa o programa: Begin*

   Com esta palavra, Pascal reconhece que se inicia o corpo do programa.

4. *O corpo do programa.*

   contador inicializada com o valor zero:
   contador := 0;
   o programa escreve o valor de contador: 0 .

WriteLn(contador);
Em seguida contador é incrementada de uma unidade:
contador := contador + 1;
O programa escreve o novo valor de contador: 1
WriteLn(contador);
ReadLn; O programa aguarda que você
se comunique com ele pelo teclado, aperte "Enter".
Se você omitir este comando, o
ambiente integrado *toma conta do espaço*
e você não vai ver o último resultado.
Tente apagá-lo.

5. *End.*

   Indica que o programa termina aqui. Observe o "ponto final".

A observação feita sobre o comando:

contador := contador + 1;

é essencial; trata-se de um comando composto. Primeiro temos uma adição: *contador + 1*, depois uma atribuição representada pelo comando ":=" significando que *contador* irá receber o valor

"contador + 1".

Trata-se de um comando e não de uma equação matemática.

Este programa faz pouca coisa, apenas escreve

0 , 1

porque apenas isto foi programado fazer... mas já lhe dá uma idéia de como deve se estruturar um programa em Pascal.

Em linhas gerais um programa tem duas partes:

1. rótulo do programa é a região em que se fazem as declarações.

2. o corpo do programa que começa com **Begin** e termina com "**End.**" .

Na região do rótulo, você tem que informar ao Pascal todas as variáveis que vão aparecer no programa. Até o final deste capítulo estas idéias estarão definitivamente claras. Vejamos mais exemplos modificando o programa anterior.

1.1.4 O terceiro programa

É fácil transformar este programa noutro que use a variável contador para imprimir ou somar sucessivamente alguns números naturais.

Exercício: 1 *Modificando o programa Pas01 ...*

1. *Acrescente um "Writeln(contador)" nalgum lugar para que o programa escreva 2, depois de escrever 1.*

2. *Acrescente mais algumas linhas no programa para que ele escreva de zero até dez.*

3. *Acrescente um "Writeln" contendo a mensagem* 'agora o numero eh' *observe que os apóstrofos indicam ao Pascal onde começa e termina a sua mensagem; omita o primeiro apóstrofo para ver o que acontece. Veja que escrevi "eh" em vez de "é" porque Pascal é americano e não sabe usar os nossos símbolos diacríticos. Veja no índice remissivo,* diacríticos*, onde pode encontrar a solução para este problema.*

   Observação: 2 *O marcador de mensagens*

   No teclado tem o apóstrofo *exato para marcar mensagens; rode o programa quantas for necessário para descobrir qual é. Em geral coincide com o* acento agudo*. Se você usar um símbolo errado, receberá uma mensagem de erro provavelmente da forma* string too long*, "cadeia de caracteres muito longa", porque Pascal tem um limite para o tamanho de mensagens e ele deve ter entendido que o resto do programa era uma mensagem.*

4. *Repita a experiência anterior deixando o primeiro apóstrofo mas omitindo o segundo.*

Se você tiver escrito uma mensagem sem apóstrofos dentro do Writeln o compilador deve ter lhe dito, (parando na primeira palavra), que encontrou um identificador desconhecido: *Unknown identifier or syntax error*; ele verificou que a "primeira palavra" que estava dentro do Writeln não havia sido devidamente explicada dentro do comando **Var** e parou. Se omitiu apenas o apóstrofo final, a mensagem de erro ficou diferente, ele lhe disse que estava recebendo uma *string* muito longa *por que todo o resto do programa virou mensagem...* Está na hora de fazer várias experiências para compreender este item.

Vocabulário: 8 String.

String *é uma "cadeia de caracteres"; quer dizer* uma sucessão de caracteres*. Pascal se limita a receber* strings *com 256 caracteres. Se você eliminar o apóstrofo que marca o fim de uma mensagem, ele continua lendo o resto do programa como se fosse a mensagem. Você assim poderá exceder este limite porque não indicou o fim da* cadeia de caracteres*. Claro, se você tiver executado a experiência...*

1.2 Loops ou laços: para repetir coisas...

Uma das funções mais importantes de um computador é de *repetir tarefas*, ou melhor, executar tarefas repetitivas, porque nem sempre se trata de apenas *repetir* mas algumas de vezes de executar tarefas em que um pequeno detalhe vai sendo mudado de uma para outra.

Compare o programa abaixo com os exemplos acima. Observe que lhe foram acrescentadas duas linhas, contendo as palavras **Repeat** e **Until**. Rode o programa *agora* para o ver o efeito; as vezes entendemos melhor as coisas depois que as vemos acontecer... Depois retome a leitura.

Vocabulário: 9 Repeat, Until, loop .

1. *As duas palavras* **Repeat** *e* **Until** *formam um par de parêntesis qualificados. Elas marcam o início e fim de um loop.* **Repeat** *significa* Repete *e* **Until** *significa* Até que*. Assim Pascal irá repetir o que fica entre estas duas palavras* até que *a condição do* **Until** se torne verdadeira*. Veja o próximo programa, como exemplo.*

2. `loop` *é um pedaço de programa que vai ser repetido algumas vezes. Também se diz "laço".*

O que lhe pedi num dos exercício acima - *escrever sucessivamente os números até dez* - foi complicado. Você se viu obrigado a repetir as linhas:

```
contador := contador + 1;
WriteLn(contador);
```

várias vezes. O `loop` **Repeat-Until** lhe permite resolver este problema de maneira mais agradável.

Observação: 3 *Erros e as mensagens de erro*

Se você omitir uma das palavras "Repeat, Until" o compilador pára lhe dizendo que houve um erro. Experimente agora: retire **Until**, *rode o programa e veja que tipo de mensagem o compilador lhe endereça. Você verá que falta alguma lógica na mensagem...Experimente para ver o que acontece... não tenha preconceitos contra erros, você verá que pode até mesmo ser divertido ver o compilador achar os seus erros,* algumas vezes chateia.... *Também lhe vai ser necessário aprender a interpretar* as observações do compilador *sobre os erros encontrados. Por exemplo, omita algum ";". Algumas vezes ele vai dizer-lhe que* esperava encontrar um ";" *outra vezes ele vai dizer-lhe que encontrou um "identificador" desconhecido... Experimente! apague algum ";" do texto do programa e mande rodar com Ctrl-F9. Depois apague outro e veja a diferença na mensagem, se houver.*

Vocabulário: 10 *Ctrl-9, rodar um programa, F10.*

Ctrl-F9 *A* tecla de atalho *Ctrl-F9 ativa o* Run *do ambiente integrado que roda os programas.*

rodar um programa *significa fazer com que o compilador execute as intruções do programa até o final.*

F10 *esta* tecla de atalho *ativa o menu superior da tela. Experimente apertando-a agora. Depois com as setas* $\rightarrow, \leftarrow$ *você pode alterar o assunto principal e com a seta* $\downarrow$ *você tem acesso aos subtópicos daquele menu. Experimente.*

Aqui está a solução do exercício para escrever uma suces-são de números. No exemplo que segue, numerei as linhas do programa. Se você fizer isto, o compilador irá acusar um erro... Eu o fiz apenas para facilitar as referências no próximo exercício.

Exemplo: 4

```
1  Program Pas03;
2  Var     contador : Integer;
3  Begin
4     contador := 0;
5     Repeat
6           contador := contador + 1;
7           WriteLn(contador);
8     Until contador > 100;
9  End.
```

Se você tiver rodado o programa, o resultado será uma listagem na tela do computador dos números crescendo sucessivamente desde 1 até 101. Ou melhor, você viu passarem rapidamente ante os seus olhos os números de 1 até 101...

Como se podem ver os resultados ?

Esta é uma falha do *sistema* TurboPascal. Roda os programas e volta em seguida para o editor de texto mostrando o *código do programa* e portanto escondendo os resultados obtidos. Aperte Alt-F5 e verá na tela *parte* dos resultados escondidos. Este problemazinho pode ser resolvido acrescentando-se a linha

ReadLn;

entre as linhas 8 e 9 do programa. Seu efeito é o de forçar o sistema a esperar que você aperte uma tecla qualquer antes de sair do ar. Seu programa ficaria assim:

Exemplo: 5

```
1  Program Pas04;
2  Var     contador : Integer;
3  Begin
4     contador := 0;
5     Repeat
6           contador := contador + 1;
7           WriteLn(contador);
8     Until contador > 100;
9  ReadLn;
10 End.
```

Ainda assim a solução é fraca, você poderá ver apenas os 23 últimos resultados dos 100 que foram executados. Pense em como resolver este probleminha. Logo me ocuparei dele diretamente.

Se o comando

contador := contador + 1

for esquecido, o programa cai num *Loop infinito* porque **contador** permanece com o seu valor inalterado nunca chegando em 100 e conseqüentemente a condição que termina o programa nunca será verdadeira:

Until contador > 100 ;

Os programas seguintes estão *errados*. Onde estão os erros ?

Não é na definição da variável contador! Observe que escrevi **Var** numa linha, contador : **integer** na seguinte. O compilador não se incomoda com o espaço que ficou entre Var e o resto do comando. Este espaçamento pode ser usado para fazer o texto do programa ficar mais bonito e sobretudo mais legível.

Observação: 4 *As marcas que enfatizam a oração*

Os textos que os humanos *lêem, estão crivados de enfeites:* palavras sublinhadas, letras acentuadas, frases em tamanho gigante,... *todos estes "enfeites" são necessários para traduzir* *aspectos psicológicos da oração. Um programa de computador não pode ter tais complementos oratórios. Mas nós, humanos, continuamos a ler os programas e precisamos de "marcas" para entendermos mais rápido o conteúdo do texto. A solução é usar de modo inteligente a distribuição espacial deste texto, sem incomodar o compilador com outras marcas que ele não poderia entender. Os espaços, a tabulação e as observações (de que logo falaremos), são para facilitar a leitura para os humanos.*

Portanto, aí, no espaço entre "**Var**" e "contador : **integer**", não há nenhum erro. O erro é de natureza lógica. Se você *rodar* este programa, vai ver que o computador não vai se manifestar. O computador não vai parar sozinho. Como isto vai acontecer algumas vezes, é bom que seja agora para que você perca preconceitos contra erros...

Uma maneira de resolver o problema consiste em desligar o computador... algumas vezes vai resolver se você apertar simultaneamente as teclas **Ctrl Break** ou **Ctrl-C**. Identifique o "Break" no seu teclado, e experimente apertar **Ctrl Break**. A maneira "certa" é: "dedo mindinho esquerdo no Ctrl e o outro dedo mindinho no Break" que se encontra na última posição dentre as teclas especiais acima no teclado.

Ctrl - Break ou Ctrl - C

O resto aparece na tela e você tem apenas que seguir as instruções.

Depois você vai aprender meios melhores de lidar com problemas semelhantes, acrescentando variáveis de controle para ver o que acontece enquanto o programa roda. Por enquanto, experimente! E finalmente desligue o computador se não tiver dado certo. Dentro do Windows **Ctrl-Alt-Del** provoca uma mensagem lhe perguntando se deseja que o sistema interrompa o programa. Responda que "sim" em vez de desligar o computador. Dentro do `LinuX`, abra outra janela, chame Top, e com ele pare o programa enguiçado.

Exemplo: 6 *Programas errados.*

```
Program Pas05;

Var
   contador : Integer;
Begin
   Repeat
     contador := 0;
     WriteLn(contador);
     contador := contador + 1;
   Until contador > 100
End.
```

```
Program Pascal0501;

Var
   contador : Integer;
Begin
     contador := 0;
     Repeat
           WriteLn(contador);
     Until contador > 100
End.
```

Os programas não funcionaram... como *avisei*. Vamos primeiro ver como resolver o problema de *parar um programa* que parece estar em **loop** infinito.

Vocabulário: 11 Alt, Ctrl, Esc, Ctrl-C, Ctrl-Break, Shift. . . , loop infinito

No teclado do computador existem algumas teclas especiais: **Alt, Ctrl, Esc, Shift** *que devem ser usadas junto com uma outra tecla. Há também teclas especiais que executam tarefas específicas:* **Ins, Del, Home, End, PgUp, PgDw**. *Elas podem ter denominação ligeiramente diferente: "Insert" em vez de "Ins". Analise o teclado, identifique estas teclas. Veja, experimentando, qual é função que estas últimas teclas têm: não há perigo de acontecer nada errado, a não ser com com* **Del** *que apaga a letra sob o* cursor*. Há outras teclas especiais cujo uso você aprenderá depois.*

1. **Ctrl-C**. *São teclas modificadoras.* **Ctrl-C** *não é o mesmo que C mas é o mesmo que* **Ctrl-c**. *Alt, Ctrl, Esc produzem* caracteres de controle *junto com outras teclas. Assim Ctrl-C é um* caráter de controle *que, com freqüência é utilizado com a finalidade de parar programas. Depois você verá que dentro do Pascal se pode ativar ou desativar o Ctrl-C. É um modo de fazer o Pascal trabalhar mais rápido. Se agora o Ctrl-C estiver desativado, a solução será desligar o computador.*

2. **Ctrl-Alt-Del** *Dentro do Windows esta seqüência ativa o modo de gerenciamento de programas que lhe vai perguntar se deseja desligar o computador ou simplesmente parar uma tarefa que não está se comportando dentro do padrão . O Windows às vezes consegue parar programas que estão rodando enlouqueci-*

dos... às vezes também não consegue e a solução mesmo é desligar o computador.

3. **Ctrl-Alt-Del** *Fora do Windows, no MS-DOS, Ctrl-Alt-Del faz com que o computador volte ao processo de inicialização . Mas algumas vezes pode não funcionar, quando estiver nalgum loop de alta prioridade que não permita acesso ao teclado. A solução será a chave "liga-desliga"...*

4. **Ctrl-Break**. *Semelhante ao* **Ctrl-C**. **Alt, Esc** *tem funcionamento semelhante, modificando as demais teclas, o que permite portanto uma multiplicação do teclado por uma quantidade de combinações destas teclas. Você deverá aprender posteriormente a modificar o uso destas teclas para atender às suas necessidades específicas. Veja no índice remissivo* personalizando o Pascal.

5. **default** *Gíria que significa o* valor padrão *de alguma variável do sistema computacional com que se trabalha. Diz-se* valor default *para indicar que o valor em referência será automaticamente usado a não ser que você deliberadamente o modifique.*

6. **loop infinito** *O computador sempre anda em círculos, mas foi concebido de modo a* parar ocasionalmente *e verificar se em alguns pontos chamados "interrupções" há* ordens de serviço *para serem executadas. Algumas vezes, entretanto, e até mesmo por defeito do* sistema operacional *o computador entra num* ciclo *que o pode impedir de procurar o teclado que se encontra ligado a uma dessas interrupções e neste momento não há comunicação possível entre o usuário e o computador. É quando se diz que ele entrou em* loop infinito. *Loop quer dizer ciclo, em inglês. Entrar em* loop *pode acontecer sempre com qualquer programa pois todos são feitos por seres humanos que terminam por esquecer algum detalhe. Também há interferências elétricas ou eletromagnéticas que devem ser consideradas como agentes de alterações inesperadas. Veja como* parar programa *no índice remissivo.*

Observação: 5 *Perda de dados*

Um dos problemas mais graves em computação é perda de dados. Esta é uma dor de cabeça *que nos acompanha a todos que usamos os computadores. Os dados se encontram na memória dos computadores,* temporariamente, *ou em discos. Em ambos os casos se podem dar problemas que podem ser evitados se tivermos alguns cuidados. Veja no* *pé da tela do ambiente integrado do Turbo Pascal. Ao lado da tecla F2 se encontra a palavra "save" que significa "gravar". Depois de escrever o seu programa, antes de rodá-lo, dê um toque na F2. Seu programa ficará gravado e se der "pau" no computador, você não precisará fazer tudo de novo.*

Outro problema é com os discos. Tudo o que você fizer grave em no mínimo dois discos. As regras de segurança indicam que devem ser tres discos, o terceiro deve ficar guardado em casa e de tempos em tempos atualizado. Muita dor de cabeça pode ser evitada se isto for feito. É isto que se chama de "backup" de discos.

Uma terceira observação: os discos de computador são material sensível. Sensível ao calor, à poeira, à umidade etc... e para isto devem ser conduzidos sempre protegidos em caixas apropriadas. Aos poucos você verá que estes cuidados são cruciais, não menos para a própria conservação do computador que você estiver usando: um disco contaminado por poeira, ou fumaça de cigarro*, pode estragar a cabeça de leitura do drive, pode produzir distorções de informações.*

Finalmente, ainda tem os virus...

As teclas modificadas por Esc, Ctrl, Alt junto com as teclas especiais F1,...,F10,F11,F12 recebem às vezes o nome de "teclas de atalho", *Hot Keys* , *atalhos*, porque elas fazem operações compostas economizando tempo. Veja no manual a respeito delas e vá aos poucos se acostumando com os seus significados.

Observação: 6 *Ativando/desativando o Ctrl-C, Ctrl-Break*

Enquanto você não tiver certeza de que o programa esteja correto, é bom deixar o compilador de alerta. No turbo isto pode ser modificado, veja no índice remissivo como parar o computador.

Para aumentar a rapidez de execução dos programas, deve-se desligar as opções de verificação.Mas isto só se faz quando se tem certeza de que o programa está funcionando bem. Em programas pequenos, esta diferença de rapidez não é sensível.

Exercício: 2 *Laboratório. Os programas mencionados abaixo se encontram no disco, portanto você não precisará digitá-los.*

1. *Rode agora os programas Pas05 ou Pas05_1 e experimente pará-los com uso do* **Ctrl-C ou Ctrl-Break**. *Antes, ative neles esta função, ver em parando programa no índice remissivo.*
2. *Digite o programa Pas03 e rode-o.*
3. *Mude a posição das linhas 6 e 7 do programa no Pas03 para que o programa imprima a partir de dois.*
4. *Modifique Pas05 para que imprima a partir de -5.*
5. *Modifique Pas05 para que imprima até 200.*
6. *Modifique Pas05 para que imprima uma p.a. de razão 2 com primeiro termo* $s_1 = 1$.

7. *Modifique Pas05 para que imprima uma p.a. de razão 3 com primeiro termo $s_1 = 1$.*

8. *Modifique Pas05 para que imprima até 30000. Analise o resultado. Se a tela estiver passando muito rápido pelos seus olhos, você poderá* congelá-la *com Ctrl-S e descongelá-la com qualquer outra tecla.*

9. *Altere Pas05 para que escrever os termos de uma p.g. cujo primeiro termos seja 1, a razão $\frac{1}{2}$ e com 10 termos.*

Vou fazer mais uma modificação no programa `Pas05`. Compare e veja o que foi modificado. Há mais um palavra do Pascal neste programa:

Vocabulário: 12 If, ","

1. *A vírgula é usada para separar objetos de mesmo tipo. Neste programa é usada para separar diversas variáveis de tipo* integer *mas também é usada para separar os tres ítens da mensagem que está dentro do* loop*.*

   Observe que as mensagens de tipo "texto" estão marcadas com apóstrofos no início e no final. Experimente tirar um dos apóstrofos, o final ou inicial. Se você ainda não tiver feito isto*, rode o programa para ver o resultado!*

2. **If** *A palavra inglesa "If" corresponde ao nosso condicional "Se". "If" é um comando de estrutura de controle do fluxo de um programa. Sua sintaxe é:*

   ```
   If        condicao
          Then comando;
   ```

   "condicao" tem que ser uma expressão lógica, ou melhor do tipo **Boolean***, que é o nome do* tipo **lógico** *em Pascal. Se o valor de "condicao" for* **true***, verdadeiro, então "comando" será executado. Se o valor de "condicao" for* **false***, falso, então o comando composto*

   "If condicao then comando;"

   será ignorado e Pascal irá executar o comando seguinte.

Exemplo: 7 *A soma dos termos de uma p.a..*

Os programas imediatamente anteriores fizeram listagens dos termos de progressões artiméticas ou geométricas. Vamos agora somar os termos dessas progressões. O método para fazer *somas em computação, consiste em* acumular *os valores de uma variável, como fizemos em* Pas05 *com a variável* contador*. Neste caso fomos acumulando de* um *em* um*. No próximo programa esta variável se chama* soma *e o comando composto que vai ser repetido tem o seguinte formato:*

$$soma := soma + parcela,$$

primeiro atualizamos soma *e depois atribuímos este novo valor à variável* soma*.*

Neste programa, estou usando vírgulas para separar dados na declaração de variáveis e dentro do **WriteLn***. Incluímos um* **Write-Ln** *para mostrar os dados processados e um* **ReadLn** *para que você os possa ver e se cansar de apertar uma tecla...*

```
Program Pas06;
Var
    contador, soma : Integer;
Begin
    soma := 0;
    contador := 1;
    Repeat
        soma := soma + contador;
        WriteLn(contador, ' ==> ',soma);
        contador := contador + 1;
    Until contador > 100;
    WriteLn('valor final da soma ==> ',soma);
    ReadLn;
End.
```

Exercício: 3 *Usando o condicional If .*

1. *Péssima correção em Pas06. Modifique Pas06, acrescentando uma linha entre 6 e 7 com um condicional de modo que se "contador" for maior que 20 o programa fique esperando um "enter",(use"ReadLn").*

2. *Melhorando Pas06. Ficou péssimo Pas06 com a modificação anterior... É preciso um condicional de melhor nível! Use uma nova variável,*

   "`numero_de_linha`",

   dentro do **If** *que cresça junto com contador, mas quando for maior que 20 o programa pare, "ReadLn", e* `numero_de_linha` *volte ao valor inicial 1. O "comando" If deverá ter o formato:*

if `numero_de_linha` > 20 *then*
begin
`numero_de_linha` $:= 1$;
ReadLn;
end;

O par "begin/end;" serve para criar comandos compostos. O "If" executa apenas um comando, como acima há dois, é preciso transformá-lo num "comando composto" para que "If" execute os dois.

3. *Modifique Pas06 incluindo a variável* `numero_de_linha`*, do tipo Integer, de modo que* `numero_de_linha` *va crescendo numa p.a. de razão 1 e primeiro termo 1 e o programa pare a cada bloco de 10 linhas.*

4. ** Modifique Pas06 para que os blocos mencionados anteriormente tenham 22 linhas.*

Observe que "soma" e "contador" não se encontram marcados com apóstrofos dentro do WriteLn, no programa Pas06, porque *são variáveis.*

Você deve ter observado que acrescentei uma nova variável, *soma*, na declaração e que fiz pequenas modificações essenciais no corpo do programa. A variável *soma* foi inicializada com o valor zero. Dentro do Loop o programa esta somando sucessivamente os valores de *contador* à *soma.* Modifiquei os parâmetros de `WriteLn`, incluindo soma e alguns espaços para tornar os resultados mais bonitos. Os caracteres *ASCII de 23 a 27* são pequenas setas que podem ser incluídas dentro de uma mensagem usando a função Chr(n) do TurboPascal.

Vocabulário: 13 *Caracteres ASCII, biblioteca,Chr, lst, Printer, Uses.*

ASCII *é uma sigla que significa* American Standard for Communications and Information Interchange*, que codifica os sinais usados na escrita para serem usados nas comunicações e em processamento de dados. Veja no manual do TurboPascal ASCII no índice remissivo. Hoje esta* codificação *já se encontra ultrapassada, mesmo porque a quantidade de símbolos a serem comunicados aumentou violentamente. Usam-se os códigos ASCII ainda, mas em casos bem elementares.*

biblioteca *São arquivos especiais em que se podem definir novos objetos da linguagem. Voltarei a discutir este conceito mais à frente; neste momento farei uso de duas bibliotecas do Pascal:* printer, crt. *Ver* **Uses** *mais abaixo.*

Chr *A função do TurboPascal* **Chr** *permite imprimir os caracteres codificados na tabela ASCII. Chr(24) é uma seta para cima. Rode o programa* `caracter`*; ele lhe vai imprimir na tela os caracteres ASCII. Este programa se encontra no disco.*

ClrScr *é um comando do Pascal para limpar a tela. Está definida na biblioteca CRT. Para usar ClrScr é preciso aparecer CRT na linha do* **Uses***. Ver* **Uses** *mais à frente.*

Lst *é uma constante definida na biblioteca* printer *para permitir que os dados que você mandar imprimir saiam numa impressora. Escrevendo*

WriteLn(Lst,contador,'==> ',soma);

faz os dados serem impressos na impressora em vez de aparecerem na tela. É preciso, entretanto, incluir a biblioteca **printer** *no comando* **Uses***:*

Uses printer;

no rótulo do programa, caso contrário Pascal não irá reconhecer "Lst".

Printer *é uma biblioteca do TurboPascal para criar acesso programa-impressora. Uses é a palavra do Pascal para chamar uma biblioteca e ler o seu conteúdo.*

Uses *É a palavra-chave do Pascal para indicar quais são as bibliotecas que o programa vai usar.*

Exercício: 4 *Alerta, pode haver um exercício errado nesta lista !*

1. *Elimine a linha "soma := 0;" e rode o programa pelo menos duas vezes e observe os resultados. Modifique a linha*

   $$Until \;\; contador > 100;$$

   substituindo 100 por 10000, 50000. Rode o programa três vezes e observe os resultados.

2. *Modifique o programa-exemplo anterior, de modo a somar os termos de uma p.a. de razão 2 e termo inicial* $s_1 = 4$

3. *Coloque apóstrofos em volta de "contador" e rode o programa...*
4. *Modifique o lay-out da saída de dados, faça experiências até encontrar algo pior ou melhor... Use "ClrScr" para apagar a tela depois que alguma coisa foi impressa. Use* `numero_de_linha` *para controlar a quantidade de linhas impressas na tela dentro de um "If".*
5. *Modifique o programa para calcular:*

$$\sum_{n=1}^{300} \frac{1}{n}$$

6. *Pascal descobriu algum erro no exercício anterior ?*
7. *Modifique o programa* caracter *para imprimir a tabela ASCII em papel. Este programa se encontra no disco.*

O quinto exercício da lista anterior pode estar errado. O erro - *e você deve ter recebido um aviso do Pascal* - foi um erro de *tipo de dados* porque defini *soma* como *número inteiro* e depois comecei a somar números fracionários. Leia a respeito de tipos de dados no seu livro de Pascal.

Sem maiores detalhes, quando um número não for *necessariamente* inteiro, será preciso declará-lo como **REAL**.

Para os matemáticos: aqui parece haver um exagero, ... mas não é bem assim. Pascal reconhece apenas duas grandes classes de tipos de números:

ordinais Integer, LongInt, byte, ...

ponto flutuante Real, Single,... Extended.

Use **Ctrl-F1** em cima da palavra **Integer** ou da palavra **Real** para ler as informações sobre estes tipos de dados contidas no *help* do Pascal.

Escreva na tela "Real", coloque o cursor sobre esta palavra e aperte **Ctrl-F1**. Como resultado, o *Help* do Pascal lhe vai trazer para a tela todos os tipos de dados "numéricos" que Pascal reconhece da classe *ponto flutuante*. Você também vai ver na tela a magnitude destes dados: a quantidade de memória gasta com eles.

Podemos corrigir o programa errado da lista de exercícios assim:

Exemplo: 8

```
Program Pas07;
Var
   contador : Integer;
   soma     : Real;
Begin
     soma     := 0;
     contador := 1;
     Repeat
       soma := soma + contador;
       WriteLn(contador, ' ==> ',soma);
       contador := contador + 1;
     Until contador > 100;
     WriteLn('valor final da soma ==> ',soma);
     ReadLn;
End.
```

Ao fazer as experiências do primeiro exercício, você deve ter encontrado resultados diversos. A primeira vez que o programa rodou a variável "soma" recebeu 0 e foi sendo acrescida a partir deste valor. Repetindo a experiência com um teto de 50000 para contador, apareceram na tela números negativos... porque Pascal tem um limite para números inteiros de 32767 depois do que passa ao número -32768. Esta limitação pode ser contornada até certo ponto substituindo o tipo de dado **Integer** por **LongInt**; veja com Ctrl-F1, colocando o cursor em uma dessas palavras, qual é o significado do que estou dizendo.

O uso do programa **ASCII** também lhe pode ter trazido problemas, a impressora pode ter jogado papel em branco para fora, por exemplo... procure entender o que houve.

Exercício: 5 *1. Calcule:* $\sum_{n=1}^{300} \frac{1}{n(n+1)}$

2. Calcule: $\sum_{n=1}^{300} \frac{1}{n^2}$

3. Calcule as somas:

$$\sum_{n=1}^{300} \frac{(-1)^n}{n} \quad ; \quad \sum_{n=1}^{300} \frac{1}{n} \quad ; \quad \sum_{n=1}^{300} n$$
$$\sum_{n=1}^{50000} n^2 \quad ; \quad \sum_{n=1}^{300} n^3 \quad ; \quad \sum_{n=1}^{300} \frac{\sin(n)}{n}$$

4. Faça um programa que calcule o valor de π aproximadamente, usando o perímetro de polígonos regulares convexos inscritos numa circunferência. As funções trigonométricas em Pascal são cos, sin. Com Ctrl-F1 você encontrará as outras.

5. *Faça um programa que determine o valor de n tal $|s_n - s_{n+1}| < 0.1$ onde $s_n = \frac{1}{n}$ A função* módulo, *em Pascal se chama* **Abs** .

6. *Somando sucessivamente números pares positivos se pode ter um resultado errado em TurboPascal. Verifique-o calculando* $\sum_{n=1}^{30000} 2n$.

7. *Faça um pequeno programa para escrever o valor de* $x+y; x,y :$ *Integer quando* $x = 32767, y = 200$. *Explique o resultado.*

8. *Repita a experiência escrevendo dentro do Write diretamente uma soma de inteiros sem declaração de variáveis:*

   WriteLn(32767 + 200);

   por exemplo. Analise e justifique o resultado. Use Ctrl-F1 sobre a palavra Integer *para compreender o que está acontecendo.*

9. *Veja qual é diferença, em Pascal, dos seguintes símbolos:*

   $$32.776; 32,767; 32767.$$

   Como é que Pascal entende cada um dos três.

10. *Modifique o programa Pas07 para que sejam listados na tela os valores das funções*

    $$f(x) = \frac{1}{x^2+1} \; ; \; f(x) = \frac{2x}{x^2+1}$$

    para $x \in 0.1, ..., 1$ *com passo* 0.1. *Sugestão :*
    Na declaração **Var** *substitua: "soma" por "x"*
    e inclua deltax: " x,deltax : Real;"
    substitua:

"$soma := 0;$"	*por*	"$x := 0; deltax := 0.1$"
"$soma := soma + contador;$"	*por*	"$f := 1/(x * x + 1);$"
"$contador := contador + 1$"	*por*	"$x := x + deltax;$"
"$Until\ contador > 100$"	*por*	"$Until\ x > 1;$"

    A primeira linha[2]*: "$x := 0; deltax := 0.1$" deve vir antes do* **Repeat**, *por que ?*

11. *Crie um condicional que controle a saída de dados na tela usando*

[2]em Pascal não há o conceito de linha, mas os programas se escrevem ao longo de linhas...

`numero_de_linha`

para que apareçam apenas 10 valores da função de cada vez.

12. *Reaproveite o programa acima para "varrer" os valores das sucessões abaixo desde o menor valor possível para k até que $k > 100$:*
    a) $s_n = \frac{1}{n+1}$ *b)* $s_n = \frac{n+1}{n+2}$ *c)* $s_n = \frac{n-1}{n+1}$ *d)* $s_n = \frac{n-1}{n+10}$
    e) $s_n = \frac{3n+1}{3n-1}$ *f)* $s_n = \frac{3n+30}{3n+1}$ *g)* $s_n = \frac{5n+1}{2n+30}$ *h)* $s_n = \frac{5n+30}{2n+1}$

13. *Calcule a soma das somas parciais para cada uma das sucessões acima.*

Comentando alguns exercícios.

O "cálculo" de π:

Na solução abaixo há comandos novos ... Dentro do **Writeln**: os "dois pontos". Experimente tirá-los junto com os números que lhe estão justapostos e compare o resultado.

Na expressão "**WriteLn**(x:14:4);", o número 14 indica que o conteúdo da variável numérica x deve ocupar 14 espaços, 4 significa que "4 casas decimais após a vírgula", devem ser usadas *para mostrar o valor de x* no *dispositivo de saída de dados*. O uso deste comando não tem nenhuma conseqüência relativamente à precisão interna para efetuar as operações.

O programa também usa uma constante do Pascal "pi". Aqui há um óbvio furo lógico... qual?

Recentemente π foi calculado com 50 bilhões de casas decimais exatas, ver [2].

```
{**  Programa para simular o calculo de pi **}
program Pas7\_1;

Var
  i : integer;
  y : Real;
Begin
     i := 3;
     Repeat
       y := i*sin(pi/i);
       WriteLn(i,'   ',y:14:4);
       i := i + 1;
```

```
    Until i > 1000;
    ReadLn;
End.
```

Acima, fiz referência aos nomes das funções trigonométricas e da função valor absoluto. Se você escrever *uma palavra qualquer* e colocando o cursor sobre ela apertar **Crtl-F1**, TurboPascal vai lhe dizer se tem informações sobre ela chamando a tela inicial do "Help" se não a conhecer. Nela você vai ver em destaque, entre outras, a palavra **functions** que você poderá selecionar com as setas. Fazendo-o, TurboPascal lhe apresenta uma sucessão de telas em que se encontram os nomes das funções que ele reconhece. Uma nova seleção pode ser feita com as setas para receber mais informações sobre a sintaxe de uma determinada função . Experimente.

O programa `Pas07_1` usa o número π para calculá-lo... na verdade é um exercício do uso do Repeat/Until. As funções trigono-métricas em Pascal só aceitam "ângulos" na forma de radianos. Não podemos escrever $\sin(90^o)$, pois Pascal não entenderia. Temos que escrever os ângulos no formato de *radianos*, neste caso $\sin(\frac{\pi}{2})$ ou $\sin(1.57)$, aproximadamente. A constante π está definida dentro da linguagem Pascal com o valor aproximado até 13 decimais.

O *furo lógico* no cálculo do π se encontra no uso do próprio π dentro do programa. Para encontrá-lo experimentalmente, teríamos que produzir um programa que usasse a definição geométrica desta constante...

Os comandos " : " são para formatação de saída de dados. Indicam quantas casas decimais queremos que apareçam. Veja mais a este respeito no livro de Pascal.

Exemplo: 9 *Soma de inteiros em Pascal*
O programa

```
Program Pas08;
Begin
    WriteLn(32768 + 32768);
    ReadLn;
End.
```

tem como resultado 65536 passando do limite de inteiros do TurboPascal. Mas o programa

```
Program Pas09;
Var x,y : Integer;
Begin
    x := 32767; y := 1;
    WriteLn(x + y);
    ReadLn;
End.
```

vai ter como resultado -32767 *porque o proxímo número na tabela de inteiros do TurboPascal, depois de* 32767*, é este. Ver* Integer *com Ctrl-F1. No primeiro caso, Pascal automaticamente considera o tipo mais amplo de dados, os números são considerados LongInt. TurboPascal considera a classe mais ampla de dados se você não estabelecer restrições.*

1.2.1 Juros

... os juros da "dívida" externa.

No próximo programa usamos o conceito de *biblioteca* de que já tratamos. Você só vai ter grande utilidade no uso de *bibliotecas* quando os seus programas começarem a ficar maiores do que a tela do computador... O TurboPascal lhe é vendido com um sistema de bibliotecas em que várias funções já se encontram prontas para uso. É esta a utilidade delas. *Pequenos programas que executam tarefas específicas são guardados dentro de bibliotecas depois de transformados em* **procedures***, as sub-rotinas*, é uma forma de fazer a linguagem crescer com comandos típicos do usuário. Analise minha biblioteca *Tipos*, por exemplo, para ver como se constrói uma biblioteca e estude o seu conteúdo. Neste momento talvez *Tipos* lhe pareça muito difícil de entender, mas vá se acostumando a lê-la e aos poucos ela lhe parecerá natural.

É um exercício difícil a leitura de programas feitos por outras pessoas e a razão é bem simples: *programas contêm atitudes subjetivas do programador.* Por outro lado, é um exercício importante a leitura de programas feitos por outras pessoas a fim de ganhar a experiência que outros programadores já têm. Supere as dificuldades e procure ler a *Tipos*. Leia também sobre o assunto *Biblioteca* no seu manual do TurboPascal, entretanto somente no final do capítulo é que você começará a sentir a real necessidade de trabalhar com bibliotecas.

O comando do TurboPascal para dar-lhe acesso às bibliotecas é *Uses*. Veja sua utilização dentro do programa abaixo que é absolutamente semelhante a todos os programas que fizemos até agora: *todos são programas de varredura inteira*, ou seja, a variável vai percorrer um subconjunto dos números inteiros.

Exemplo: 10 *Programa: Evolução da "Dívida" Externa.*
Este programa tem as seguintes variáveis:
capital, anoinicial, anofinal.
Parte da hipótese de que a chamada dívida externa teria o valor de 100 bilhões de dólares em 1972, capital := 100 e segue fazendo acréscimos constantes de 15 % a.a. e ao mesmo tempo subtraindo 12 % a.a. do chamado serviço da dívida*. Imprime anualmente qual é o valor de* nossa dívida*. Observe que anoinicial vai sendo atualizado para ser o* <u>*anoatual*</u>*, assim economizamos uma variável no programa.*

```
Program Pas10;
Uses Crt;
Var
   capital            : Real;
   anoinicial, anofinal: Integer;
Begin
     ClrScr;
     anoinicial:= 1972;
     anofinal  := 1995;
     capital   :=  100;
     Repeat
           capital := capital*(1 - 0.12);
           capital := capital*(1 + 0.15);
           WriteLn(anoinicial,' => ' ,capital:10:0,'Bilhoes');
           anoinicial := anoinicial + 1;
     Until anoinicial > anofinal;
     ReadLn;
End.
```

Dentro do **WriteLn** *os ":" que se encontram depois da variável capital ":10:0" é o* comando de formatação de saída de dados *que já vimos anteriormente.*

:10 significa que desejamos que sejam impressos 10 algarismos significativos, enquanto que :0 indica que não deve ser impresso nada depois da vírgula.

Exercício: 6 *Biblioteca e entrada de dados*

1. *Retire a linha Uses Crt; do programa Pas10 e rode-o para ver o que acontece. Analise o resultado: o programa deixou de reconhecer* **ClrScr**, *por que?*

2. *Use o comando* **ReadLn** *para modificar o ano inicial e o capital*

   ReadLn(anoinicial); Então você não precisará incluir a linha **anoinicial := 1972;**. *Pascal vai parar e esperar que você diga o valor de anoinicial. Tente.*

3. Melhore sua entrada de dados com uma mensagem.

   O comando: ReadLn(anoinicial);

   faz com que Pascal pare e aguarde que você lhe forneça um valor pelo teclado.

   Mas você terá pela frente um cursor piscando e nenhuma sugestão sobre o que fazer.

   Acrescente antes do ReadLn um comando WriteLn('Qual eh o ano inicial'); *em que você informa ao usuário do programa que tipo de dado é esperado.*

4. *Escreva uma saída de dados ao final, depois do loop, para o que programa diga o que fez, por exemplo,* incitando o usuário a acreditar *que é perfeitamente justo e possível pagar a "dívida externa"...*

5. *Faça um programa para atualizar a sua caderneta de poupança. Não se esqueça de uma mensagem incentivando-o a poupar cada vez mais...*

   Use WriteLn, ReadLn para fornecer ao programa o valor inicial de sua poupança. Antes do Repeat ...

   Use WriteLn, ReadLn para fornecer ao programa o índice oficial de inflação .

   Use mesinicial *e* mesfinal *para definir o período de sua poupança.*

   Como você não é bobo, inclua também o índice extra-oficial de inflação para que seu programa calcule o que você tiver perdido enquanto estava poupando.

1.3 Sucessões

1.3.1 Números reais

Uma classe fundamental de problemas com que nos defrontamos com freqüência consiste em determinar um número, por exemplo, encontrar uma raiz de uma função: encontrar x ; $f(x) = 0$.

Observação: 7 *Números irracionais*

Os números são de diversos tipos: naturais, inteiros, racionais, reais ou complexos. *Neste livro não tratarei de nada envolvendo números complexos diretamente. Os números racionais e os números inteiros são os aqueles com que podemos trabalhar com uma linguagem como Pascal. Em computação mais avançada é possível trabalhar com alguns números irracionais, os chamados números algébricos, porque eles são definidos usando-se polinômios que por sua vez são definidos com um número finito de números racionais. Tais números se chamam* números algébricos sobre os racionais*, porque os coeficientes dos polinômios são números racionais.*

Os números reais são limites de sucessões. Vamos ver superficialmente *como definir números reais e como tratar com eles computacionalmente: sob forma de aproximação, obviamente.*

Os exemplos mais comuns de números reais, que não são racionais, são $\pi, e, \sqrt{2}$; *corremos o risco de imaginar que há poucos números* não racionais...

Embora com um *furo lógico*, vimos um método para determinar π através dos valores sucessivos dos perímetros de um polígono regular convexo de n lados; $n = 1, 2, ..., n$.

Podemos mostrar, usando métodos geométricos, que a sucessão de tais perímetros P_n é limitada superiormente se os polígonos estiverem inscritos num mesmo círculo, sendo, ao mesmo tempo, crescente: *uma sucessão convergente.* O limite desta sucessão é o número real $2\pi r$, sendo r o raio do círculo em que os polígonos estiverem inscritos.

Todo número real é o limite de uma sucessão convergente. Com um pouco mais de elaboração algébrica, podemos enunciar a recíproca: toda sucessão convergente define um número real. Duas sucessões que tenham o mesmo limite são equivalentes no mesmo sentido das frações equivalentes quando definem o mesmo número racional.

Isto torna *sucessões convergentes* equivalentes aos números reais e mostra a importância das sucessões como um instrumento de aproximação.

Por outro lado, num computador só podemos lançar dados, informações, finitas. As *sucessões finitas* são assim um método para representar números, medições diversas que possamos fazer, enfim *representar grandezas aproximadamente. Elas são um método de discretização.* Estas sucessões finitas são aproximações das sucessões.

Quando escrevemos $\lim_n s_n$ estamos nos referindo ao número real definido pela sucessão $s = (s_n)_{n\in N}$ e poderíamos naturalmente escrever mais abreviadamente: $lim(s)$.

A definição que usualmente aparece nos livros de Cálculo para limite tem um defeito intrínseco:

$$L = lim(s) = lim_n s_n \iff \tag{1.1}$$

$$\iff \forall\ \epsilon\ \exists n_o\ ;\ n > n_o\ \Rightarrow\ |s_n - L| < \epsilon \tag{1.2}$$

em que L é o limite de s. O *defeito* se encontra em que geralmente não sabemos escrever L, em geral apenas sabemos provar sua existência. Vou descrever na próxima seção um modo implícito de definir limite que, além do mais, se presta de forma natural para ser aplicado em computação, sobretudo porque, na grande maioria das vezes, não nos interessa (*ou não sabemos calcular*) o valor exato do *limite L* mas sim uma sua *aproximação*. Disse "*interessa*"? Com freqüência não nos é possível obter o valor de L mas *apenas* seu valor *aproximado*: é o caso dos números irracionais que conhecemos.

Discretização e sucessões

As sucessões representam o principal método de discretização de fenômenos que temos, no fundo o único.

1.3.2 O critério de Cauchy

Ao longo da história da Matemática, três métodos foram desenvolvidos com sucesso para definir os números reais. Um deles foi descrito acima, via sucessões. Outro se deve ao matemático G. Cantor, que é muito geométrico, quanto à sua natureza e se assemelha muito ao método desenvolvido por Dedekind, chamado de *cortes de Dedekind.* Os três métodos são equivalentes.

O matemático Dedekind, para construir os números reais, definiu o método dos *cortes de Dedekind*, que tem uma explicação intuitiva muito simples:

Observação: 8 *Cortes de Dedekind*

O conjunto dos números reais se confunde com uma reta, conseqüentemente, para encontrar com boa aproximação um número para o qual se tenha uma definição formal *tudo que se tem que fazer é ir "cortando" a reta acima ou abaixo do número que se procura usando-se a* definição formal *para estabelecer desigualdades.*

O resultado prático deste método consiste em encontrar um intervalo $[a, b]; |b - a| < \varepsilon$ *em que* ε *é precisão com que se calculou* $x \in [a, b]$. *Fazer* artimética *usando este método é chamado de* <u>aritmética intervalar</u>. *Não vou adotar neste livro* esta aritmética, *prefiro o método das sucessões . Os dois métodos são equivalentes, entretanto a artimética intervalar, ao final de contas, trabalha com duas sucessões porque os intervalos têm dois extremos...*

O método de Cantor, chamado dos intervalos encaixados, *é muito semelhante ao de Dedekind e também conduz à* aritmética intervalar.

O método das sucessões leva o nome do matemático Cauchy e é o que prefiro usar. Vou fazer uma referência muito superficial ao método. Mais detalhes você poderá encontrar num bom livro de Cálculo.

Vamos escrever uma expressão equivalente à definição de limite de sucessões acima.

Para isto, consideremos dois valores da sucessão que satisfaçam a definição 1.1:

$$\forall\, \epsilon\ \exists n_o\ ;\ n > n_o \ \Rightarrow\ |s_n - L| < \frac{\epsilon}{2}$$

$$\forall\, \epsilon\ \exists n_o\ ;\ m > n_o \ \Rightarrow\ |L - s_m| < \frac{\epsilon}{2}$$

pela desigualdade triangular, se somarmos as duas expressões em módulo teremos:

$$\forall\, \epsilon\ \exists\ n_o; n, m > n_o \ \Rightarrow\ |s_n - s_m| < \epsilon \tag{1.3}$$

A implicação recíproca é um pouquinho mais elaborada, *afinal deverá aparecer L ....* mas é verdadeira, o que significa que o critério de Cauchy equivale à definição de limite.

Critério de Cauchy

Esta segunda definição é usualmente chamada de *Critério de Cauchy*. Como são equivalentes, pode qualquer uma servir de definição para o conceito de limite. No *Critério de Cauchy*, comparando com a outra definição, podemos ver que tanto s_n como s_m são aproximações para o limite $L = lim(s)$.

Quando uma sucessão s for convergente, ou satisfizer ao Critério de Cauchy, se expressa isto dizendo que *s é de Cauchy.*

A definição "Critério de Cauchy" não é *computacional*, uma vez que ela depende de um *quantificador existencial*, uma operação lógica de alto nível que ainda é impossível, de acordo com os nossos conhecimentos, de se produzir computacionalmente. Isto é assunto para Inteligência Artificial. Mas podemos trabalhar com uma versão *estocástica* do Critério de Cauchy: *usando a rapidez e precisãodum computador para fazer um levantamento de dados com grande quantidade de amostras, testes obtidos com a desigualdade do critério de Cauchy, e, finalmente, avaliando probabilisticamente a convergência.* É o conteúdo do próximo programa. É isto que chamo de:

$\boxed{\textbf{Critério Estocástico de Cauchy}}$

O próximo programa vai realizar o Critério Estocástico de Cauchy, mas precisamos de algumas definições:

Vocabulário: 14 If/Then/Else, Begin, End;, Function, Procedure.

Begin - End; *é um par de "parenteses" para indicar ao Pascal a presença de um comando composto. Quando Pascal encontrar um* **If***, ele vai executar o comando que se encontra depois do* **Then** *. Mas se quisermos que ele execute mais de um, como é o caso no programa abaixo, temos que envolvê-los com* **Begin , End;**

If-Then-Else *Forma alternativa de estrutura condicional.*

If condicao

Then processo1

Else processo2*;*

ausência de ponto e vírgula *Observe a ausência de "ponto e vírgula"* antes *de* **Else***.*

Se `condicao`	*for verdadeira,*
	`processo1` *será executado,*
caso contrário	
	será executado `processo2`*.*

Veja que `processo1` *ou* `processo2` *podem ser comandos compostos e portanto em ambos os casos se pode ter um par* **Begin-End;** *envolvendo* `processo`X*. Entretanto antes do* **Else** *não pode haver ";" portanto* `processo1` *não pode terminar com ";". A razão disto se encontra em que você pode precisar de fazer uma cadeia de* **If***s sem* **Else***s. Quando uma sentença do tipo* **If** *não terminar com ";" significa que virá um* **Else***. Uma outra maneira de explicá-lo: ";" termina um comando, portanto uma sentença do tipo* **If** *ou termina com ";" ou continua a partir do* **Else***.*

function *é a palavra-chave do Pascal para identificar que se inicia a definição de uma função. Em Pascal há:*

- **function** *uma rotina que produz um valor e*
- **procedure** *uma rotina que silenciosamente produz uma modificação ou um conjunto de modificações.*

São os dois métodos para definir rotinas*. No programa que se segue você tem um exemplo de como usar* **function***. Uma* **function** *é um pequeno programa que produz um resultado, muito a semelhança de como entendemos* funções *em Matemática. No programa abaixo você encontra exemplos de* functions*.*

procedure *São , como as* functions*, pequenos programas mas com a diferença de que não produzem nenhum resultado* explicito*. A tradução de* procedure *é* rotina*. Há linguagens de processamento que não fazem distinção entre estes dois tipos de* rotinas*,* `C, LISP, Python` *por exemplo, onde todas as rotinas são do tipo* função *. Em Maple há apenas* procedures *que se chamam* **proc***. Grandes programas são feitos de pequenas operações. Estas pequenas operações quase sempre* nascem *como pequenos programas que são testados e viram depois* procedures *e são colocadas em bibliotecas. Veja no índice remissivo* top-down*. Veja a biblioteca* Tipos*, analise em particular as procedures: Apeteco, Apito. No programa abaixo você encontra exemplos de* procedures*.*

Observação: 9 `Procedure` *é a metamorfose final duma rotina*

Já mencionei o livro de Polya, "How to solve it" mostrando como se fazem algoritmos. O livro de Polya é uma coleção de exemplos mostrando como se resolvem problemas. Vou agora construir mais um exemplo: o teste estocástico de Cauchy. Se o livro de Polya não resolve a questão "como se resolve isto", seria pretensioso esperar que mais este *exemplo fosse tão definitivo. Não espero, mas sempre acredito que um bom exemplo ajude a compreender um pouco mais a construção de um* método.

Algumas regras gerais precisam ser enunciadas. Quando você for construir um programa para resolver uma grande tarefa, o primeiro passo se constituí de uma análise dos diversos tipos de operações envolvidas. Nem sempre esta análise se pode fazer em forma completa logo no início, muitas vezes o projeto *precisa ser reavaliado e refeito*[3].

A primeira etapa de um planejamento na produção de um programa, um grande programa, é a construção de pequenos programas que executam pequenas tarefas dentro do projeto. *Pequenos programas que você roda assim como cada membro da equipe roda e verifica se de fato a operação desejada está sendo executada em todos os casos possíveis. Uma equipe boa sempre tem um grupo só para fazer testes.*

Quando os pequenos programas *estão funcionando, eles viram* procedures *e o seu conjunto vai compor uma biblioteca. Foi assim que a* Tipos *e as nossas outras bibliotecas foram feitas, aos poucos...*

Você verá a seguir os passos desta metamorfose em que pequenos programas viram procedures ou functions.

As etapas do teste estocástico de Cauchy são :

1. Uma entrada de dados para dizer ao programa a partir de onde deve começar seu levantamento de dados e um limite, um teto, onde deve parar.
2. Uma rotina (leia *procedure*) para contar os elementos que satisfazem a condição de Cauchy: o levantamento de dados.
3. Uma rotina para fazer o cálculo percentual: $\frac{\text{os que satisfazem}}{\text{o total analisado}}$.
4. Uma rotina de decisão em função de um percentual escolhido para responder ao usuário se "*o programa acha*" que o teste de Cauchy foi positivo ou negativo.

A idéia geral é realmente bem simples, como parece. É semelhante ao que ocorre com um levantamento de opinião, em que um certo número de pessoas são entrevistadas e depois se conclui que "a maioria acha que o presidente está cumprindo com as promessas de campanha"... apenas o nosso levantamento é sério, não faremos nenhum escolha viciada dos entrevistados.

Teste de Cauchy via procedures isoladas

A concepção de trabalho começa com o *corpo do programa* funcionando como um planejamento:

```
Begin
   Entrada_de_Dados;
   Repeat
```

[3] Leia a respeito em "top-down", ver índice remissivo.

```
        If Testa_Cauchy(n,teto) Then
        Responde_Afirmativo(numerador,denominador,f(teto))
        Else
            Responde_Negativo(numerador,denominador,f(teto));
            teto := 2*teto;
            deltax := deltax/2
    Until Decide_Parar(resposta);
End.
```

É um exemplo de **Top-Down design**, planejamento de cima para baixo, ver "*top-down design*"

Veja que inventei os nomes que preciso de modo que a redação do programa traduzisse o objetivo desejado. Como estou consciente de que a língua inglesa às vezes pode ter um efeito bloqueador, vou traduzir para o Português o planejamento acima:

```
Inicio
    Entrada_de_Dados;
    Repete
        Se Testa_Cauchy(n,teto) entao
            Responde_Afirmativo(numerador,denominador,f(teto))
            senao
            Responde_Negativo(numerador,denominador,f(teto));
        teto := 2*teto;
        deltax := deltax/2;
    Até que Decide_Parar(resposta) ;
Fim.
```

Vamos analisar o planejamento feito.

`Decide_Parar` é uma função lógica; ela deverá opor ao usuário uma pergunta do tipo: "Você está satisfeito com os testes?"

Enquanto a resposta do usuário for "N", ela deverá produzir o valor **FALSE** dentro do **Until** fazendo o programa voltar a rodar.

`Testa_Cauchy` é uma função lógica. Observe sua posição depois do "If", tudo que vier depois de um "If"; tem que ser de *tipo Boolean*. Assim `Testa_Cauchy` deve produzir um valor *true* ou *false* e obviamente que sua estrutura deve ser semelhante a uma "bateria de testes de Cauchy" no formato da equação 1.3. Antes de escrever uma procedure, vou escrever o seguinte programa que chamarei de *teste* para verificá-la:

```
Program teste;
Var n,k, numerador, denominador, teto : LongInt;
    deltax : Real;
```

```
    Resultado : Boolean;
Function f(n:LongInt):Real;
Begin
    f := ln(n);
End;
Begin
    WriteLn('inicio de operacoes');ReadLn(n);
    WriteLn('teto de operacoes');ReadLn(teto);
    numerador := 0; denominador := 0;
    While n ¡ teto do
    Begin
        While (Abs(f(n)-f(k))¿ deltax) and (n ¡ teto) do
        Begin Inc(n);Inc(denominador);
            k := Round((n+teto)/2);
        End;
        While (Abs(f(n)-f(k))¡= deltax) and (n ¡ teto) do
        Begin Inc(n);Inc(numerador);Inc(denominador);
            k := Round((n+teto)/2);
        End;
End;
    ** Resultado eh uma variavel logica ***
    Resultado := ((numerador/denonminador) ¿ 0.9 );
    If Resultado Then
        WriteLn(' Provavelmente tem limite = ', f(teto))
    Else
        WriteLn(' Provavelmente nao tem limite ');
End.
```

Depois de rodar este programa várias vezes com sucessões que têm limite e com sucessões que não têm limite, conclui que o teste está funcionando corretamente e assim vou transformá-lo na função `Testa_Cauchy`:

```
Function Testa_Cauchy(n,teto : LongInt): Boolean;
Begin
   While  n < teto   do
   Begin
     While (Abs(f(n)-f(k))> deltax) and (n < teto) do
       Begin Inc(n);Inc(denominador);
       k := Round((n+teto)/2);
       End;
     While (Abs(f(n)-f(k))<= deltax) and (n < teto) do
       Begin Inc(n);Inc(numerador);Inc(denominador);
       k := Round((n+teto)/2);
       End;
   End;
   Testa_Cauchy := ((numerador/denominador) > 0.9 );
End;
```

Veja a metamorfose de que já falei. O programa testado se transformou numa `function` que será implantada numa biblioteca adequada de onde será chamada.

Observe como termina a função *atribuindo* o valor lógico

$$\texttt{(numerador/denonminador)} > 0.9$$

ao nome da função:

$$\texttt{Testa\_Cauchy} := ((numerador/denominador) > 0.9);$$

Pascal avalia uma expressão como "$4 < 3$" dando-lhe o valor "false" e avalia uma expressão como "$4 < 5$" lhe dando o valor "true". Assim o valor de "`(numerador/denonminador) > 0.9`" será "true" ou "false".

Observação: 10 *Valores lógicos, funções lógicas*

A função lógica do Pascal "$>$", responde com "true" ou "false" quando colocada entre dois números:

Vou insistir neste ponto. A expressão

$$x > 3;$$

em Pascal, não é uma expressão matemática e sim um comando do Pascal formado pelo operador lógico $>$ *colocado entre dois* parâmetros. *Se você escrever apenas a expressão acima, Pascal vai reclamar indicando um erro de sintaxe, como se você, no meio de uma conversação com amigos, dissesse uma palavra fora do contexto. Os demais se ririam e diriam:* parece que enlouqueceu.... *Pascal é mais educado e diz:* erro de sintaxe.

Como $x > 3$ é **True** *ou* **False**, *uma forma legal de usar-se esta expressão será dentro de uma atribuição, como acima:*

$$\texttt{Testa\_Cauchy} := (x > 3);$$

que é então uma frase dentro do contexto.

De modo análogo, escrevo um programa "teste" para construir e testar a procedure `Responde_Afirmativo` que deve receber os valores "numerador", "denominador", "f(teto)" e exibir uma mensagem com a fração $\frac{numerador}{denominador}$ e o limite provável "f(teto)".

Veja o resultado depois da *metamorfose*:

```
Procedure Responde_Afirmativo(numerador,denominador:LongInt;
limite:Real);
   {*** se a probab. for alta  ****}
Begin
    ClrScr;
    Write('Para o valor inicial e ');
    WriteLn(' a aproximacao escolhidos ');
```

```
    Write('a estimativa de limite');
    WriteLn(' aproximado: ',limite:5:5);
    Write('com probabilidade de  ');
    WriteLn(100*(numerador/denominador):2:2,' % ' );
End;
```

A procedure `Responde_Negativo` é semelhante, posso reeditar a anterior invertendo as mensagens:

```
Procedure Responde_Negativo(numerador, denominador : LongInt;
limite : Real);
Begin
   {*** se for baixa a probabilidade... ****}
    ClrScr; GotoXY(01,10);
    Write('teto ultrapassado');
    WriteLn('condicao de Cauchy nao satisfeita.');
    Write('Probabilidade de ter limite: ');
    WriteLn(100*(numerador/denominador):2:2,' % ' );
    WriteLn('Ultimo valor encontrado: n= ',
numerador,' -> ', limite:8:10);
End;
```

O programa que estou construindo fica agora assim:

```
Program Cauchy;
Uses Crt,tipos;
Var Resposta:Char;
    denominador,numerador,n,k,teto : LongInt;
    deltax : Real;

Function f(n:LongInt):Extended;
Begin
    f := (n*n)/(3*n*n + n + 1);
End;

Procedure Entrada_de_Dados;
Begin
  ClrScr;
  Write('Primeiro indice n = ');  ReadLn(n);
  Write('Precisao  deltax = ');  ReadLn(deltax);
  Write('Teto de operacoes, teto=');  ReadLn(teto);
  k:= n+Round(teto/2); numerador := 0; denominador:=0;
  Apito; Apeteco;
End;

Function Testa_Cauchy(n,teto : LongInt): Boolean;
Begin
   While  n < teto   do
   Begin
     While (Abs(f(n)-f(k))> deltax) and (n < teto) do
       Begin Inc(n);Inc(denominador);
       k := Round((n+teto)/2);
       End;
     While (Abs(f(n)-f(k))<= deltax) and (n < teto) do
       Begin Inc(n);Inc(numerador);Inc(denominador);
```

```
       k := Round((n+teto)/2);
       End;
   End;
   Testa_Cauchy := ((numerador/denominador) > 0.9 );
End;

Procedure Responde_Afirmativo(numerador,denominador:LongInt;
limite:Real);
   {*** se a probab. for alta  ****}
Begin
    ClrScr;
    Write('Para o valor inicial e ');
    WriteLn(' a aproximacao escolhidos ');
    Write('a estimativa de limite');
    WriteLn(' aproximado: ',limite:5:5);
    Write('com probabilidade de  ');
    WriteLn(100*(numerador/denominador):2:2,' % ' );
End;
Procedure Responde_Negativo(numerador,denominador : LongInt;
limite : Real);
Begin
   {*** se for baixa a probabilidade... ****}
    ClrScr; GotoXY(01,10);
    Write('teto ultrapassado');
    WriteLn('condicao de Cauchy nao satisfeita.');
    Write('Probabilidade de ter limite: ');
    WriteLn(100*(numerador/(numerador+denominador)):2:2,' % ' );
    WriteLn('Ultimo valor encontrado: n= ',teto,' -> ', limite:8:10);
End;

Function Decide_Parar(resposta : Char):Boolean;
Begin
   Write('repetir os testes < s >, caso contrario < n > ');
   Readln(resposta); resposta := UpCase(Resposta);
   Decide_Parar := (resposta = 'N');
End;

Begin
 Entrada_de_Dados;
 Repeat
 If Testa_Cauchy(n,teto)  Then
    Responde_Afirmativo(numerador,denominador,f(teto))
 Else
    Responde_Negativo(numerador,denominador,f(teto));
 teto := 2*teto;
 deltax := deltax/2
 Until Decide_Parar(resposta) ;
End.
```

Se as procedures e functions do programa forem transferidas para uma biblioteca de nome `Calculo`, o programa ficaria assim:

```
Program Cauchy;
Uses Crt,tipos,Calculo;
```

```
Var Resposta:Char;
    denominador,numerador,n,k,teto : LongInt;
    deltax : Real;
Begin
 Entrada_de_Dados;
 Repeat
  If Testa_Cauchy(n,teto) Then
    Responde_Afirmativo(numerador,denominador,f(teto))
  Else
    Responde_Negativo(numerador,denominador,f(teto));
 teto := 2*teto;
 deltax := deltax/2
Until Decide_Parar(resposta) ;
End.
```

Veja que o *algoritmo* não mudou; tudo o que fiz foi transferir as *procedures* e *functions* para a biblioteca `Calculo` e colocar o nome desta biblioteca dentro do **Uses**.

Este programa foi exaustivamente testado. Aplicado na sucessão fracamente crescente $s_n = Ln(n)$, indica que ela não converge porque as sucessivas probabilidades decrescem rapidamente. A função que define a sucessão cuja convergência se deseja testar é função f.

A observação feita sobre transferir as procedures e funções para uma biblioteca chamada "`Calculo`" é bem procedente. Uma procedure como `Decide_Parar` pode ser usada em muitas situações semelhantes em que vários testes serão feitos e o usuário precisa ser consultado se está satisfeito com os resultados. Se a consulta ao usuário for feita em palavras mais secas: "Parar ? (S/N)", esta procedure pode ser aplicada em muitos lugares.

De forma semelhante se pode fazer maior *abstração* relativamente às demais procedures para que possam ser usadas sempre que uma resposta negativa ou positiva deva ser dada com dois parâmetros numéricos.

Mas o lugar apropriado para estas procedures seria na `Tipos` e não na `Calculo`. Por que? Também um nome melhor lhe poderia ser dado: *Sim_Nao* que traduz melhor o que se espera dela como resposta: um `Sim` ou um `Não`.

Exercício: 7 *Critério estocástico de Cauchy*

1. *Transforme em programas as* procedures Apeteco *e* Apito *e teste-as.*

2. *Escreva um programa que leia "s" ou "n" do teclado e caso tenha lido "s" imprima:*

   ```
   Voce respondeu SIM
   ```

   e caso tenha lido "n" imprima:

   ```
   voce respondeu NAO
   ```

   Compare o seu programa com rotina Apeteco *definida na biblioteca* Tipos.

3. *Transforme o programa construído acima em uma procedure* Sim_Nao *e inclua na biblioteca* Calculo *e depois altere o programa* Cauchy *para usá-la em lugar de* Decide_Parar.

4. *Substitua na definição da função do programa Cauchy o valor da função f pelo valor da série:* $\sum_{n=1}^{\infty} \frac{1}{n^2}$ *e calcule numericamente o valor da mesma.*

5. *Verifique estocasticamente se* $\sum_{n=1}^{\infty} \frac{1}{n}$ *converge.*

6. *Defina verdadeiro como maior ou igual a 95 % e modifique o programa anterior considerando este nível de verdade para que o programa emita um diagnóstico simplificado dizendo se a sucessão:*

   (a) *provavelmente converge;*

   (b) *provavelmente não converge;*

   indicando a probabilidade encontrada em cada caso.

7. *[4] *Analise o tempo gasto contra a precisão dos resultados e tire conclusão quanto à eficiência do método.*

8. *Aplique o programa Cauchy às sucessões abaixo e compare o resultado com o teste formal de Cauchy para* $\epsilon = deltax$.

a) $s_n = \frac{1}{n}$	b) $s_n = \frac{n+1}{n}$	c) $s_n = \sum_{k=1}^{n} \frac{1}{k^2}$
d) $s_n = (-1)^n$	e) $s_n = sen(n)$	f) $s_n = cos(n)$
g) $s_n = cos(\frac{n\pi}{2})$	h) $s_n = \frac{cos(n)}{n}$	i) $s_n = \frac{n^2+1}{n}$
j) $s_n = \frac{n}{n^2+1}$		

[4] Exercícios marcados com "*" são de natureza teórica, decida se lhe interessa fazê-los.

Comentando os exercícios

1. No primeiro exercício, você pode acelerar o cálculo memorizando os valores de uma soma até um determinado índice antes de calcular a próxima soma e então usar o valor memorizado começando a soma a partir do índice sucessor.

2. Aqui há um erro de sintaxe matemática, $\sum\limits_{n=1}^{\infty}\frac{1}{n}$ não converge por que é uma constante, melhor, é um símbolo que representa uma constante: o limite de uma soma, se ela existir. A maneira correta de falar é " $\sum\limits_{n=1}^{N}\frac{1}{n}$ converge ?".

Nas próximas seções, estudaremos com detalhes somas e em particular *Somas de Riemann*.

1.4 Calculando somas

Exemplo: 11 *Verificando se uma série é convergente.*

No último exercício do bloco anterior, você encontra uma sucessão cujo termo geral, a equação, *é definida por uma soma. Dizemos* "soma parcial", *é assim que se deve entender uma série: uma sucessão de somas parciais. A expressão*

$$\sum_{k=1}^{\infty}\frac{1}{k^2}$$

representa o limite da sucessão

$$s_n=\sum_{k=1}^{n}\frac{1}{k^2}$$

se este limite existir. Repetindo, porque é importante que isto fique bem claro, uma série, se for convergente, tem um limite que é representado pelo símbolo à direita:

$$\lim_{n}\sum_{k=1}^{n}\frac{1}{k^2} = \sum_{k=1}^{\infty}\frac{1}{k^2}. \tag{1.4}$$

Assim, a teoria envolvendo as séries nada tem de novo relativamente à das sucessões. Séries são sucessões cujos termos gerais têm por equações somas parciais. Na equação acima tem-se o limite da sucessão das somas parciais de uma série:

$$\begin{aligned}\lim_{n} s_n &= \lim_{n}\sum_{k=1}^{n}\frac{1}{k^2} = \sum_{k=1}^{\infty}\frac{1}{k^2}.\\ s_n &= \sum_{k=1}^{n}\frac{1}{k^2}\end{aligned}$$

Infelizmente, a verificação da convergência das séries envolve cálculos mais complicados pelo simples fato de que seu termo geral seja uma soma.

Para verificar formalmente, no último exemplo, o critério de Cauchy, temos que considerar a diferença

$$|s_m - s_n| = \sum_{k=n+1}^{m} \frac{1}{k^2} \ ; \ n < m \tag{1.5}$$

Se fizermos alguns cálculos experimentais, veremos um comportamento interessante desta soma de quadrados:

$$\frac{1}{(n+1)^2} + \frac{1}{(n+2)^2} = \frac{(n+2)^2 + (n+1)^2}{(n+1)^2(n+2)^2}$$

um quociente de polinômios, no denominador um polinômio de grau 4 e no numerador dois de grau 2. Se somarmos três tais frações resultará novamente um quociente de polinômios no denominador um polinômio de grau 6 enquanto que no do numerador haverá de 3 polinômio de grau 4. Isto nos conduz a escrever $m = n + q$ *e concluir que*

$$|s_n - s_m| = \sum_{k=n+1}^{m} \frac{1}{k^2} = \frac{(q+1)P_q(n)}{Q_q(n)} \ ; \tag{1.6}$$

$$m = n + q \tag{1.7}$$

$$P_q \ \text{do grau } 2(q-1) \tag{1.8}$$

$$Q_q \ \text{do grau } 2q. \tag{1.9}$$

Portanto, $\frac{P_q(n)}{Q_q(n)} \equiv \frac{n}{n^2} = \frac{1}{n}$ *para grandes valores de n e assim é "independente" de m. Estes cálculos mostram, assim, que esta* sucessão *de somas parciais é de Cauchy, portanto convergente e além disto podemos determinar diretamente* N_0 *do Critério de Cauchy:*

$$|s_m - s_n| = \sum_{k=n+1}^{m} \tfrac{1}{k^2} \equiv \tfrac{1}{n} < \epsilon \ ; \ n < m \tag{1.10}$$

$$n \geq \tfrac{1}{\epsilon} \tag{1.11}$$

$$\sum_{k=1}^{1000} \tfrac{1}{k} = 1.64483408185 \tag{1.12}$$

$$\sum_{k=1}^{\infty} \tfrac{1}{k^2} = \tfrac{\pi^2}{6} \approx 1.64493406685 \tag{1.13}$$

$$\text{erro} \quad \leq 0.0001 < 0.001 = \tfrac{1}{1000} \tag{1.14}$$

$$\tag{1.15}$$

definindo ϵ *pela desigualdade acima qualquer número natural* N_o *que satisfaça à condição* $N_o > \frac{1}{\epsilon}$ *preenche a condição do Critério de Cauchy. Um valor aproximado para o limite desta série com erro menor do que* ϵ *é:*

$$s_n = \sum_{k=1}^{N_0} \frac{1}{k^2}; \tag{1.16}$$

Nem sempre é fácil mostrar-se que uma sucessão é de Cauchy, como os cálculos no exemplo acima evidenciaram.

Exemplo: 12 *A sucessão* $s_n = \frac{n}{n+1}$ *é de Cauchy.*

Vamos calcular a diferença

$$|s_{n-1} - s_{m-1}| = |\frac{n-1}{n} - \frac{m-1}{m}| = |\frac{n-m}{nm}| \tag{1.17}$$

com a hipótese de que $n < m$, que é uma hipótese razoável porque não tem sentido $n = m$, logo um dos índices tem que ser o maior. Se apagarmos o número negativo $-m$ do númerador produziremos uma majoração:

$$|s_{n-1} - s_{m-1}| < |\frac{n}{nm}| = \frac{1}{m} = \epsilon \tag{1.18}$$

a última desigualdade se devendo a hipótese: $n < m$. Vemos assim que como $n, m > N$ basta que N seja maior que $Round(\frac{1}{\epsilon})$ = "cota superior inteira de $\frac{1}{\epsilon}$", para se obter o limite com erro inferior a ϵ. Como este limite é bem conhecido, vale 1, podemos avaliar imediatamente a precisão dos nossos cálculos tomando $\epsilon = 0.01$, N tem que ser maior do que 100

$$|s_{101} - 1| = |\frac{101}{102} - 1| = |\frac{101-102}{102}| = \frac{1}{102} < 0.01 \tag{1.19}$$

como esperávamos.

Veja o poder do *Critério de Cauchy* para compensar sua relativa dificuldade. Se você quiser calcular uma aproximação para o valor da série do outro exemplo, $\sum\limits_{k=1}^{\infty} \frac{1}{k^2}$, e o erro estipulado for ϵ, basta escolher $N_o > \frac{1}{\epsilon}$ e com um programa de computador calcular, $\sum\limits_{k=1}^{n_1} \frac{1}{k^2}; n_1 > N_0$, porque

$$(n_1 > N_0) \quad \Rightarrow \quad |\sum_{k=1}^{\infty} \frac{1}{k^2} - \sum_{k=1}^{n_1} \frac{1}{k^2}| < \epsilon \tag{1.20}$$

Exercício: 8 *Séries*

1. *Analise o programa abaixo e identifique qual é função de "contador","contador2", "Cauchy" e de "Cauchy2".*

```
program Cauchy02;
Var contador, contador2        : LongInt;
soma, Cauchy, Cauchy2, delta   : Extended;
Begin
   soma := 0;  Cauchy := 0; contador := 1;
   contador2 := 1000;
   Cauchy2:= 1; delta := 0.000001
   Repeat
      Soma := Soma + 1/(contador*contador);
      contador := contador + 1;
      If ( contador > contador2 ) Then
      Begin
         Cauchy2 := Cauchy; Cauchy := Soma;
         contador2 := contador2 + 1000;
      End;
   Until Abs(Cauchy - Cauchy2) < delta;
   WriteLn('precis = ',delta);
   WriteLn('lim. aprox. da soma p/n = ');
   Write(contador, ' ===> ' , soma);
   ReadLn;
End.
```

2. *Calcule para $\epsilon = 0.000001$ o valor da cota N_o no Critério de Cauchy para a série definida no programa Cauchy02 e compare com o valor obtido pelo próprio programa para o mesmo valor de ϵ.*

   o valor desta série com um erro de 0.000000001 é 1.644934068, *todos os algarismos são significativos neste número. O valor "exato" desta série pode ser obtido usando-se um teorema de* Séries de Fourier.

3. *Calcular, usando o programa Cauchy2, valor aproximado das somas:*

$$a)\ \sum_{n=1}^{\infty} \frac{1}{n(n+1)} \quad b)\ \sum_{n=1}^{\infty} \frac{1}{n^3} \quad c)\ \sum_{n=1}^{\infty} \frac{1}{n^4}$$
$$d)\ \sum_{n=1}^{\infty} \frac{1}{n^{4/3}} \quad e)\ \sum_{n=1}^{\infty} \frac{1}{n^{3/2}} \quad f)\ \sum_{n=1}^{\infty} \frac{1}{n^{5/4}}$$

   Em Pascal não existe potência. Para calcular a^b temos que usar a fórmula $a^b = exp(b * ln(a))$.

> Observe que escrevi Cauchy e não cauchy no programa acima.
> Para Pascal é irrelevante usar minúsculas ou maiúsculas. Isto nos permite redigir um programa de forma mais próxima dos nossos hábitos de redação convencional. Neste sentido veja que os nomes das variáveis não precisam ser curtos, podem e devem ser nomes significativos.

1.5 Coeficiente angular quase-instantâneo

Observação: 11 *Modelos e Matemática*

A Matemática poderia ser definida como uma teoria geral de modelos, *e* modelo *significa um objeto, um meta-algoritmo, que serve para representar algum aspecto da* realidade *ou do* universo *em que vivemos.*

Há duas grandes classes de modelos matemáticos: modelos contínuos *e* modelos caóticos. *Algumas pessoas gostariam de opor* contínuo *a* discreto *mas prefiro não fazê-lo aqui e deixar confinados os modelos discretos numa das duas categorias anteriores. Os modelos contínuos se caracterizam por um uso mais ou menos acentuado da seguinte hipótese:*

Hipótese: 1 *Alta diferenciabilidade dos fenômenos.*
Os fenômenos têm modelos com alta classe de diferenciabilidade.

Uma forma fraca de diferenciabilidade é a *continuidade*...embora a *teoria das distribuições* não a veja assim...

Nos modelos caóticos se admite a hipótese inversa.

A hipótese da diferenciabilidade é perfeitamente aceitável e os modelos caóticos também são excelentes. Não discutirei aqui modelos caóticos. Os dois pontos de vista têm objetivos distintos e naturalmente se excluem... É possível se provar que as funções contínuas se encontram muito próximas de funções altamente diferenciáveis, ver[8], de modo que estas funções servem perfeitamente como modelos para os diversos fenômenos, mesmo os caóticos, em alguns casos.

Com base nesta hipótese, estarei sempre falando das derivadas das funções . Neste livro sempre assumirei que todas as funções são de ordem de diferenciabilidade muito alta, exceto, possivelmente, em poucos pontos nos quais elas não precisam nem ser contínuas, mas tais pontos devem ser considerados exceções. São estes os modelos matemáticos que nos interessam aqui.

Vou discutir derivação aproximada e aproveitar a oportunidade para apresentar as rotinas gráficas do TurboPascal e junto com elas as nossas rotinas gráficas `ModoGrafico, Ponto, Reta, RelogioGr` que se encontram definidas na biblioteca `Tipos`.

A derivada, definida como *taxa de variação instantânea* de uma função, é o limite:

$$\lim_{\Delta x = 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

se o limite existir então posso substituir Δx por uma sucessão e o limite da sucessão assim definida vai nos dar o valor da derivada no ponto x. Como estou *calculando aproximadamente*, usarei um dos valores da sucessão como aproximação do limite. Ao final desta seção, você pode encontrar um programa que traça gráfico de retas tangentes a um determinado ponto do gráfico de uma função. A derivada vai ser calculada *aproximada e graficamente*, no ponto $x = a$.

O coeficiente angular instantâneo é o limite dos coeficientes angulares das secantes quando os pontos de secância se aproximam de um ponto. Isto significa que estou estudando o comportamento do quociente

$$\frac{f(a) - f(x_n)}{a - x_n}$$

quando $\lim_n x_n = a$.

Análise da variação da tangente

Há exatamente duas coisas que podem ocorrer na vizinhança dum ponto em que uma função é derivável:

1. O gráfico corta a tangente.

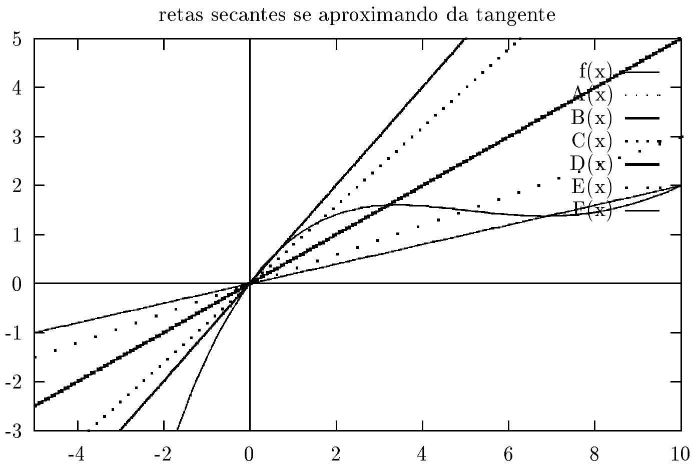

Figura 1.1: Secantes e a tangente-limite, este é o resultado gráfico do programa DERIV01.

2. O gráfico fica de um mesmo lado da tangente, *numa vizinhança do ponto de tangência.*

O instrumento que nos pode ajudar a decidir qual é das duas situações aquela que ocorre é a taxa de variação da tangente: a segunda derivada que também pode ser calculada discretamente por quocientes de diferenças de segunda ordem em cima dos pontos da sucessão:

$$\frac{\frac{f(x_{n+2})-f(x_{n+1})}{x_{n+2}-x_{n+1}} - \frac{f(x_{n+1})-f(x_n)}{x_{n+1}-x_n}}{x_{n+1}-x_n}. \tag{1.21}$$

Há alguns problemas que envolvem o cálculo da derivada segunda aproximadamente.

1. **Quantidade informações**: A primeira delas é que precisamos de tres informações para o cálculo da derivada segunda:

$$x_{n+2}, x_{n+1}, x_n$$

2. **Ordem de grandeza**: Outra é a precisão dos dados, a fração (1.21) é um quociente em que aparece um denominador ao

quadrado; se a partição for uniforme; isto é imediatamente visível porque

$$x_{n+2} - x_{n+1}, x_{n+1} - x_n = \Delta$$

e na medida em que aumenta o grau das expressões, aumentam proporcionalmente os erros. Este problema é geralmente caracterizado como *de ordem de grandeza*.

3. **Fora de "fase"**: Ainda uma outra dificuldade vem do fato de que a sucessão $x_{n+2} - x_{n+1}, x_{n+1} - x_n = \Delta$ é formada de pontos que se encontram *antes* do ponto a, ou *depois* do ponto a, levando-se em conta o sentido em que é feito o levantamento de dados, enfim "fora de fase". Isto dificulta a previsão do valor da segunda derivada no ponto a.

   Este problema é estudado no capítulo 1 do nosso curso avançado de Cálculo Numérico, no capítulo 3 em contexto diferente e com outras técnicas, ver [8].

4. **Oscilação da derivada**: Se você rodar o programa `Comp_D_G.pas` poderá analisar que a oscilação da derivada é maior do que a oscilação da função. O programa compara graficamente f e f'' usando três métodos para calcular a segunda derivada que você deve ignorar neste momento. Se f for uma função com forte oscilação, a derivada terá oscilação ainda mais intensa e o resultado gráfico indica que há pontos em que o erro entre a derivada formal, *cálculo humano* e as derivadas aproximadas *cálculo numérico*, ficam muito grandes.

Veja a tabela de dados (exemplo 19) produzida pelo programa `comp_der.pas` na última seção deste capítulo. Apenas parte dos dados estão apresentados ali. Se você quiser uma maior quantidade de dados, rode o programa. Ignore no momento os métodos indicados para o cálculo aproximado da derivada. Analise apenas as discrepâncias encontradas no cálculo da derivada segunda. O "passo" indicado no programa é o salto entre os nós onde foram calculadas as derivadas. O valor do acréscimo no cálculo da derivada por quocientes de diferenças é Δ com valor estipulado no programa.

O outro programa, `comp_D_G.pas`, executa as mesmas tarefas mas apresenta os gráficos simultâneos da *derivada formal*, da *derivada por quociente de diferenças* e *derivada por convolução*.

Ignore os métodos indicados para calcular derivada neste momento. Os programas gráficos usam as rotinas gráficas `Ponto, Eixos` cuja discussão virá logo a seguir, (veja onde no índice remissivo).

A conclusão a que se chega pela análise destes dados é que a derivada segunda calculada aproximadamente não oferece precisão confiável. Os programas mencionados acima usam uma função fortemente oscilante. Se você substituir esta função por outra, uma função polinomial de grau pequeno, por exemplo, verá que os erros entre as derivadas aproximadas e a derivada formal de segunda ordem caem. O fato, entretanto, é que derivada aproximada de segunda ordem não merece confiança, o que nos força à busca de outros métodos de análise.

Vou fazer outra forma de análise com base nos resultados dos programas `comp*.pas`, que mostram a confiabilidade da derivada primeira calculada aproximadamente. Se a sucessão $(x_n)_n$ oscilar em torno do ponto a, isto é:

$$x_{n+2} = a - \Delta, x_{n+1} = a, x_n = a + \Delta,$$

as duas possibilidades geométricas acima se traduzem em:

<u>Análise da variação da tangente</u>

1. **Gráfico fica dum mesmo lado da tangente**: as diferenças de primeira ordem trocam de sinal.

2. **Gráfico corta a tangente**: as diferenças de primeira ordem não trocam sinal.

A **Hípotese 1** está em uso; sendo uma função derivável, sua derivada pode ser calculada seqüencialmente (com auxílio de qualquer sucessão que convirja para zero), uma vez que o limite não depende da sucessão escolhida. Como a derivada é contínua, seus valores são preservados no limite por qualquer sucessão que tenda a zero, (*podemos encontrar o valor aproximado da derivada usando qualquer sucessão que tenha zero como limite, em outras palavras*). Este é o grande uso da *continuidade* que os livros de Cálculo não costumam deixar claro.

O fato de que o limite não dependente da sucessão se traduz pelo uso de sucessões da forma `a + passo`, cada rodada do *loop* do programa produzindo um novo elemento da sucessão. A análise da troca de sinal na derivada de primeira ordem é feita pelo seguinte trecho de programa:

```
teste := (f(a+2*passo)-f(a+passo))*(f(a+passo) - f(a)) > 0
If teste  {** falso <==> negativo <==>  troca sinal **}
then comando1
Else comando2;
```

Observe o uso da variável lógica, `boolean`, que recebe **true** ou **false**, dependendo de que Δf troque de sinal de um subintervalo a outro.

Observação: 12 *Crítica do método*

Já mencionei antes que um problema básico é o de separação de zeros. Um dos disfarces com que este problema se apresenta se encontra na solução de desigualdades. Uma função pode oscilar de tal modo intensamente que num saltar de subintervalos consigamos informações contraditórias com a realidade...

A única solução que temos para este problema é uma análise minuciosa local com intervenção do usuário, que deverá reduzir o `passo` *manualmente até se certificar de que a análise está correta pela reincidência de resultados que a confirmem.*

Não conheço outros meios computacionais para tratar melhor este problema. Veja no programa `CauchyX.pas` *um método estatístico para fazer tais análises automáticamente.*

Alguns exemplos de uso desta análise são:

1. as soluções gráficas de equações diferenciais, [8, capítulo 3],
2. aproximação polinomial de funções [8, capítulos 1,3],
3. determinação aproximada de raízes, (capítulo 2),
4. cálculo aproximado de integrais, (capítulo 1, volume II).

Vocabulário: 15 CloseGraph, Eixos, Graph, ModoGrafico,Ponto, RestoreCrtMode, Reta, SetLineStyle, SetColor

1. **CloseGraph** *é a rotina do TurboPascal, definida na biblioteca* `Graph` *que retorna ao* modo texto*. Há vários "modos" tanto de texto como de gráficos; estes modos estão relacionados com as diferentes placas gráficas existentes no mercado. Ver com Ctrl-F1 com o curso sobre as palavras* `TextMode, GraphMode`*.*
2. **Eixos** *é a nossa rotina que cria um sistema de eixos retângulares no centro do vídeo. Definida na Tipos. Veja EixosXY(x,y) que traça um sistema de eixos centrado no ponto (x,y), x,y* $\in \mathbf{Q}$*. Definida na Tipos.*
3. **Graph** *é a biblioteca do TurboPascal onde se encontram definidas as rotinas gráficas, isto é a ligação entre o Pascal e o sistema operacional para controlar o vídeo do computador. Ver, por exemplo com Ctrl-F1 em* `PutPixel`*. É uma rotina que "acende"*

uma célula no vídeo, (estas células se chamam em inglês, "pixels".

4. **ModoGrafico** *é uma das nossas rotinas, definida na biblioteca Tipos, usa primitivas do TurboPascalpara iniciar o modo gráfico.*
5. **Ponto** *é uma rotina definida na* `Tipos` *que marca um ponto* (a, b) *na tela em que* a, b *são dois números racionais quaisquer. Se forem muito "*quaisquer*" o ponto pode não aparecer...*
6. **RestoreCrtMode** *é uma rotina do TurboPascal, definida em* `Graph` *que retorna ao "modo texto" imediatamente anterior ao "modo gráfico" corrente (em que o computador se encontra no momento).*
7. **Reta** *É uma de nossas rotinas gráficas definida na biblioteca* `Tipos`*, usa coordenadas numéricas, (quatro números racionais) representando dois pontos do plano, e traça a reta que os liga. Usa a rotina* `Line` *do TurboPascal que liga duas células do vídeo entre si com uma reta, mas as coordenadas de* `Line` *são números inteiros.*
8. **SetLineStyle** *Define um estilo de ponto para traçar curvas. Ver sua sintaxe com Ctrl-F1.*
9. **SetColor** *Permite mudar a cor dos pontos. Veja no manual do TurboPascal (ou com Ctrl-F1) quais são as possibilidades do sistema que você usa.*

Exemplo: 13 *Cálculo de derivadas aproximadamente.*

O próximo programa se encontra no arquivo `pas.zip` *e pode ser rodado usando o compilador do sistema TurboPascal. As rotinas gráficas* `ModoGrafico, Eixos, Ponto, Reta, RelogioGr` *estão definidas na biblioteca* `Tipos` *que também se encontra no arquivo* `pas.zip`*. Todas as outras rotinas gráficas pertencem à biblioteca* `Graph` *do TurboPascal, e você pode obter informações sobre a sintaxe destas rotinas com Ctrl-F1.*

O programa `Deriv01` *traça os gráficos de várias secantes e finalmente o gráfico da tangente no ponto,* aproximadamente, *claro. Produz um resultado semelhante ao da figura (fig.1.1).*

```
Program DERIV01;
Uses Crt, Graph,tipos;

Function f(x:Real):Real;
Begin
  f:= 10*sin(x/10);
End;

Function df1(x:Real):Real;
```

```
Begin
  df1 := (f(x+0.000001) - f(x))/0.000001;
End;
Begin  {** rotina principal **}
  RestoreCrtMode;ModoGrafico;Eixos;
  inicio := -3; fim := 30; a:= 12; delta := 0.01;
  x := inicio;SetLineStyle(2,3,4);
  Repeat              {*** Loop 1 ***}
    Ponto(x,f(x),15); x:= x + delta;
  Until x > fim;
  x := inicio;SetLineStyle(1,3,3);
  Reta(a,0,a,f(a));
  Repeat              {*** Loop 2 ***}
    t := (f(x)-f(a))/(x-a);  y := f(a) + t*(x - a);
    Reta(a,f(a),x,y);
    x:= x + 10*delta; RelogioGr(0,0,t);
  Until Abs(df1(a) - t) < delta;
  x := fim; SetLineStyle(1,3,3);
  Repeat        {*** Loop 3 ***}
    t := (f(x)-f(a))/(x-a);  y := f(a) + t*(x - a);
    Reta(a,f(a),x,y);
    x:= x - 10*delta;RelogioGr(0,0,t);
  Until Abs(df1(a) - t) < delta;
  y1 := df1(a)*(inicio-a)+f(a);
  y2:=df1(a)*(fim-a) + f(a);
  Reta(inicio,y1,fim,y2);
  RelogioGr(0,0,df1(a));
  Readln;
  CloseGraph;
End.
```

Rode o programa e o analise frente ao gráfico obtido. Os *relógios* que aparecem no alto da tela trazem os coeficientes angulares da derivada aproximada e das secantes. Eles são produzidos pela rotina `relogioGr` definida na biblioteca `Tipos`. Nos exercícios abaixo vou convidá-lo a fazer modificações no programa que vão levá-lo a compreender certas deficiências dela.

Exercício: 9 *Gráficos, derivadas e tangentes.*

Os exercícios marcados com um ou mais asteríscos podem ser mais difíceis.

1. **Secantes e tangente**: *Você verificou ao rodar Deriv01 que ele traça várias secantes ao gráfico, à direita e à esquerda de um determinado ponto. Identifique qual dos* `Loop`*s produz as secantes à direita?*

2. **Ponto de tangência**: *Na segunda linha depois do* **Begin**, { *** rotina principal *** }, *Deriv01 atribui valores às variáveis* `inicio, fim, a , delta`. *Qual é o uso de* `a` *?*

3. **Precisão**: *Modifique Deriv01 incluindo uma* **procedure** *chamada* `En-trada` *que permita ao usuário fornecer pelo teclado o* `inicio` *e o* `fim` *do intervalo, o ponto de tangência e a precisão do passo de varredura. Não se esqueça de apagar a segunda linha, depois do* **Begin** *onde estes valores são atribuídos.*

4. **Gráficos de funções**: *O programa* **GraFun.pas** *traça o gráfico de uma função nele definida. Estude-o, troque a função nele definida por outras que você mesmo defina e rode o programa.*

5. ** Altere o programa* **Grafun.pas** *até conseguir um programa que*[5]*:*
   (a) *Faça gráficos simultâneos de* f, f'.
   (b) *Faça gráficos simultâneos de* $f, f' f''$

6. ** Modifique o programa* **Deriv01.pas** *para que seja feito o gráfico conjunto de* f, f' *e seja traçada apenas a reta tangente no ponto* $x = a$ *que o usuário deve fornecer pelo teclado. Use a função* **df1** *definida em* **Deriv01.pas**.

7. **<u>Im</u>precisão da segunda derivada**: *Objetivos deste exercício: aprender a lançar dados em disco, compreender a imprecisão da segunda derivada. O programa* `comp_der.pas` *calcula derivadas primeira e segunda de uma função definida no próprio programa,* formalmente*, por* quociente de diferenças*, e por* convolução*, (um método que é estudado em [8, capítulo 3]. Ignore por enquanto a convolução.*
   (a) *Estude o método usado neste programa para lançar os dados em disco.*
   (b) *Modifique o programa, em particular a rotina "Escreve" para que os dados apareçam na tela.*
   (c) *Rode-o para analisar os valores obtidos.*
   (d) *Modifique* $\Delta x = 0.000000001$ *definido no programa por valores maiores e menores e compare os resultados.*
   (e) *Use valores diferentes para* Δx *no cálculo da derivada primeira e da derivada segunda e compare os resultados.*

8. **Análise dos gráficos.*

[5] consulte os programas Deriv*, Comp*.

(a) *Substitua a função f, no programa Deriv01, por f(x) = sin(x) e analise o efeito do resultado. Isto lhe mostra que gráficos não são absolutos na tela: é preciso criar condições para que eles apareçam... Solução para este problema de escala: altere os valores de f dentro de* **Ponto** *com duas mudanças de variável: Ponto(x,zoom*(f(x/acelera),cor);*

    i. *Se zoom > 1 o gráfico será dilatado.*

    ii. *Se acelera > 1 o gráfico será acelerado.*

(b) ** Permita que o usuário forneça "zoom" e "acelera" pelo teclado.*

(c) **** usando* **ReadKey** *e analisando se o usuário aperta uma das seta (p/cima ou p/baixo) faça o programa dar um zoom expandindo ou contraindo o gráfico. Use processo análogo, para acelerar/decelerar com as teclas p/direita ou p/esquerda. Por exemplo: p/esquerda acelera, p/direita atrasa... As setas são definidas por Chr(24),..Chr(27) e seu programa deverá guardar numa variável: teste:=* **ReadKey** *e com um* **If** *controlar se o coeficiente zoom, por exemplo, deve ser multiplicado ou dividido por um número inteiro.*

9. *Modifique os parâmetros de SetLineStyle e analise os resultados. Com Ctrl-F1 você pode ver todos os parâmetros existentes.*

10. *Observe que dentro dos loops 2 e 3, do programa* `Deriv01`*, são desenhadas as retas secantes. Veja a modificação feita no incremento* $(10 * delta)$. *Modifique o coeficiente 10, (aumente-o ou o diminua), para ver o resultado.*

11. *Modifique a escala do Δx no cálculo da derivada e analise os resultados.*

12. *Estude a rotina* `Ponto`*, definida em* `Tipos` *para compreender como foi criada a* Geometria Analítica *dos programas. Observe:*

    (a) *estou usando as variáveis* `xeixo` *e* `yeixo` *onde se encontram armazena-das as quantidades de células do vídeo na horizontal e na vertical.*

    (b) *estas quantidades foram divididas em pacotes de 10 unidades para representar os intervalos entre dois inteiros. Assim,* *posso atingir a precisão máxima de uma casa decimal com as coordenadas dos pontos na geometria que defini.*

(c) *há uma diferença entre as coordenadas na horizontal e na vertical, estas últimas estão multiplicadas pela constante* `razao` *que que memoriza a distorção do vídeo. Se você retirar* `razao`*, os círculos vão virar elipses. Veja a definição de* `razao` *no final da biblioteca* `Tipos`*, ela usa função* `Detect` *do TurboPascal.*

13. **Modifique a minha* Geometria Analítica *alterando a quantidade de pontos nos intervalos inteiros.*

14. *Verifique que há duas rotinas chamadas* `Eixos, EixosXY` *em* `Tipos`*. Uma delas produz um sistema de eixos sempre no mesmo lugar; a outra coloca o sistema de eixos onde o usuário desejar (e se ele não tiver cuidado, nenhum sistema de eixos vai aparecer...).*

15. *Faça um programa que peça as coordenadas de dois pontos no "plano"* $[-34, 34]\ X\ [-18, 18]$ *e desenhe estas retas.*

16. *Complete o programa anterior para desenhar uma reta a partir de sua equação cartesiana. Melhore o seu programa para desenhar parábolas, círculos e elipses. Faça um menu para o seu programa para que o usuário do mesmo escolha o que fazer, (veja o "programa" GaPlana no arquivo* `pas.zip`*, ele representa um bom começo).*

17. *Complete o seu programa para desenhar retas tangentes em pontos escolhidos pelo usuário.*

18. *Faça um programa que escolha quatro pontos num círculo e por estes pontos faça passar quatro tangentes determinando um quadri-látero no qual o círculo fica inscrito. Podem ser quatro pontos quaisquer? Não podendo, o seu programa deve analisar os pontos e emitir uma mensagem indicando a impossibilidade.*

19. *Modifique a saída de dados dos relógios que indicam os coeficientes angulares para que apareçam números mais curtos (com menos casas decimais). Acrescente uma mensagem para que o usuário saiba o que significam.*

20. *Modifique a procedure RelogioGr para que ela aceite mais dois parâme-tros: uma mensagem e um local de posicionamento na tela,(possivelmen-te mais outro parâmetro..., de modo que os coeficientes angulares das secantes apareçam do lado direito ou do lado esquerdo).*

Não vou resolver os problemas gráficos que ficaram pendentes na discussão acima, dentro dos exercícios. Se você tiver entendido o que aconteceu, deverá também estar a caminho de encontrar a solução. De qualquer maneira, esta solução passa pelos seguintes passos:

1. determinação do máximo e do mínimo da função no intervalo dado.
2. Com auxílio do máximo e do mínimo criar constantes de escala que distribuam o gráfico na tela inteira, para evitar o que aconteceu com a função $f(x) = x^2$, que ficou pequena e achatada no centro da tela.
3. A procedure `eixos` deve ser modificada, ser substituida por `eixosXY` para otimizar o gráfico na tela: se o gráfico for todo positivo, o *eixo dos* x talvez nem precise aparecer...

Estes *problemas* são sérios e tomarão algum tempo para corrigi-los, mas neste livro não há lugar para ir muito a fundo nesta questão sem desvirtuar o objetivo central do texto, que não é sobre gráficos em computação, que muita gente confundo com *computação gráfica...* apenas quero despertar a sua atenção para a existência deles.

1.6 Cálculo de integrais

Nos cursos de Cálculo, se mostra que

$$\int_a^b f(x)dx \approx \sum_{k=0}^{n-1} f(a + k\Delta x)\Delta x,$$

em que Δx é o comprimento do subintervalo $[x_k, x_{k+1}]$. A soma que se encontra à direita da integral se chama "soma de Riemann"[6]

[6] Temos aqui um tipo particular de soma de Riemann, uma soma de Riemann uniforme.

e representa uma forma de calcular-se aproximadamente a integral porque é a soma de áreas dos retângulos com uma base pequena e tendo como altura uma aproximação do valor de f. Neste capítulo vou me limitar a discutir somas de Riemann como método de cálculo de integrais, mas vou adverti-lo que este não é o melhor método para calcular integrais.

1.6.1 Somas de Riemann

Vou calcular aproximadamente a integral:

$$\int_0^{10} e^{-x^2} dx$$

usando *Somas de Riemann*. As somas de Riemann, que recebem este nome em homenagem ao matemático alemão Bernard Riemann (1826-1866), são somas das áreas dos retângulos que se encontram logo acima ou abaixo do gráfico de uma função, considerando-se o intervalo em que ela está definida subdividido em pequenos subintervalos que formam a base dos retângulos. Consulte a respeito um livro de Cálculo.

Preciso de alguns novos conceitos e novas variáveis, no próximo programa:

Vocabulário: 16 nó, partição, precisão, soma, deltax, inicio, fim.

1. `nó` *O intervalo $[a, b]$ sobre o qual está definida uma função deve ser dividido em subintervalos. Os pontos que fazem estas subdivisões se chamam "nós", e seu conjunto define uma partição do intervalo $[a, b]$.*
2. `partição` *é conjunto dos subintervalos em que se subdividiu o intervalo $[a, b]$ que é o domínio de uma função. Cada subintervalo está definido por dois nós consecutivos. O conjunto dos nós define também a partição.*
3. `precisao` *é uma variável que indica o número de divisões do intervalo de integração. No próximo programa é uma constante. Esta é uma prática que não adotarei nos meus programas: uso de constantes. Faço isto apenas neste caso para que você tenha um exemplo.*

4. `soma` *é uma variável onde vou acumular os sucessivos valores das áreas dos retângulos da soma de Riemann. Observe que, logo no início do programa,* `soma` *é inicializada com zero.*

5. `deltax` *representa a norma da partição uniforme do intervalo em que estou calculando aproximadamente a integral. Vou usar apenas partições uniformes e as somas de Riemann a elas associadas.*

6. `inicio, fim` *são duas variáveis que guardam inicialmente os extremos do intervalo de integração .*

Observe o uso distinto de variável para definição da fun-ção e da variável que varre o intervalo. Veja inclusive que escrevo no programa f(inicio) ...e que inicio vai sendo modificada para *receber* o nó seguinte da partição, dentro do loop. Logo depois do loop, soma é atualizada para se transformar em $soma*deltax$. Dentro do WriteLn, no loop, apenas se manda imprimir o produto $soma*deltax$ mas soma não está sendo multiplicada por `deltax` dentro do loop. Isto só é feito depois do `loop`. O efeito disto é que o programa vai imprimindo a sucessão dos valores da integral ao longo da partição. Também o programa economiza tempo de processamento porque se faz uma multiplicação, ao final.

O programa abaixo se encontra no arquivo `pas.zip`. Ele cálcula uma *soma de Riemann.*

Exemplo: 14 *Soma de Riemann.*

```
Program Riem01;
Uses Tipos,Crt;
Const  precisao = 100; {numero de divisoes do intervalo}
Var  contador               : Integer;
   deltax, soma, inicio, fim : Real;
Function f(x : Real) : Real;
Begin
     f := exp(-x*x);
End;
Begin
     ClrScr; soma := 0; inicio := 0; fim := 10;
     deltax := (fim - inicio)/precisao;
     While inicio < fim Do
     Begin
          soma := soma + f(inicio);
          inicio := inicio + deltax;
          WriteLn('S = ',soma*deltax:9:4);
     End;
     soma := soma*deltax;
     WriteLn('O valor da integral aproximada e: ',soma:9:4);
```

```
    Apito; Apeteco; ClrScr;
End.
```

Exercício: 10 *Criando entrada/saída de dados para integração . Algumas sugestões se encontram no próximo exemplo.*

1. *Verifique que no programa (exemplo,14), página 60, existe uma variável que foi declarada mas se encontra sem uso. Apague-a, mas observe que o programa roda com ela. Isto é um defeito do Pascal que não analisa a* otimização *do programa. Está sendo ocupada memória inutilmente...*
2. *Modifique o programa para ler* `precisao` *pelo teclado. Observe que* `precisao` *não mais poderá ser uma constante. Não esqueça a mensagem dizendo que* tipo de número se espera*. No (exemplo,14), página 60,* `precisao` *é número de divisões do intervalo: uma constante inteira.*
3. *Calcule a integral com 1000 divisões de precisão .*
4. *Melhore a mensagem final indicando qual foi a precisão usada.*
5. *Modifique o (exemplo,14) envolvendo o loop* **While-Do** *com um* **Rep-eat-Until** *para calcular sucessivamente a integral com novos valores para* `precisao` até que *o usuário diga que não mais deseja continuar:*

   $$ReadLn(Resposta);$$

   Não se esqueça de inicializar o valor de soma antes de executar o loop interno. Se você já tiver construído sua função `Sim_Nao` *pedida em outro exercício, use-a.*
6. *Modifique o programa para que* `deltax` *seja lida pelo teclado em vez de ser calculada dentro do programa.*
7. *Inclua no* `Loop` *que calcula a soma de Riemann a rotina* `Ponto` *para que seja feito o gráfico da função ao mesmo tempo que a integral aproximada está sendo calculada, (antes de entrar no* `Loop` *inicie o modo gráfico... e não se esqueça de o fechar depois). Se o programa não funcionar, pode ser que você tenha esquecido um* **ReadLn** *para congelar o gráfico na tela..., entre outras coisas...*

8. *Inclua no seu programa gráfico algumas retas verticais para que se possa ver onde estão os retângulos da* soma de Riemann.

9. *Verifique que a integral* $\int_{-\infty}^{\infty} e^{-x^2}dx$ *se pode deduzir de* $\int_{0}^{\infty} e^{-x^2}dx$ *e modifique o (exemplo,14) para calcular* $\int_{-\infty}^{\infty} e^{-x^2}dx$, *solicitando os dados necessários ao usuário e com uma saída de dados que o informe com que precisão a integral foi calculada.*

10. ** teste de Cauchy. Envolva o cálculo de* $\int_{-\infty}^{\infty} e^{-x^2}dx$ *num loop, memorizando em* `soma_anterior` *o valor da integral calculada a cada passagem, de tal modo que a condição para encerrar o laço seja:*
    `Abs( soma_anterior - soma ) < erro` *" e "*`erro`*" escolhido pelo usuário.*

1.6.2 Teste de convergência aplicado à Soma de Riemann

Você pode encontrar aqui sugestões para alguns dos exercícios anteriores. O programa apresentado abaixo repete uma soma de Riemann reduzindo "deltax" até que a diferença entre dois valores sucessivos da soma seja menor que a "precisao" estipulada.

Vocabulário: 17 Write, Read

1. **Write** *é semelhante ao WriteLn, com a exceção de não passar a uma linha nova depois de escrever a mensagem. Fica bonito usá-lo em conexão com* **ReadLn**, *pois o cursor fica piscando logo depois da mensagem.*

2. **Read***: existe este comando no Pascal, mas, por enquanto, você deve evitá-lo, ele tem algumas complicações extras: exige que seja feita a limpeza do* `buffer` *de de entrada de dados.*

Observe o uso de `somaanterior` e `soma` no início do loop **Repeat/Until** essencial para renovar o valor de `somaanterior` que será testado dentro da condição do **Until**.

Exemplo: 15 *Somas de Riemann com teste de Cauchy.*

```
{***  programa para calcular integrais com somas de Riemann
 ***  com um teste de Cauchy: abs( somaanterior-soma) < precisao
 ***  autores Geronimo/Praciano-Pereira data 26/01/92    **}
program Riem02;

Uses Tipos, Crt;

Var contador  : Integer;
    precisao, delta,x, somaanterior, soma, inicio, fim, deltax : Real;

Function f(x:Real):Real;
Begin
   f := sin(x*x);
End;

Begin
  ClrScr; soma := 0; contador := 1;
  Writeln('Inicio e fim do intervalo de integracao');
  Write('Inicio: ');Readln(inicio);
  Write('Fim: ');Readln(fim); deltax := (fim-inicio)/1000;
  Write('precisao: 0.00001  < precisao  < 0.1  ');Readln(precisao);
  x := inicio; ClrScr;
  Repeat
    somaanterior := soma; soma := 0;
    x := inicio;
    While x < fim Do
      Begin
        soma := soma + f(x);
        x := x + deltax;
        Writeln('S=  ', soma*deltax:9:4);
      End;
    contador := contador + 1;
    soma := soma*deltax;
    WriteLn('Experiencia no. ',contador);
    WriteLn('Valor da Integral: ',soma:9:4);
    deltax:= deltax/10; Apito;Apeteco;
  Until (abs(somaanterior-soma) < precisao) ;
  ClrScr;
  WriteLn('Valor da integral = ',soma:10:10);
  WriteLn(' precisao = ',precisao:10:10);
  WriteLn('numero de experiencias = ',contador);
End.
```

Exercício: 11 *1. Rode o programa (exemplo, 15), página 62, e verifique que a contagem do número de experiências está errada. Identifique no programa onde se encontra o erro e corrija-o, (é apenas uma "linha" mal colocada...),* erro comum*!*

2. Substitua no programa (exemplo, 15)

Write('Inicio'); por WriteLn('Inicio');

Rode o programa e analise a diferença.

3. *Substitua no programa (exemplo, 15)*

   Write('Inicio'); por GotoXY(10,10);Write('Inicio');

   Rode o programa e analise o efeito de GotoXY(10,01);

4. *Usando GotoXY, modifique o lay-out da entrada de dados do programa (exemplo, 15).*

5. *Usando GotoXY modifique o lay-out da saída de dados do programa (exemplo, 15).*

6. *Usando GotoXY modifique o lay-out da entrada e da saída de dados do programa (exemplo, 15) de modo que*

   a entrada de dados ocupe a metade superior da tela.

   *no centro apareça mensagem "*Calculando ...*" durante o processamento.*

   a saida de dados ocupe a metade inferior da tela.

   Não se esqueça de apagar tudo antes de começar de novo, **ClrScr***...*

7. *Inclua uma mensagem final que indique o valor da integral, o número de vezes que o cálculo foi repetido, o intervalo e a equação da função.*

8. *Acelere o programa (exemplo, 15), eliminando a mensagem que mostra a evolução da integral dentro do laço interno.*

   Observação: 13 *Teste interno de funcionamento A mensagem referida é um método de verificação do funcionamento de um programa em faze de elaboração e que deve ser eliminado quando o programa já se encontra suficientemente testado.*

9. *Modifique o programa (exemplo, 15) para calcular a área entre os gráficos de duas parábolas.*

10. *Calcule a área limitada pela reta* $y = x$ *e a parábola* $y = x^2 + 3x - 7$. *Deixe que o programa encontre os pontos de interseção...*

11. *Defina* sen_+ *a parte positiva do sen e calcule a sua área de* -3π *até* 3π.

12. *Calcule*

(a) $\frac{1}{2\pi}\int_0^{2\pi} sen(x)dx$	(b) $\frac{1}{2\pi}\int_0^{2\pi} cos(x)dx$	(c) $\frac{1}{2\pi}\int_0^{2\pi} sen(x)cos(x)dx$
(d) $\frac{1}{2\pi}\int_0^{2\pi} sen(2x)dx$	(e) $\frac{1}{2\pi}\int_0^{2\pi} sen(3x)dx$	(f) $\frac{1}{2\pi}\int_0^{2\pi} cos(2x)dx$
(g) $\frac{1}{2\pi}\int_0^{2\pi} sen(3x)cos(5x)dx$	(h) $\frac{1}{2\pi}\int_0^{2\pi} sen^2(3x)dx$	(i) $\frac{1}{2\pi}\int_0^{2\pi} cos^2(3x)dx$
(j) $\frac{1}{2\pi}\int_0^{2\pi} sen^2(5x)dx$	(k) $\frac{1}{2\pi}\int_0^{2\pi} cos^2(5x)dx$	

13. *Considere* $f(x) = -1 \leftarrow x \in [-3,0)$ *e* $f(x) = 1 \leftarrow x \in [0,3]$. *Calcule*
$$\frac{1}{2\pi}\int_0^{2\pi} f(x)sen(nx)dx$$
com $n \in \{0,1,2,3,4,5,6\}$.

14. *Considere* $f(x) = -1 \leftarrow x \in [-3,0)$ *e* $f(x) = 1 \leftarrow x \in [0,3]$. *Calcule*
$$\frac{1}{2\pi}\int_0^{2\pi} f(x)cos(nx)dx$$
com $n \in \{0,1,2,3,4,5,6\}$.

15. *Considere* $f(x) = x; x \in [-\pi,\pi)$. *Calcule*
$$\frac{1}{2\pi}\int_0^{2\pi} f(x)cos(nx)dx$$
com $n \in \{0,1,2,3,4,5,6\}$.

16. *Considere* $f(x) = -x; x \in [-\pi,\pi)$. *Calcule*
$$\frac{1}{2\pi}\int_0^{2\pi} f(x)cos(nx)dx$$
com $n \in \{0,1,2,3,4,5,6\}$.

1.6.3 Testes de Cauchy com Somas de Riemann

O programa (exemplo,15), página 62, só para quando a diferença entre duas somas de Riemann sucessivas for menor que a precisão “precisao” escolhida. O teste de Cauchy é o único método de garantir precisão . É interessante incluir uma condição para que o programa pare quando contador atingir um determinado valor, uma tal condição impede que o loop se prolongue por tempo demasiado.

É um *pseudoteste de Cauchy* porque analisa a diferença entre os termos sucessivos de uma sucessão e não entre dois termos quaisquer. Não prova a integrabilidade de nenhuma função; só podemos usar este método para cálculo de integrais com boa garantia de precisão se soubermos, por outra fonte, que a integral existe. Como estou aplicando o método em funções contínuas, sabemos que o resultado da integral tem a precisão esperada. Se pudermos provar que uma função é integrável, então o programa (exemplo,15) calcula sua integral com aproximação arbitrária e possivelmente muito tempo de processamento...

O programa usa, portanto, o fato de que, se uma função for integrável, o refinamento sucessivo de partições uniformes converge para o valor da integral. Observe que não estou dizendo que é possível deduzir com cálculos numéricos se uma função é integrável. Bem diferente, *estou dizendo que, se uma função for integrável podemos usar partições uniformes para calcular-lhe uma aproximação da integral definida, e portanto um programa de cálculo numérico pode obter boas aproximações para uma tal integral definida.*

No final do capítulo, voltarei a discutir um programa para cálculo de integrais que especializa um pouco mais o que discuti aqui.

1.7 Máximos e mínimos

Neste parágrafo, vamos trabalhar com máximos e mínimos. É relativamente fácil para os humanos calcular extremos; para os computadores, *isto é uma tarefa em estudos... Tudo o que podem fazer é calcular aproximadamente, e ainda assim, sob monitoramento humano.*

Veja no exemplo seguinte qual é a técnica para calcular extremos. Vamos discutir o cálculo de máximos; o de mínimos se faz simetricamente.

Primeiro se inicializa a variável `maximo` com um *valor qualquer assumido pela função* . Depois se a faz crescer à medida que o intervalo da função seja varrido, comparando-se o seu valor com o da função.. Podemos usar também o valor da função no início do intervalo porque ele é menor ou igual ao máximo que desejamos calcular. O programa abaixo faz exatamente isto.

Exemplo: 16 *Cálculo aproximado de máximo de uma função.*

```
{***  programa para calcular maximo de uma funcao       ***
 ***  autores Geronimo/Praciano-Pereira data 26/01/92    **
 ***  programa testado - sem erros encontrados           **}
```

```
program minMax01;
Uses Tipos, Crt;

Function f(x:Real):Real;
Begin
   f := sin(x*x);
End;

Begin
  ClrScr;
  Writeln('Inicio e fim do intervalo de pesquisa');
  Write('Inicio: ');Readln(inicio);
  Write('Fim: ');Readln(fim);
  Write('passo: ');Readln(deltax);
  ClrScr; max := f(inicio);
  While inicio < fim Do
      Begin
        y := f(inicio);
        If y > max Then max := y;
        inicio := inicio + deltax;
      End;
  WriteLn('Maximo = ',max:8:4);
  ReadLn;
  ClrScr;
End.
```

Exercício: 12 *Cálculo de Máximos. Todos os exercícios usam o (exemplo, 16), página 66, como programa básico que deve ser modificado.*

1. *Mude a equação da função e volte a calcular o Máximo no mesmo intervalo.*
2. *Otimize o programa evitando a variável y.*
3. *No (exemplo, 16) defina passo como variável e leia o seu valor pelo teclado com um ReadLn.*
4. *Envolva o While/Do com outro loop, refine o passo sucessivamente para obter mais precisão no cálculo do máximo.*
5. *Inclua uma variável no programa para memorizar o valor de x em que ocorre o máximo, (observe que ela tem que ser atualizada sempre que "maximo" o for). Inclua este ponto na saída de dados. Possivelmente você precisará de Begin/End; porque o* **If** *terá um comando composto para executar ...*
6. *Porque o programa modificado pelo exercício 5 não consegue registrar todos os pontos de máximo? Modifique o programa*

para registrar todos os máximos (ou mínimos relativos) relativos. *Você pode resolver* sofrivelmente *esta questão usando uma* variável de tipo vetorial, **array**. *O problema é que variáveis do tipo vetorial têm o número de coordenadas definido. Se você estipular este número como n, mas se houver $n+1$ pontos de máximo, um será perdido.*

7. *Modifique o programa do item anterior para que o máximo seja escrito na tela assim como o ponto em que ê lê ocorre, em cada ocorrência.* *Uma solução* visual *para o problema de registro dos máximos...*

8. *Modifique o programa incluindo rotinas gráficas de modo que o gráfico seja feito paralelamente ao cálculo do* Max *ou do* min. *Inclua no seu programa um segmento de reta vertical no ponto de* máximo *e possivelmente um apito para alertar o usuário.*

9. *Modifique o programa para cálculo de máximo da função* $f(x,y) = x^2 + y^2$.

10. *Substitua em todas as questões acima* Max *por* min, *se você ainda não tiver feito isto.*

Se você tiver rodado o programa minMax01 com a função $f(x) = sen(x)$, ele não terá feito o registro de todos os pontos de máximo de f. Só irá encontrar um deles: o maior deles e ainda assim o último se este ponto de máximo ocorrer várias vezes como na função *sen*. O programa não está fazendo o registro de onde o máximo está ocorrendo.

Os exercícios acima não ofereceram alternativas suficientes para encontrar todos os pontos de máximo de uma função. Uma maneira de fazê-lo consiste em usar o tipo de dado **array** = matriz. Veja na próxima seção como se definem "arrays". Voltarei a discutir como registrar os pontos de máximo. Uma outra maneira de memorizar dados seqüencialmente, que é o caso das matrizes é **set** = conjunto. Veja com Ctrl-F1 com o cursor em cima de "set" a sintaxe deste tipo de dado em Pascal. Os conjuntos em Pascal são muito limitados. Veja o programa *conjunto.pas* para uma alternativa mais ampla em que se usam *cadeias de caracteres* para criar um conjunto. Rode o programa, veja o resultado e depois o analise com cuidado.

Exercício: 13 *Determinando os pontos de máximo.*

1. *Acrescente um novo* loop *ao programa* `MinMax01`*, (exemplo, 16), para que ele volte a varrer o intervalo e quando* $|f(x) - max| < erro$ *então o valor de* x *seja escrito. Tente* If $f(x) = max$ Then Writeln(x); *e analise por que não irá funcionar. As igualdades são problemáticas...*

2. *Modifique o programa para fazer um refinamento da pesquisa.*

3. *Defina um conjunto* S *num programa, de modo que você forneça os seus elementos pelo teclado com um* **Read** *dentro de um* Loop*. Ao final o programa deverá escrever o conjunto*[7]*.*

4. *Modifique o programa acima para que o conjunto* S *seja formado de pares ordenados, vetores com duas coordenadas. Faça o seu programa imprimir os vetores usando parenteses e vírgulas.*

5. *Modifique* `MinMax01` *para registrar todos os máximos e mínimos de uma função, indicando-os no gráfico e finalmente escrevendo na tela com mensagens adequadas os dois conjuntos de máximos e mínimos, (devem ser pares ordenados* $(x, f(x))$ *).*

6. *Rode o programa* conjunto.pas *com* delta:= 0.1 *e analise o resultado, verifique que apenas parte dos dados foram impressos. Com* `Ctr-F1`*, veja o tamanho que podem ter as* `string`*s e procure compreender o que ocorreu.*

Exemplo: 17 *Máximo de funções de duas variáveis.*

A versão do programa `MinMax02` *que se encontra no disco é ligeiramente diferente da que você tem abaixo.*

```
{** programa para calculo de maximo de funcoes de duas
   variaveis autor Tarcisio Praciano-Pereira  01/03/92 **}


program MinMax02;

Function f(x,y:Real):Real;
Begin
     f:= x*x - y*y;
End;
Begin
  ClrScr;
  WriteLn('Calculo de Maximos em duas variaveis.');
  Writeln('Varre regiao retang. em busca do maximo.');
  Writeln('Intervalos que definem esta regiao:');
```

[7] Ver o programa *conjunto.pas*

```
  Write('Eixo OX: Inicio do intervalo: ');ReadLn(inicioX);
  Write('Fim do intervalo: '); ReadLn(FimX);
  Write('Eixo OY: Inicio do intervalo: ');ReadLn(InicioY);
  Write('Fim do intervalo: '); ReadLn(FimY);
  WriteLn('Passo de varredura: '); ReadLn(passo);
  x:= inicioX; max := f(inicioX,inicioY);
  GotoXY(1,12);WriteLn('PENSANDO...');
  While inicioY < FimY Do
  Begin
       x := inicioX;
       While x < FimX Do
        Begin
          z := f(x,inicioY);
          If z > max Then max := z;
          x := x + passo;
        End;
        inicioY := inicioY + passo;
  End;
  WriteLn('MAXIMO = ',max:8:5);
  ReadLn;
  ClrScr;
End.
```

Observe que a única mudança para calcular um máximo de função de duas variáveis foi a inclusão de novas variáveis para controlar o fluxo de um segundo loop, de maneira que o programa pudesse varrer um retângulo. A técnica se aplica imediatamente para cálculo de integrais duplas, triplas etc...

`MinMax02`, (exemplo, 17), calcula apenas o valor do máximo e não os pontos de máximo. Para encontrar os pontos de máximo se pode incluir um loop para fazer outra varredura para encontrar onde a desigualdade $Abs(f(x,y) - Max) < erro$ é satisfeita e neste caso escrever o par (x,y). Use alternativamente uma variável do tipo `string` e nela inclua os ternos $(x,y,f(x,y))$ que sejam extremos relativos[8].

Exercício: 14 *Integrais múltiplas. Os exercícios deste bloco usam o (exemplo, 17).*

1. *Transforme o programa (exemplo, 17) para calcular a integral de $f(x,y) = x * y$. Observe que $\int_{-a}^{a}\int_{-b}^{b} f(x,y)dxdy = 0$ e assim teste se seu programa está correto com diferentes valores para a, b.*

2. *Escolha algumas integrais duplas do seu livro de Cálculo, e calcule seus valores com o programa modificado.*

[8] veja o programa *conjuntos.pas*

3. *Altere o programa (exemplo, 17) para calcular integrais triplas.*
4. *Escolha algumas integrais triplas do seu livro de Cálculo, e calcule seus valores com o programa modificado.*

1.7.1 Análise estocástica de integrais

Vou apresentar um método de análise estocástica para integrais e estudar os programas `R_AleatX` que produzem esta análise. É preciso insistir num ponto: os testes estocásticos assim conduzidos produzem apenas a hipótese de que as integrais existem (ou não existem). A demonstração deve ser feita por outros métodos. Interessa apresentar a metodologia em si mesma. Os programas mencionados se encontram no arquivo `pas.zip`, ver índice remissivo. São dois programas: você deve primeiro estudar `R_Aleat1` porque o segundo, `R_Aleat2`, é semelhante mas grava os dados em um arquivo em disco.

Como vou fazer vários testes e ao final deles calcular o *desvio padrão* e o *afastamento da média* , preciso de mais uma estrutura de dados multivariados do Pascal:

Vocabulário: 18 Array , record, Random, Randomize, Str
Há dois tipos de dados "multivariados" em Pascal: registros, matrizes

1. **Array** = *matriz. Matrizes são variáveis com várias coordenadas* todas do mesmo tipo*:* `x : Array[1..n] of Extended;` *cria uma variável que tem as coordenadas*

$$x[1] \in \mathbf{R}, \cdots, x[n] \in \mathbf{R};$$

observe que o método de acesso às coordenadas em Pascal é: $x[i]$ *e não como em Matemática:* x_i *ou seja:*

$$matemática \rightarrow x_i \equiv x[i] \leftarrow Pascal$$

As matrizes podem ser multivetoriais:

`Mat : Array[1..n,1..m] of Extended;`

define uma matriz cujo acesso em Pascal se faz com $Mat[i][j]$ *que corresponde em Matemática a* Mat_{ij} :

$$matemática \rightarrow Mat_{ij} \equiv Mat[i][j] \leftarrow Pascal$$

O nível de "profundidade" de uma matriz depende apenas da memória disponível, mas é bom observar que as matrizes ocupam muita memória e devem ser usadas com economia.

2. **record** = *registro. Um registro é um tipo de dado composto de várias coordenadas, cada uma delas podendo ser de tipo diferente. A sintaxe para definição de um* record *é:*

```
NomeReg :  record
           NomeCampo1 : tipo
           ...
           NomeCampoN : tipo
end;
```

por exemplo, se quisermos fazer a lista de cadastro de clientes de uma firma precisaremos de alguma coisa como:

```
Endereco  : record
            Nome     : string[40];
            Rua      : string[60];
            Bairro   : string[20];
            CEP      : string[9];
            cidade   : string[20];
            saldo    : Real;
end;
```

3. **Random** *Função do Pascal para produzir um número aleatoriamente. Ver sintaxe com Ctrl-F1.*

4. **Randomize** *Rotina do Pascal para ativar o gerador de números aleatórios.*

5. **Str** *Comando do TurboPascal para transformar número em cadeias de caracteres.* **Str***(numero,mensagem) transforma "numero" em uma variável do* tipo cadeia de caracteres *e o guarda na variável "mensagem". É útil para criar uma saída de dados mais bonita. Seu uso se encontra exemplificado no próximo trecho de programa. Se você quiser construir uma procedure genérica para saída de dados, certamente precisará deste comando.*

Neste livro não haverá nenhuma ocasião em que precisemos de usar registros, mas as matrizes serão usadas.

O objetivo do próximo programa é *estatística automatizada*:

1. calcular várias somas de Riemann, não uniformes, de uma função, feitas no loop interno do planejamento abaixo, ver[9] $\{2**\}$;
2. comparar estas diversas somas de Riemann entre si para analisar a diferença entre elas, feito na estatística final, ver $\{7**\}$;
3. calcular o valor médio das somas de Riemann, ver $\{7**\}$;
4. calcular o afastamento de cada uma da média,ver $\{7**\}$;
5. calcular o desvio padrão ,ver $\{7**\}$.

Desvio padrão é uma medida de dispersão, isto é, se tivermos uma *amostra* muito dispersa isto significa uma das duas seguintes situações:

dados heterogêneos: os dados que temos são heterogêneos, não apresentando nenhuma *tendência* ou comportamento assintótico; ou

amostragem mal feita: será preciso refazer a amostragem para tentar descobrir alguma tendência dos dados.

Consulte um livro de Estatística para ler mais sobre o assunto.

De acordo com estas diretrizes, preciso que o programa trabalhe com valor para *deltax* variável em cada soma de Riemann.

O planejamento do programa `R_Aleat1.pas` é:

Exemplo: 18 *Trecho do programa Riemann Aleatoria.*

```
Begin
 n := 1;
 Repeat
   {1**** inicializacao dos dados}
   deltax := Random(Random(1000))/denominador + DeltaMin;
   soma:=0; x:=inicio;
   Norma :=0;
   If n = 1 then Begin min := dif; Max := 0;  End;
   {2*** Calculo de uma Soma de Riemann}
   Repeat
      soma := soma + f(x)*deltax;Randomize;
      If Norma < deltax Then Norma := deltax;
      x := x+deltax; contador:= contador +1;
```

[9] As referências são relativas ao (exemplo, 18)

```
      deltax := Random(Random(1000))/denominador + DeltaMin;
   Until x > fim;
   {3 **** registro dos dados para avaliacao estatistica}
   dif := Abs(Soma-SomaAnterior);  Media := Media + Soma;
   afastamento[n] := Soma; {3.1 ** atribuicao no array **}
   If min > dif Then min := dif;
   If Max < dif Then Max := dif;
   {4***  Saida parcial  de dados ***}
   Str(n,Msg1);Str(soma:8:8,Msg2); Str(dif:8:8,Msg3);
   Msg1 := 'n= '+ Msg1 + espaco +  'Int= '+ Msg2 + espaco + 'dif = ' + Msg3;
   WriteLn(Msg1);
   Str(norma:8:8,Msg1); Str(n,Msg2);
   Msg1 := 'Norma = '+ Msg1 + espaco + 'Num. elem. part. = '+ Msg2;
   WriteLn(Msg1);
   Msg1 := '--------------------------------------------------';
   WriteLn(Msg1);
   {5*** atualizacao da somaAnterior e do contador}
   SomaAnterior := soma; n:= n+1;
 Until n > 10; {6** numero de testes  **}
 {7 *** estatistica final }
 Media := Media/n; Desvio := 0;
 For i := 0 to 10 Do
  Begin
    afastamento[i]:= Sqr(afastamento[i]-Media);
    Desvio := Desvio + afastamento[i];
  End; Desvio := Sqrt(Desvio);
 {8 *** saida de dados final }
 Str(Media:8:8,Msg1); Str(Desvio:8:8,Msg2); Str(Max:8:8,Msg3);
 Msg1:= 'Dif_Max =  '+ Msg3 + ' Desvio = '+Msg2+' Media=  '+Msg1;
 WriteLn(Msg1);
 Str(min:8:8,Msg1); Msg1 := 'Dif_min = '+ msg1+ '=========================';
 WriteLn(Msg1);
 Apeteco;
End.
```

Uma utilização de uma variável do tipo **array** pode ser o de guardar dados seqüencialmente. Este é o seu uso no "planejamento" acima. Matrizes podem ser usadas para guardar os pontos de máximo ou de mínimo, esta é outra solução que anunciei anteriormente.

Exercício: 15 *Cálculo de Máximos e mínimos e integrais.*

1. *Complete o planejamento acima e rode-o.* A solução é `R_Aleat1.pas`.

2. *Calcule*
$$\frac{1}{2}\int_0^2 x^2 + 3x + cos(x)dx;$$
subdividindo o intervalo de integração em 200 subintervalos iguais. Calcule a integral exatamente e compare o resultados.

3. *Calcule as integrais abaixo com grande precisão , (Δx pequeno):*

   (a) $\frac{1}{2}\int_0^2 x^3 + 3x\exp(x^3)dx$ (b) $\frac{1}{2}\int_0^2 \exp(x^{-2})dx$
   (c) $\frac{1}{2}\int_{-2}^2 \exp(x^{-2})dx$ (d) $\frac{1}{2}\int_{-10}^{10} \exp(x^{-2})dx$
   (e) $\frac{1}{2}\int_{-100}^{100} \exp(x^{-2})dx$ (f) $\frac{1}{2}\int_{-10}^{10} x^5\exp(x^{-2})dx$

4. *Faça uma tabela para distribuição normal de frequências com passo 0.1 e com amplitude 10.*

5. *Faça um programa que escolha a partir de um menu alguns tipos de integrais a serem calculadas, incluindo funções polinomiais até o 5º grau, trigonométricas, e funções mistas de polinômios e trigonométricas. O usuário deve poder escolher o intervalo, a precisão e o tipo de função.*

6. *Leia do disco minMax02 e rode-o. Modifique o programa incluindo um novo* loop *para que, sob decisão do usuário, o programa repita o cálculo com passo mais fino.*

7. *Modifique minMax02 para calcular mínimos. Solução: é* `MinMax03`.

8. *Modifique* `MinMax02` *para calcular máximos e mínimos.*

9. *Inclua as derivadas parciais de f e quando ambas forem nulas faça uma varredura circular em volta do ponto para determinar se é ponto de sela, de máximo local ou de mínimo local. O programa deve então registrar a análise feita (escrever na tela).*

10. *Modifique* `MinMax02` *incluindo uma matriz para memorizar os pontos em que ocorrem os máximos e os mínimos.*

11. *Calcule as integrais abaixo:*

$$\int_0^{200} e^{-x^2}dx$$

$$\int_{-200}^{200} e^{-x^2}dx$$

$$\int_{-200}^{200}\int_{-200}^{200} e^{-x^2-y^2}dxdy$$

Modifique minMaxXX para calcular integrais duplas, máximos e mínimos.

12. *Calcule a integral abaixo em que $\mathcal{D}$ é um anel delimitado por dois círculos de centro na origem com raios respectivamente 10 e 5:*

$$\int\int_{\mathcal{D}} e^{-x^2-y^2} dxdy$$

13. *Calcule a integral abaixo em que $\mathcal{D}$ é o disco de raio 10.*

$$\int\int_{\mathcal{D}} e^{-x^2-y^2} dxdy$$

14. *Calcule a integral abaixo em que $\mathcal{D}$ é a região do plano delimitada pelas párabolas $y = x^2 - 16$, $y = x^2$ e pelas retas $y = \pm 3$*

$$\int\int_{\mathcal{D}} e^{-x^2-y^2} dxdy$$

15. *Calcule o máximo e o mínimo das funções definidas nos exercícios anteriores nos respectivos domínios de integraçãop*

16. *Observando que* alguns *dos domínios acima são circulares, modifique seus programas para usar coordenadas polares.*

17. *Verificando as simetrias das funções nos exercícios acima, otimize os seus programas para calcular as integrais com maior precisão .*

18. *Calcule o volume da esfera de raio 1.*

19. *Calcule o volume da pirâmide cujos vértices são (2,0,0), (0,2,0) e (0,0,2).*

20. *Calcule*

$$\int\int_{\mathcal{D}} 2 - x - y - \sqrt{1 - x^2 - y^2} dxdy;$$

em que $\mathcal{D}$ é o triângulo de vértices (0,0), (0,2),(2,0).

21. *Verifique que a função $f(x,y) = \frac{x^2y+xy+x^3}{x^2+y^2}$ é limitada sobre o disco de raio 1 e centro na origem. Calcule*

$$\int\int_{\mathcal{D}} \frac{x^2y + xy + x^3}{x^2 + y^2} dxdy;$$

em que $\mathcal{D}$ é o disco de raio 1 e centro na origem e calcule Máximo e mínimo de f sobre $\mathcal{D}$.

22. *Considere a tabela de dados abaixo de uma função obtida por um levantamento de dados automático. Queremos encontrar a reta que melhor aproxima estes dados: $y = mx+b$: o coeficiente angular m deve ser o coeficiente angular médio e o coeficiente linear b deve ser o coeficiente linear médio. Calcule-os com um programa.*

f(-3.2000) = -2.5419	f(-2.8000) = -1.6849	f(-2.4000) = -0.4646
f(-2.0000) = 1.0931	f(-1.6000) = 2.5321	f(-1.2000) = 3.5475
f(-0.8000) = 4.3748	f(-0.4000) = 5.4893	f(-0.0000) = 7.0000
f(0.4000) = 8.5107	f(0.8000) = 9.6252	f(1.2000) = 10.4525
f(1.6000) = 11.4679	f(2.0000) = 12.9069	f(2.4000) = 14.4646
f(2.8000) = 15.6849	f(3.2000) = 16.5419	

Sugestão : use "array" e memorize os valores numa variável deste tipo.

23. *Discretize a função* $f(x) = \exp(x^2 - \frac{x}{2})(1+\frac{1}{x})^{-x^3}$ *com auxílio de alguma sucessão crescente ilimitada e verifique computacionalmente a existência de* $\lim\limits_{x=\infty} \exp(x^2 - \frac{x}{2})(1 + \frac{1}{x})^{-x^3}$.

24. *Quais das séries abaixo é convergente/divergente ?*

$$\sum_{1}^{\infty} \frac{1}{(1+1/n)^n}; \quad \sum_{2}^{\infty} \frac{1}{n^n-1}; \quad \sum_{1}^{\infty} \sqrt{n}\sin(\tfrac{1}{n^2});$$
$$\sum_{2}^{\infty} = \frac{1}{\log(n)}; \quad \sum_{1}^{\infty} \sqrt{n}\cos(\tfrac{1}{n^2}); \quad \sum_{2}^{\infty} \frac{\tan(\frac{1}{n})}{\log(n)};$$

25. *Discretize a função* $f(x) = \frac{\sin^2(x)-\log(1+x^2)}{1-\cos(x^2)}$ *com auxílio de uma sucessão adequada e calcule o limite:*

$$\lim_{x=0} \frac{\sin^2(x) - \log(1 + x^2)}{1 - \cos(x^2)}$$

26. *Encontre* α *tal que* $\lim\limits_{x=0} \frac{x-\sin(x+\alpha x^3)}{x^5}$ *exista.*

27. *As séries abaixo são* convergentes *ou* divergentes*?*

$$\sum_{1}^{\infty} = \frac{\log(n)}{n^2}; \qquad \sum_{1}^{\infty} n^{\frac{1}{3}}(1 - n\arctan(\frac{1}{n}));$$

28. ***Considere a série alternada* $\sum_{k=0}^{n} \frac{(-1)^n}{n+1}$. *Esta série converge condicionalmente, (não converge absolutamente).*

(a) *Faça um programa que calcule a soma parcial de ordem n, leitura de n pelo teclado.*

(b) *Faça um programa que some primeiro os termos de ordem par e depois os de ordem ímpar da soma parcial da série dada. Compare com o resultado anterior.*

(c) *Deduza dos resultados que a distributividade não se pode aplicar aos termos de uma série se a convergência da mesma não for absoluta.*

29. ***Faça um programa para testar a desigualdade*

$$|\int_a^b f(t)dt| \leq M\frac{(b-a)^2}{4}$$

sabendo que $|f'(x)| \leq M$; $x \in [a,b]$ *e que* $f(a) = f(b) = 0$.

30. *** Teste a desigualdade de Young*

$$xy \leq \int_0^x f(t)dt + \int_0^y g(t)dt;$$

em que f é uma função contínua estritamente crescente; $f(0) = 0$; *e g é a inversa de f com* $0 \leq x \leq a$; $0 \leq y \leq f(a)$

1.8 Algumas soluções de exercícios

Exercício 23: substitua $f(x)$ por $f(n)$ $n \in \mathbf{N}$ e use alguma variante do programa *Cauchy*. Não se esqueça de definir $n : LongInt$ ou possivelmente como "extended".

Exercício 24

diverge, termo geral não converge para zero;	converge;	converge
diverge: $\frac{1}{n} < \frac{1}{log(n)}$;	diverge;	diverge ;

No (exercicio, 25), página 76, substitua x por $1/n$ e use alguma variante do programa *Cauchy*

O seguinte trecho de programa resolve o (exercício, 28) sobre série alternada.

```
{**** Corpo do programa -  Rotina principal  *****}

Begin
  Write('Fim: ');Readln(fim);
  deltax := 0.01; soma1 := 0; soma2 := 0; soma3 :=0;
  ClrScr;
  n := 1;
  Repeat
     soma1 := soma1 + f(n);
     Inc(n);
  Until n > fim;
  WriteLn(soma1:4:4);Apeteco;
  n := 1;
```

```
    Repeat
      soma2 := soma2 + f(n);
      n:= n+2;
    Until n > fim;
    n := 2;
    WriteLn(soma2:4:4); Apeteco;
    Repeat
      soma3 := soma3 + f(n);
      n:= n+2;
    Until n > fim;
    WriteLn(soma3:4:4);Apeteco;
    WriteLn(soma2+soma3:4:4);Apeteco;
  End.
```

Observe que a soma das variáveis que guarda as somas parciais dos termos pares e dos termos ímpares resulta num valor igual ao da soma dos termos da série alternada, *porque só podemos somar com um programa, um número finito de termos e para um número finito qualquer de termos* sempre *vale a associatividade.* Entretanto cada uma das subséries diverge separadamente. *Prove isto* e portanto não vale o teorema "soma dos limites é o limite da soma". A única função positiva deste programa é sugerir que as duas somas parciais guardadas em "soma2" e "soma3" divergem, porque crescem em valor absoluto.

O programa *Riem9* é uma sugestão para o (exercicio, 29). Observe que o programa referido não demonstra a desigualdade, apenas a testa em alguns casos e ainda assim para um intervalo dado. O programa pode ser "generalizado", permitindo que os extremos do intervalo sejam fornecidos pelo teclado.

Modifique substancialmente... o programa *Riem9* para fazer o teste do (exercicio, 30). Observe que o programa referido não demonstra a desigualdade.

1.9 Perguntas que ficaram sem resposta

Vou responder aqui a *algumas* perguntas que ficaram sem resposta. Todos os assuntos se encontram indexados de forma que você possa retornar aos mesmos em algum momento mais oportuno porque nem sempre as respostas podem ficar inteiramente claras no *seu* atual contexto.

1.9.1 Sinais diacríticos

Os sinais diacríticos da língua brasileira. . Veja a biblioteca `Tipos.pas`, onde uma solução para este problema foi produzida. A tabela ASCII estendida contém os sinais diacríticos de diversas línguas. Por exemplo, vá para o final do arquivo `Tipos.pas` lá você irá encontrar:

$$eagu := Char(130);$$

A variável $eagu$ é alí inicializada com este valor. Na tabela ASCII, o código 130 corresponde ao carácter "é". De forma análoga estão definidas as outras variáveis: aagu, iagu etc... Se agora você olhar a rotina "Mask", que produz o nosso carimbo nos programas, irá ver:

```
  versao   := '2.1995.'; autor := 'Prof. Tarcisio Praciano Pereira';
  Local    := 'Dep. de Matem. - Univ. do Rio Grande - RS ';
  Projeto  := 'Monitor Eletr'+chr(147)+'nico de C'+Chr(160)+'lculo
- vers'+Chr(132)+'o :'+ versao;
  Programa := 'Laborat'+Chr(162)+'rio Experimental de C'+Chr(160)+'lculo';
```

ou, outra versão:

```
  versao   := '2.1995.'; autor := 'Prof. Tarcisio Praciano Pereira';
  Local    := 'Dep. de Matem. - Univ. do Rio Grande - RS ';
  Projeto  := 'Monitor Eletr',ocir,'nico de C',aagu,'lculo
- vers',atil,'o :'+ versao;
  Programa := 'Laborat',oagu,'rio Experimental de C',aagu,'lculo';
```

dependendo da versão da `Tipos.pas`. O segundo bloco corresponde a uma versão mais recente. A segunda versão gasta mais memória no computador, entretanto sobrecarrega menos a memória do programador... A escolha do método é sua.

1.9.2 Ativando o Ctrl-Break

Quando um programa se encontra em *loop infinito* se pode pará-lo, sem desligar o computador, com Ctrl-Break. Mas é preciso ativar a função que verifica chamadas ao Ctrl-Break. Isto se faz dando-se o valor

$$CBreak := True;$$

à variável $CBreak$, logo depois do **Begin** que marca o início do algoritmo. Esta variável está definida na biblioteca **Crt**, portanto o programa deve ter esta biblioteca declarada dentro do **Uses**.

Se você/ desativar o Ctrl-Break, seu programa rodará mais rápidamente e você deverá fazê-lo quando tiver certeza de que ele não tem erros, ou *menos erros...* enquanto o programa estiver em testes, dê o valor

$$CBreak := False;$$

a variável.

1.9.3 Personalizando o Pascal

Consulte no seu manual do Pascal o assunto "personalizando o Pascal", (em inglês, customizing). É um programa que se encontra no disco do Pascal chamado *Tinst*, que faz isto. Você pode até tentar usá-lo, sem receios de estragar o Pascal. O limite para modificações não vai além de A_{24}^3 ...

Desviei esta observação para cá devido a um risco que você corre. Nada grave e você é incentivado a usar o *Tinst*, mas pode ser que as modificações terminem por atrapalhá-lo na primeira vez que as fizer. Para evitar chateações, primeiro faça um *backup* do TurboPascal, porque *Tinst* grava as modificações num arquivo de *overlay* que é usado pelo programa para se inicializar. Faça as modificações com *Tinst* e veja se elas o satisfazem. O próprio *Tinst* tem como desfazer as modificações, mas se você não descobrir como, copie de volta o TurboPascalda sua cópia de reserva e recomece as modificações novamente até descobrir uma personalização que o agrade.

Derivada primeira, derivada segunda aproximadas e derivadas formais com uma função fortemente oscilante.

1.10 Tabelas

Exemplo: 19 *Precisão das derivadas*

```
f :=  (x*x*x + 3*x*x + 3*x + 1)*sin(3*x);
Amostragem No intervalo [-5.00 , 5.00 ]
======================================
 x =      -5.00
Der. por convolucao ------->   114.64485632
derivada formal ----------->   114.64626294
quoc. de difer. ----------->   114.64623405
-------------------------------------
derivada seg. formal ------>   171.38258519
Der. seg. por convolucao -->  -508.19940568
quoc. difer. de seg ord.--->  -577.75210453
========================================
 x =      -4.00
Der. por convolucao ------->   -53.86429107
derivada formal ----------->   -53.86470187
quoc. de difer. ----------->   -53.86468900
-------------------------------------
derivada seg. formal ------>    -3.34119028
Der. seg. por convolucao -->   224.32020104
quoc. difer. de seg ord.--->   257.43415312
========================================
 x =      -3.00
Der. por convolucao ------->    16.92185307
derivada formal ----------->    16.92170446
quoc. de difer. ----------->    16.92169995
-------------------------------------
derivada seg. formal ------>   -30.98342610
Der. seg. por convolucao -->   -78.16230629
quoc. difer. de seg ord.--->   -90.32816146
========================================
 x =      -2.50000000
Der. por convolucao ------->    -9.84041516
derivada formal ----------->    -9.84118244
quoc. de difer. ----------->    -9.84118274
-------------------------------------
derivada seg. formal ------>    50.97247946
Der. seg. por convolucao -->   -10.33035800
quoc. difer. de seg ord.--->    -6.00933083
========================================
 x =       0.00
Der. por convolucao ------->     3.00012275
derivada formal ----------->     3.00000000
quoc. de difer. ----------->     3.00000090
-------------------------------------
derivada seg. formal ------>    18.00000000
Der. seg. por convolucao -->    15.20247393
quoc. difer. de seg ord.--->    18.00026999
========================================
 x =       1.50
Der. por convolucao ------->   -28.20724559
derivada formal ----------->   -28.20974280
quoc. de difer. ----------->   -28.20973785
-------------------------------------
derivada seg. formal ------>  -175.84265200
Der. seg. por convolucao -->    98.09922973
quoc. difer. de seg ord.--->    99.09318842
========================================
 x =       2.00
Der. por convolucao ------->    70.22801932
derivada formal ----------->    70.22957477
quoc. de difer. ----------->    70.22958569
-------------------------------------
derivada seg. formal ------>    82.62014141
Der. seg. por convolucao -->   185.34077491
quoc. difer. de seg ord.--->   218.41264935
========================================
 x =       3.00000000
Der. por convolucao ------->  -155.15147959
derivada formal ----------->  -155.15532299
quoc. de difer. ----------->  -155.15534748
-------------------------------------
derivada seg. formal ------>   -15.13442428
Der. seg. por convolucao -->  -422.37580537
quoc. difer. de seg ord.--->  -489.88645847
========================================
 x =       3.50000000
Der. por convolucao ------->  -183.43010010
derivada formal ----------->  -183.44142511
quoc. de difer. ----------->  -183.44139889
-------------------------------------
derivada seg. formal ------>  -918.54548092
Der. seg. por convolucao -->   499.69182369
quoc. difer. de seg ord.--->   524.40041073
========================================
 x =       4.50
Der. por convolucao ------->   369.86294343
derivada formal ----------->   369.88321278
quoc. de difer. ----------->   369.88317013
-------------------------------------
derivada seg. formal ------>  1554.02589296
Der. seg. por convolucao -->  -811.51721412
quoc. difer. de seg ord.--->  -853.15212311
========================================
 x =       5.00
Der. por convolucao ------->  -422.03626808
derivada formal ----------->  -422.04668080
quoc. de difer. ----------->  -422.04676745
-------------------------------------
derivada seg. formal ------>   795.29215598
Der. seg. por convolucao --> -1528.20161662
quoc. difer. de seg ord.---> -1733.00404374
=======================================
```

Capítulo 2

Raízes aproximadas de uma função

Solução aproximada de equações "algébricas"

Introdução

O capítulo 2, do ponto de vista de programação em Pascal representa um aprofundamento significativo. Nele vou discutir o resto das estruturas de dados que Pascal oferece com exceção de `ponteiros` que não discutirei neste livro.

Um pouco mais de Matemática vai se encontrar como pano fundo. Se você desejar apenas aprender Pascal, *passe por cima* dos conceitos matemáticos e estude direto os programas, claro aceitando a Matemática sem demonstrações. Tentei descrever os conceitos antes das demonstrações para deixá-los aceitáveis para quem não os quisesse estudar a fundo. Também escrevi as demonstrações, a Matemática em geral, em letra pequena para facilitar sua decisão, se quiser evitá-la numa primeira leitura.

2.1 Raízes de uma função

Vou discutir neste capítulo o seguinte problema:
Dada uma função

$$f : \mathbf{R}^n \longrightarrow \mathbf{R}^m; f = (f_1, \ldots, f_m)$$

de classe $\mathcal{C}^2$, *quero determinar* aproximadamente *todos os valores de* x *tais que* $f(x) = 0$.

Posto nestes termos o problema é aberto, *e no momento não há menor pista de como ele se poderia resolver. Nos últimos anos se vem desenvolvendo uma área de pesquisa em Matemática que é uma projeção da Inteligência Artificial chamada Computação Algébrica com a qual certos problemas deste tipo encontram até mesmo uma solução exata.*

Programas como Derive, Maple, Mathematica, *entre outros, são exemplos de programas de Computação Algébrica que conseguem resolver equações com exatidão em muitos casos. Eles também têm incorporadas rotinas numéricas e o usuário pode decidir se deseja que a solução seja tentada diretamente via cálculo numérico ou não .*

A produção de rotinas de Computação Algébrica se encontra afastada do escopo de um livro como este, pois envolveria um aprofundamento muito maior, tanto em Álgebra abstrata *como em* Computação *, pela exigência da construção de* objetos *computacionais avançados e uso intenso de estruturas algébricas . A tendência de cursos deste tipo, Cálculo Numérico, é de que aos poucos cedam lugar para cursos de* Computação Algébrica*, certamente.*

Retornando ao problema $f(x) = 0$*, vejamos que é possível simplificá-lo. Se* $f(x) = 0$; $f = (f_1, \ldots, f_m)$ *então*

$$\forall\ i \quad f_i(x) = 0$$

e reciprocamente. Temos assim um "problema mais simples":

$$f(x) = 0\ ;\ f : \mathbf{R}^n \longrightarrow \mathbf{R}.$$

Considerada a equação acima, sempre dentro da nossa hipótese de alta diferenciabilidade, podemos escrever:

$$\frac{\partial f}{\partial x_1} dx_1 + \ldots + \frac{\partial f}{\partial x_n} dx_n = 0$$

e se entendermos "dx" como representando uma aproximação em volta de um ponto a, isto é, $x - a$ *é pequeno, podemos interpretar a expressão diferencial acima como:*

$$\frac{\partial f}{\partial x_1}(x_1 - a_1) + \ldots + \frac{\partial f}{\partial x_n}(x_n - a_n) = 0,$$

que identificamos como a equação de um hiperplano *tangente à* variedade diferenciável

$$f(x) = 0$$

no ponto $(a_1, \ldots, a_n)$.

Definição : 1 *Variedade, hiperplano e hipersuperfície.*

A palavra "variedade" representa "espaço" num sentido mais amplo que o geométrico que se encontra confinado no tridimensional.

Planos são "subvariedades lineares maximais próprias" no espaço de dimensão três, ou ainda "variedades lineares de dimensão 2. Analogamente, os hiperplanos são as "subvariedades lineares maximais próprias" num espaço de dimensão n qualquer, ou ainda as variedades lineares de dimensão $n-1$ *deste um espaço. Assim, as retas são os "hiperplanos dos planos", (*espaços de dimensão dois*). E os "planos são os hiperplanos dos espaços tridimensionais", e um "espaço tridimensional é um hiperplano dum espaço de dimensão quatro" se estiver contido nele.*

O adjetivo próprio *significa que não pode ser igual ao universo que se estiver considerando.*

Uma superfície é uma subvariedade maximal própria do espaço de dimensão três. Os planos são superfícies: variedades lineares. Hipersuperfície generaliza a palavra superfície. Assim, as hipersuperfícies são as subvariedades maximais próprias num espaço de dimensão qualquer. As curvas são as hipersuperfícies dum plano, por exemplo.

Hiperplano e hipersuperfície são conceitos relativos a uma determinada dimensão, são as variedades máximais.

Hiperplanos são variedades lineares de dimensão $n-1$, *em que n é a dimensão do espaço que se estiver considerando, e conseqüentemente o objeto a que eles forem tangentes têm idêntica dimensão,* pelo menos localmente*, sendo então uma hipersuperfície do* $\mathbf{R}^n$*, ou uma variedade* diferenciável *de dimensão* $n-1$*. Isto significa que o "número" de soluções da equação* $f(x)=0$ *pode ser* infinito*: cortar uma hipersuperfície com um hiperplano pode ter como solução uma curva.*

Este problema está longe de ser "simples"... Mas com auxílio do Teorema da função Implícita *podemos "resolver" a equação* $f(x)=0$ *em* x_i

$$x_i = g((x_k)_{k\neq i})\ ;\ g:\mathbf{R}^{n-1}\longrightarrow \mathbf{R}$$

localmente. Assim reduzimos a dimensão da variedade¡ *de uma unidade e portanto podemos,* sempre localmente*, reduzir sucessivamente o problema até chegar em:*

$$f(x)=0\ ;\ f:\mathbf{R}\longrightarrow \mathbf{R}, \tag{2.1}$$

Será esta versão simplificada do problema que irei discutir aqui. Fica claro que a versão n-dimensional pode ser tratada localmente se soubermos resolver o caso em dimensão 1, entretanto isto não quer dizer que a solução do problema simplificado conduz imediatamente a uma solução do problema geral. Isto está longe de ser verdadeiro. Fiz esta introdução apenas para mostrar-lhe a imensidão das águas em que se pode nadar...

Do ponto de vista do Cálculo Numérico, o problema é muito frágil no sentido de que as técnicas conhecidas não podem ser transformadas em programas *para funcionarem sem monitoramento. Todos os programas que conheço, e naturalmente todos os algoritmos que puderem ser encontrados aqui, dependem de uma decisão inteligente do usuário sobre a existência de uma solução em algum determinado momento. Basta que se tome como exemplo uma função* fortemente oscilante *ou que seus valores sejam muito pequenos para perturbar qualquer programa. Situações deste tipo são bem representadas pelo* problema *da separação dos zeros de uma função: você verá logo em seguida programas que apresentam vários zeros para uma função que tem um zero apenas.*

Dei preferência, portanto, às técnicas globais de análise de funções , que são no fundo o que há de mais importante, uma vez que mostram um método *que pode ser usado em distintas situações, por exemplo na construção de gráficos. Acho que é mais significativo estudar os métodos do que sugerir soluções que não temos.*

Os programas desta seção se caracterizam por percorrer um intervalo fazendo a "variável" saltar de passo em passo*. Este passo em geral não será um número inteiro, e com freqüência* quanto menor for*, melhor será o resultado obtido na varredura,* se pudermos desprezar o tempo que o computador vai gastar para executar o programa*.*

Esta é a técnica básica. Com uma varredura grossa, se pode, na maioria das vezes, sem nunca desprezar o monitoramento do usuário*, determinar intervalos onde existam raízes. Estes programas podem ser aperfeiçoados para, com uma varredura fina, associada a um dos métodos clássicos, que vou discutir na segunda parte do capítulo, produzir soluções mais precisas. É interessante observar que estou seguindo de perto uma receita de Dieudonné, o* bourbakista nuclear, (- ,1993) *, veja [4, capítulo 2].*

Observação: 14 *Tempo de processamento*

Hoje já podemos nos preocupar mais com algoritmos eficientes, deixando um pouco de lado a preocupação com o tempo de sua execução, uma vez que a tecnologia de arquitetura de computadores e da microeletrônica evoluiu muito, nos oferecendo máquinas muito velozes. Duas técnicas ligadas ao hardware *se tornaram populares recentemente: multiprocessadores e processamento distribuído. A primeira consiste numa arquitetura de computadores contendo mais de um processador e a segunda na conexão de vários computadores entre si, em rede. Na medida em que estas tecnologias se tornem mais comuns e as linguagens de processamento também incorporem comandos que permitam o acesso a elas, a velocidade de processamento irá aumentar ainda mais. O processamento distribuído parece estar mais* *acessível porque usa o que já existe, entretanto ainda há pouca gente conhecendo a técnica e a utilização. As linguagens de programação ainda não incorporaram, para o grande público, o uso destas novas ferramentas, mas elas já existem, à sua volta, em* LinuX, *ver [10].*

Aqui começa se perpetrar mais um crime contra nós que vivemos nos países em desenvolvimento. Governos míopes não entendem que o futuro se inicia hoje e não investem num ensino moderno e ligado ao conhecimento tecnológico atual.

O problema básico deste capítulo se caracteriza como típico de processamento paralelo e poderia ser um exercício de aprendizado desta técnica de processamento:

1. *busca preliminar para detectar intervalos onde existam raízes.*
2. *Posteriormente a* busca paralela *refinada de raízes, nos distintos subintervalos.*

Não vou discutir processamento paralelo aqui porque TurboPascal não é uma linguagem de processamento capaz de executá-lo. Chamarei sua atenção para o assunto no ponto certo.

Vocabulário: 19 *Hardware, processador, sistema operacional, software.*

1. **Hardware** *Do inglês, hard = duro, em oposição a soft = macio, para fazer referência à parte da computação que diz respeito aos componentes eletrônicos ou os* programas *neles gravados em forma permanente.*

2. **Processador** *Entre os componentes eletrônicos de uma computador existe um, às vezes alguns, que formam o centro da máquina e que contêm o programa básico que a define: suas operações aritméticas e lógicas. Este componente é comumente chamado de "CPU" - a Unidade de Processamento Central (*<u>C</u>entral <u>P</u>rocessing <u>U</u>nity*). Entre a CPU e o usuário, ainda há o* sistema operacional *que é outro programa, algumas vezes gravado em forma permanente, que gerencia o uso da comunição entre os periféricos: video, drives, teclado, impressoras etc...* `LinuX` *é um exemplo de* `sistema operacional`*.*

3. **Software** *Em oposição ao hardware, referência feita aos programas ou à parte que pode ser apagada ou reescrita no processamento de dados. São definições frágeis, ou difusas, essas que demos de* software, hardware*, obviamente. Uma linguagem de programação está fora de acesso para ser reescrita, ou de alguma forma, ser manipulada, pelo usuário, mas é comumente considerada "software"... talvez porque venha distribuída em discos flexíveis...*

4. **Sistema operacional** *É um programa que se situa entre a CPU e os programas que o usuário normalmente usa. Hoje em dia eles atingiram uma tal evolução que podem trabalhar*

silenciosamente *sem que o usuário precise sequer tomar conhecimento da existência deles.* `LinuX, FreeBSD, Unix, DOS` e `WindowsNT` *são exemplos de sistemas operacionais.* `LinuX` e `FreeBSD` *são sistemas operacionais de domínio público que podem ser adquiridos via CD ou transferidos pela Internet, consulte por exemplo:*

(a) **www:http://www.cdrom.com** *é um nó comercial onde você pode conseguir LinuX ou FreeBSD gratuitamente, se quiser transferir diretamente os arquivos, ou comprar o CD contendo os programas.*

(b) **www:http://www.frebsd.org** *é o* home-site *do FreeBSD na Universidade de Berkeley.*

Ver shareware, freeware.

2.2 Procurando raízes com varreduras

Vou iniciar o nosso estudo da busca de raízes de uma função com dois objetivos iniciais: *(1) mostrar o método computacional básico que é varredura; (2) mostrar as dificuldades naturais do problema.*

Algumas palavras novas:

Vocabulário: 20 Const , = , While/Do, { , }

1. **Const** *indica o início da área em que você vai definir as* constantes *do programa. Pascal não vai aceitar que elas mudem de valor durante a execução.*
2. = *Observe que o sinal* **"="** *em vez de* **":="** *para a atribuição de valor às constantes. O sinal "=" tem dois usos distintos em Pascal: (1) para atribuir valores às constantes como aqui; (2) comando lógico na verificação de igualdade entre dois símbolos.*
3. **While/Do** *é uma outra maneira de fazer loops. Sintaxe:*

   While *"condição lógica"* **Do** *"comando".*

   Então "comando" será executado enquanto *a "condição lógica" for verdadeira, (while = enquanto). No exemplo abaixo a "condição lógica" é "$x < a$".*

   Veja pela sintaxe a diferença entre **While/Do** *e* **Repeat/Until***:*

While/Do primeiro `condicao` *é testada e* enquanto *for verdadeira* `comando` *será executado:*

While *condicao* **Do**
comando;

Repeat/Until `comando` *será executado* até *que* `condicao` *se verificar falsa:*

Repeat
comando;
Until *condicao;*

conseqüentemente no caso **Repeat/Until**, `comando` *será executado pelo menos uma vez. No caso* **While/Do**, `comando` *pode não ser executado nenhuma vez: basta que* `condicao` *seja falsa logo no começo.*

Além disto **Repeat/Until** *dispensa Begin/End; ao passo que* **While/Do** *não o dispensa se tiver que executar um comando composto.*

Veja outra forma de comparar estas duas estruturas de controle do fluxo, observe a reversão lógica que `condicao` *sofre:*

```
While   condicao Do
        Begin
                Comando1;
                Comando2;
        End;
Repeat
        Comando1;
        Comando2;
Until   (Not condicao);
```

Esta reversão da condição lógica algumas vezes pode ser utilizada para otimizar o programa.

4. **Begin/End;** *em geral essencial nas estruturas de controle do tipo*

   While/Do, If

   para marcar o início e o fim de um bloco *de comandos que desejamos considerar como um* `comando composto`. *Por exemplo, experimente tirar o Begin/End; do programa abaixo, ele não vai mais parar...(*Aperte CTRL-C ou CTRL-Break, você entrou num *loop* infinito... *)*

5. **{ , }** *, chaves, marcam o início e o fim de um comentário. Comentários são ítens essenciais num programa. Todo programa deveria começar com um comentário indicando os seus*

objetivos e possivelmente a técnica empregada para desenvolver o algoritmo, o nome do programador, a data em que o programa foi feito e sua última modificação. Todos estes dados podem ficar arquivados como comentários no início do programa como está feito no programa abaixo. Junto com os comentários você pode arquivar temporariamente dados sobre o funcionamento do programa, temporária ou definitivamente.... *Veja os comentários, abaixo, no programa que não funciona.*

Exemplo: 20 *Um programa que não funciona ...*

Digite o programa abaixo e rode-o. Você não verá resultado algum. Por que?

Observe que **sin** *é a palavra do Pascal para a função trigonomé-trica* **sen**.

```
{** Programa para calcular zeros de uma funcao
 ** Varre  [-3 , 3] com passo 0.01 procurando x / sen(x)=0
 ** Possui erro: Nao acha nenhum valor para x
 ** A saida de dados eh vazia.
 ** Objetivo do programa: ilustrar
 ** o uso deficiente da igualdade em testes.   **
 ** Autor Tarcisio Praciano-Pereira - 1992     **}

Program Raizes01;
Uses Crt;
Var
   x : Real;
Const
     a = -0.0000000001;
     b = 0.0000000001;
     passo = 0.00000000001;
Begin
     ClrScr;
     x :=  a;
     While x < b Do
     Begin
          If sin(x) = 0  Then WriteLn(x,' => ', sin(x));
          x :=  x + passo;
     End;
     WriteLn('FIM DA VARREDURA');
     ReadLn; {para o programa, esperando ler alguma coisa}
End.
```

O comando **WriteLn***('Fim da Varredura'); é essencial, pois é tudo que o programa vai dizer...*

O **ReadLn** *ao final tem como efeito manter os dados na tela para que se possam ver os resultados impressos. Experimente apagá-la e rode o programa.*

Por que o programa não funcionou? Na verdade ele funcionou... o que não fez foi encontrar x tal que sen(x) $= 0$. Se você atribuir a passo um valor muito pequeno é possível que o programa encontre $x \neq 0$ tal sen(x) $= 0$ por que para valores muito pequenos de x, *sen(x), 0 , sen(0)* são idênticos para o computador.

Isto exemplifica como as condições do tipo "igualdade" são perigosas, pois são muito difíceis de serem satisfeitas: *é muito difícil de se resolver uma equação, é bem mais fácil resolver-se uma desigualdade*, teoricamente, pelo menos. Isto sugere uma modificação para se obter um programa que funcione.

Exercício: 16 *A diferença entre igualdade e desigualdade.*

1. *Substitua a condição lógica do "If" por*
   $$Abs(\ sin(x)) < delta;$$
   definindo antes delta *como constante ou variável.*

2. *Porque o programa entra em* loop infinito *se for tirado o par Begin/end que se segue ao While/Do ?*

3. *Substitua a função sin por outras e teste o programa exaustivamente.*

Exemplo: 21 *Determinação de zeros aproximados*

Neste programa estou resolvendo um dos exercícios acima.

```
Program Raizes02;
Uses Crt;
Const
     a     = -6.55;
     b     = 6.5;
     passo = 0.1;
     delta = 0.3;
Var
   x : Real;
Function f(x: Real): Real;
Begin
     f := sin(x);
End;
Begin
     ClrScr;
     x :=  a;
     While  x  <  b  do
     Begin
          If Abs(f(x)) < delta Then WriteLn(x,'=> ',f(x));
          x :=  x + passo;
```

```
    End;
    WriteLn('FIM DA VARREDURA');
    ReadLn;
End.
```

Exercício: 17 *Uso de funções e desigualdades: laboratório.*

1. *Rode Raizes02 e analise o resultado. Deveria encontrar as raízes de sen no intervalo* $[-6.55, 6.5]$, *por que achou apenas duas raízes* $x = -3.15$ *e* $x = 3.14$.

2. *Substitua a = -6.58 e veja qual o resultado.*

3. *Substitua a = -6 e veja qual o resultado.*

4. *Como os resultados não podem depender do ponto inicial do intervalo, então deve haver outra razão para o fenômeno observado nos exercícios anteriores. Substitua: delta = 0.1 e rode o programa com os valores iniciais* $a \in \{6.58, -6.55, -6\}$. *Tente encontrar uma justificativa que associe* delta, passo .

5. *Use Raizes02 para encontrar as raízes de*
$$f(x) = x^2 + 3x + 2.$$

6. *Modifique Raizes02 para encontrar as raízes de*
$$f(x) = x^2 + 3x + 2$$
com "grande" precisão em cada intervalo em que houver raíz.

7. *Encontre todas as raízes de*
$$f(x) = x^3 + 3x^2 + 2x - 5.$$

8. *Encontre todas as raízes de*
$$f(x) = x^3 + 3x^2 + 2x - 5$$
com boa aproximação.

9. *Encontre as raízes de*
$$f(x) = x^5(x^2 + 3x + 2)^2.$$
Se tiver algum problema analise o tamanho do `delta`*...*

2.2.1 Retângulo *passo* x *delta*.

Vou analisar alguns problemas que você teve na solução dos exercícios anteriores. Você deve ter observado que, ao modificar a relação de grandeza entre *passo, delta* a saída de dados aumentava ou diminuía em quantidade de "raízes". Com um passo muito pequeno se multiplicam as "raízes" que ficam próximas da *raiz exata*. Este é o *problema* que mencionei na introdução deste capítulo: *a separação das raízes*. Simplesmente rodando um programa, não sabemos se a função tem várias raízes bem próximas umas das outras...

Eis alguns resultados que você pode ter obtido com as experiências sugeridas nos exercícios.

```
x <- [ -6.58 , 6.55]      passo =0.1000 delta=0.0300

f(-6.2800) = 0.0032
f(0.0200) = 0.0200
f(3.1200) = 0.0216
```

Neste caso, não foram encontradas raízes aproximadas nas vizinhanças de $\pi, -2\pi$.

```
x <- [ -6.55 , 6.55]       passo =0.1000 delta=0.0300

f(-3.1500) = 0.0084
f(3.1500) = -0.0084
```

Neste caso, perdemos as raízes aproximadas nas vizinhanças de $-2\pi, 0, 2\pi$.

A diferença entre os dois casos reside na última casa decimal do ponto inicial 5 ou 8, para o mesmo valor do *passo*, as demais condições são iguais. O que ocorreu foi que o programa, saltando a razão de 0.1 e trabalhando com a margem de erro 0.03, perdeu raízes aproximadas que excedessem esta distância de zero. Se você repetir estas duas experiências com $delta = 0.5$ verá aparecer as raízes que não apareceram num caso ou no outro.

Vemos que a procura de raízes por varreduras precisa ser tornada mais sofisticada: é insuficiente incrementar uma variável e fazer um teste de proximidade com zero. As próximas seções se dedicam à especialização deste algoritmo, mas é preciso insistir no fato de que *não temos meios de resolver este problema*, os programas continuarão na dependência de decisões do usuário.

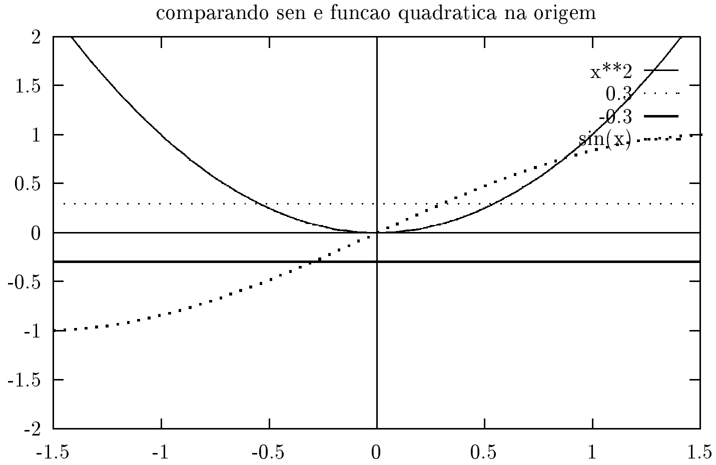

Figura 2.1: O retângulo *delta* `versus` *passo* e o coeficiente angular da função no ponto.

O gráfico comparando a função *sen* e a função quadrática, (fig. 2.2.1), sugere qual é razão da multiplicidade de raízes encontradas, ou a falta delas. A função *sen* corta os eixos com coeficiente angular 1 (em módulo), conseqüentemente, próximo de uma raiz o gráfico de *sen* é muito semelhante a uma reta de coeficiente angular ± 1.

Vamos designar por *epsilon* o erro na expressão :

$$abs(f(x)) < epsilon$$

e *passo* é o salto de varredura, portanto

$$x := x + passo$$

é o incremento da variável dentro do programa. Observe que estamos estudando as raízes do *seno*. As possibilidades que temos de encontrar raízes são :

1. Se $epsilon = passo$ então o programa pode pescar as raízes dependendo do ponto inicial de varredura.

2. Se $epsilon < passo$, corresponde a figura (fig. 2.2.1) diminuem as possibilidades de serem pescadas raízes aproximadas

ainda dependendo do ponto inicial do intervalo. Por exemplo se *epsilon* $= \frac{passo}{2}$ e *passo* for pequeno se o retângulo ficar centrado na origem o programa não irá, com certeza, encontrar nenhuma raiz em volta do ponto $x = 0$. Observe que esta desigualdade significa, geometricamente, "entrar no retângulo".

3. Se *epsilon* for muito maior que *passo* então muitas "raízes" próximas das raízes exatas serão "pescadas".

Exemplo: 22 *A separação dos zeros.*

Vamos rodar Raizes02 *com uma equação polinômial e analisar o resultado que será esclarecedor:* $f(x) = x^2$.

```
x <- [ -3.20 , 3.20] * f(x)= x^2  *  passo:=0.1 delta:=0.1

f(-0.3000)= 0.0900  f(-0.2000)= 0.0400  f(-0.1000)= 0.0100
f(-0.0000)= 0.0000  f(0.1000)=  0.0100   f(0.2000)= 0.0400
f(0.3000) = 0.0900
```

É isto que você pode constatar na seguinte listagem de dados, em que foram encontradas 4 raízes aproximadas nas vizinhanças de cada raiz exata:

```
x <- [ -6.58 , 6.55]    passo =0.05 delta=0.1

f(-6.3800) = -0.0967     f(-6.3300) = -0.0468
f(-6.2800) = 0.0032      f(-6.2300) = 0.0532

f(-3.2300) = 0.0883      f(-3.1800) = 0.0384
f(-3.1300) = -0.0116     f(-3.0800) = -0.0616

f(-0.0800) = -0.0799     f(-0.0300) = -0.0300
f(0.0200) = 0.0200       f(0.0700) = 0.0699

f(3.0700) = 0.0715       f(3.1200) = 0.0216
f(3.1700) = -0.0284      f(3.2200) = -0.0783

f(6.2200) = -0.0631      f(6.2700) = -0.0132
f(6.3200) = 0.0368       f(6.3700) = 0.0867
```

Isto nos mostra que a análise do coeficiente angular da função f tem que ser usada para construirmos um bom programa que encontre raízes aproximadas de uma função. Veja mais um exemplo:

Exemplo: 23 *Rodando* `Raizes02` *com passo = delta*

Rodando `Raizes02` *com passo = delta, com a mesma função* $f(x) = x^2$, *o programa encontrou 7 raízes na vizinhança da* *raiz exata $x = 0$, com a mesma Esta função tem uma tangente horizontal no ponto $x = 0$, neste ponto sua derivada se anula. Conseqüentemente Δf é muito menor que Δx nas proximidades da raíz, o que leva o programa a encontrar muitos valores de f com pequena oscilação quando se aproxima da raiz exata.*

O gráfico (fig. 2.2.1) ilustra isto, com duas funções bem apropriadas, as situações aqui discutidas:

caso do seno o coeficiente angular no *zero* é diferente de zero ou

caso da parábola quando o coeficiente angular no *zero* é zero.

Este parágrafo se caracterizou por aspectos laboratoriais. Era preciso sentir quais as dificuldades existentes na busca de raízes. No próximo parágrafo vou discutir os métodos que podem tornar as varreduras mais habilitadas na busca de raízes.

2.3 Métodos clássicos

2.3.1 Revisão dos métodos clássicos

Há alguns métodos clássicos de busca de raízes. Vou me fixar aqui nos seguintes:

1. *método (de Newton) da secante,*
2. *método da tangente - Newton-Raphson,*
3. *busca binária,*

que vou discutir de um ponto de vista teórico incluindo trechos de programas que os representem.

Os métodos clássicos para encontrar raízes aproximadas levam em conta os seguintes pontos numa análise preliminar:

1. *o tamanho de $f(x)$ em valor absoluto;*
2. *trocas de sinal de $f(x), f(x + \Delta x)$;*
3. *o tamanho de $f'(x)$ em valor absoluto.*

Há dois objetivos que em algum sentido são contraditórios:

1. *filtrar raízes repetidas,*
2. *encontrar todas as raízes.*

Estes objetivos são , em geral, demasiado ambiciosos. Já me referi às funções muito oscilantes *como um problema para a determinação de todas as raízes. Experimente usar uma função como:*

$$f(x) = \frac{sin(nx)}{n^2}(x-3)(x+3) \tag{2.2}$$

em algum dos programas `RaizXX`*, com grande valor de n. Como a parábola tem duas raízes simétricas, o seu extremo se dá no zero e vale* $\frac{9}{n^2}$*. Uma tal função terá muitas raízes no intervalo* $[-3,3]$*, se* $n = 9$ *terá mais de 14 raízes, enquanto que a amplitude máxima da função neste intervalo*[1] *será menor* $\frac{1}{9}$*, rode o programa* `Raiz02` *e veja se ele consegue encontrar todas as raízes.*

Também, pela sua pequena amplitude qualquer programa irá encontrar uma grande quantidade de raízes aproximadas *na vizinhança de uma raiz exata:* problema da separação de raízes.

Não há dúvida que os dois objetivos acima entram em choque um com o outro. Em algumas ocasiões um exclue o outro: funções que tenham alta freqüência (raízes muito próximas entre si) e que oscilem muito suavemente dificultam a separação das raízes*, o usuário do programa terá que intervir para escolher as raízes múltiplas que o programa venha apresentar nas vizinhanças de uma raiz exata. Os exercícios ao final desta seção o levarão a experiências desta natureza com o próximo programa: RaizFin2,* `(RaizFina - versão 04)` [2]*.*

Isto nos mostra que o problema central é outro: interessa menos encontrar todas as raízes do que saber:

os intervalos em que f tem raízes.

os intervalos em que $|f(x)| < \epsilon$, para $\epsilon > 0$ dado.

A segunda questão é muito importante, saber quando f é pequena. Voltarei a esta questão depois de discutir todos os

[1] Por causa do fator $sen(nx)$, a função se anula na origem onde se daria o extremo da parábola

[2] Atualmente há apenas duas versões deste programa cada uma com uma variante que escreve os dados em disco: RaizFin1/2 e RaizFin5/6, uma dessas versões está no final desta seção

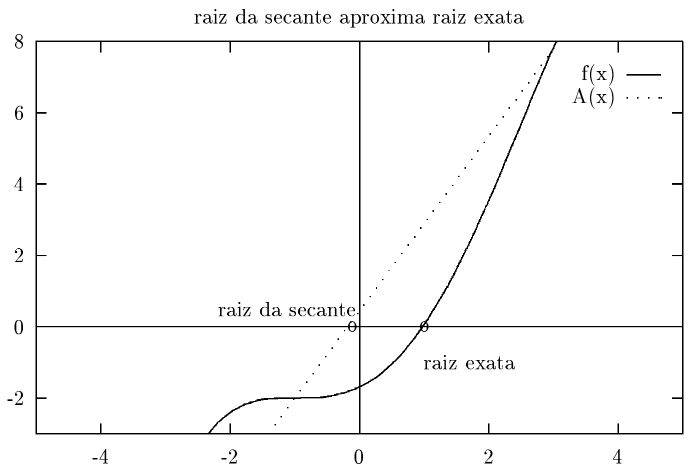

Figura 2.2: Raiz aproximada obtida com a reta secante.

métodos clássicos, porque você verá que eles podem interagir uns com os outros. Veremos, quando estudarmos `busca binária`*, como se pode tornar mais eficiente o* `método da secante`*. Finalmente, a* `varredura`*, objeto do primeiro parágrafo, será o complemento de todos os demais* métodos.

2.3.2 O método da secante

O gráfico, (fig, 2.2), ilustra o conteúdo empírico do método.

Com uma varredura, como vimos na primeira parte do capítulo, podemos determinar um intervalo em que f troque de sinal. Como suponho que f seja altamente diferenciável, logo contínua, pelo *Teorema do Valor médio de Weirstrass*, f se anula em algum ponto deste intervalo.

Isto prova que *a raiz da reta secante é uma solução aproximada do problema* $f(x) = 0$, mas pode ser uma péssima aproximação, veja a sugestão gráfica, (fig.2.2). Aqui começa busca de um *algoritmo* que nos garanta uma melhor aproximação.

Em suma:

Teorema: 1 *Método de Newton da secante.*
Se f for uma função contínua e se tiver pelo menos uma raiz num intervalo I, então existe um sub-intervalo $[a,b]$ de I tal que o zero da reta secante

$$\overline{(a,f(a)),(b,f(b))}\ ;\ x' = a - \frac{f(a)(b-a)}{f(b)-f(a)}$$

é uma solução aproximada da equação $f(x)=0$, isto é,

$$f(x) \approx 0; |x - x'| < \epsilon.$$

Dem:
Descobrimos, com uma varredura, que no intervalo $I=[a,b]$ se tem

$$sinal[f(a)] \neq sinal[f(b)],$$

suponhamos que se tenha $f(a) \leq 0 \leq f(b)$, por compatibilidade com a (fig. 2.2). O caso $f(b) > 0 = f(a)$ torna o teorema trivial com $c=a$, portanto vamos ignorar as duas possibilidades de igualdade. O Teorema do Valor médio de Weirstrass *nos garante que existe um ponto $c' \in (a,b)$ tal que $f(c')=0$. A reta que une os pontos $(a,f(a))$ e $(b,f(b))$ também corta os eixos, pelo mesmo teorema do valor médio, num ponto $c \in (a,b)$.*

Técnica da demonstração
Substituir sucessivamente os extremos do intervalo $I=[a,b]$ por novos valores, produzindo duas sucessões que convergirão para um zero de f.

Vale apenas uma das duas condições seguintes em que c é o ponto médio do intervalo $[a,b]$:

$sinal[f(c)] = sinal[f(a)]$ *ou*

$sinal[f(c)] = sinal[f(b)]$.

A que for verdadeira[3] *permite substituir o extremo do intervalo por c:*

If $sinal[f(c)] = sinal[f(a)]$ **then** $a := c$

Else $b := c$

e voltamos a iterar o método com o novo intervalo $[a,b]$:

$$[a,b] := [c,b] \text{ ou } [a,b] := [a,c].$$

Acabei de descrever[4] *"computacionalmente" o algoritmo, mas preciso descrevê-lo "matematicamente" para discutir a "convergência" e depois estabelecer a existência de uma solução aproximada*[5]*, que possa ser encontrada computacionalmente. Em Matemática escrevemos:*
Definimos tres sucessões cujos primeiros termos são :

$$c_0 = c\ ;\ a_0 = a\ ;\ b_0 = b \tag{2.3}$$

sendo c o ponto em que a reta $\overline{(a,f(a)),(b,f(b))}$ corta o eixo OX, e em seguida, iterativamente, $n > 0$ temos:

If $sinal[f(c_n)] = sinal[f(a_n)]$ **then Begin** $a_{n+1} := c_n\ ;\ b_{n+1} = b_n$ **End**

[3] Observe que não é um teste de igualdade!
[4] Nao deixe que o autor o engane, as igualdades acima não se podem produzir em Pascal.
[5] Porque *convergência* é um conceito da *Matemática*

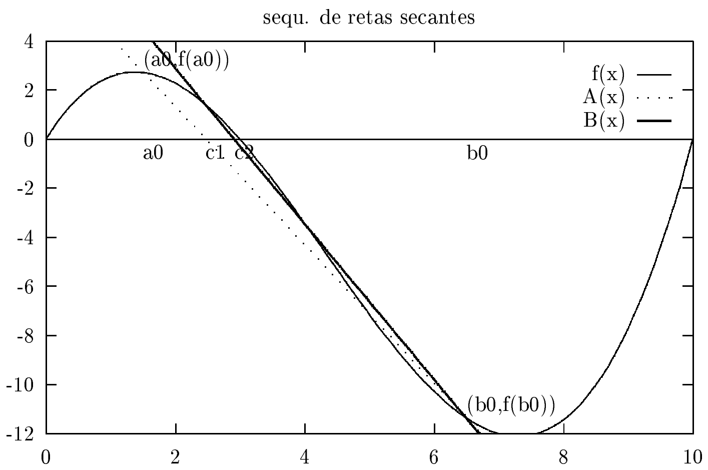

Figura 2.3: A seqüência c_n no método da secante.

Else Begin $a_{n+1} = a_n$; $b_{n+1} := c_n$ **End**

definindo duas sucessões monótonas: $(a_n)_n$ *é crescente,* $(b_n)_n$ *é decrescente, porque o novo valor é sempre intermediário:*

a sucessão a_n *é crescente:*

$$a_{n+1} := a_n \text{ ou } a_{n+1} := c_n > a_n$$

a sucessão b_n *é decrescente:*

$$b_{n+1} := b_n \text{ ou } b_{n+1} := c_n < b_n.$$

como $a_n < b_0$ *e* $b_n > a_0$ *então as duas sucessões são limitadas no mesmo sentido em que crescem, logo têm limite, quer dizer,*

$$\exists\ c \in [a,b]\ ;\ \lim_n a_n = c\ ;\ \lim_n b_n = c.$$

Veja o gráfico (fig. 2.3).

Observação: 15 *Algoritmo paralelizável*

Vou deixar-lhe um exercício refinado aqui. Veja que a curva, o gráfico de f*, poderia ser um "S" cortanto o eixo* OX *uma vez ou mais.*

Como escrevi o algoritmo, só será encontrado um dos múltiplos zero, o maior, ou o menor. Em algum lugar do **if** *descrito acima teria que haver a hipótese de que se* a_n, c_n, b_n *fosse zero isto ficasse registrado como um zero. Se isto acontecer temos um zero interno... por que?* $|a_n - b_n| \neq 0$.

Se pudessemos fazer processamento paralelo quebrariamos o intervalo inicial agora em dois e os despachariamos, cada um, para um processador para buscar os demais zeros em cada subintervalo. Desta forma achariamos todos *os zeros. Temos aqui um exemplo de aplicação para processamento paralelo, que pode ser feito mesmo com Pascal com* *aplicações recursivas*[6] *(em paralelo) da procedure de busca de raízes. TurboPascal, entretanto, não é muito bom para* recursividade. `Python`*, rodando em* `LinuX` *seria uma ótima opção, desaconselho a fazer isto em Windows.*

Pense neste detalhe. Divirta-se com um programa bonito.

Provei que as duas sucessões construídas com o algoritmo computacional são convergentes, e pelo menos uma delas converge para uma raiz, (uma delas pode ficar constante...), é o que sugere o gráfico (fig. 2.3).

Mas não nos interessam limites, *eles podem ser um ideal inatingível. Dos limites, interessa nos apenas a prova da existência. Queremos conhecer a precisão com que se obter a raiz após n iterações do algoritmo. Como o ponto c_n divide o intervalo $[a_n, b_n]$ numa razão dada, então se encontra sempre numa das duas metades do intervalo, portanto a sucessão convergente para a raiz converge com uma velocidade maior que $\frac{b-a}{2^n}$. Se igualarmos $\frac{b-a}{2^n} = \epsilon$ em que ϵ é a precisão esperada, podemos calcular o número de iterações n com esta equação.* $\boxed{q.e.d.}$

Na verdade provei mais um pouco: que o algoritmo construído na primeira parte da demonstração define duas seqüências convergentes cujo limite comum é uma raiz de f. *Quando se consegue fazer uma demonstração deste tipo se diz que* o algoritmo é convergente.

Estas divisões sucessivas que produzem uma sucessão de intervalos cujas medidas tendem a zero definindo um ponto, *é idéia típica da construção dos números reais pelo método dos intervalos encaixados, ver* `Cantor` *ou* `Dedekind` *e a* `aritmética intervalar`.

No intervalo $[a, b]$, em que se detectou a troca de sinal, pode haver várias raízes e este método encontra apenas uma delas. Refaça o gráfico (fig. 2.3), agora com mais de uma raízes entre a e b, opere o algoritmo manualmente, você verá que o método encontra apenas uma das raízes, *aproximadamente*.

Os programas `RaizFinXX` indicam o número de iterações feitas para encontrar a solução aproximada com a precisão prescrita no programa.

2.3.3 Método de Newton-Raphson

O método de `Newton-Raphson` consiste em procurar uma raiz da reta tangente como uma aproximação da raiz da função, em vez de usar a reta secante como foi feito no caso anterior.

[6] Observe que não há nenhuma vantagem, a não ser didática, em fazê-lo.

Recordemos, a equação de uma reta, conhecido o seu coeficiente angular m e sabendo que passa no ponto (a, b) é

$$y = m(x - a) + b$$

Conhecemos, (ou podemos calcular aproximadamente), o coeficiente angular de f no ponto a, então a equação acima se escreve:

$$y = f'(a)(x - a) + f(a)$$

cuja raiz será:

$$x = a - \frac{f(a)}{f'(a)}.$$

Se f for uma função localmente convexa[7], e tiver raízes, o método funciona muito bem porque, por definição de conjunto convexo, todos os pontos de uma reta que não o intercepte, ou no máximo lhe seja tangente, ficam do mesmo lado desta reta. As funções convexas se definem usando o conceito de *hipergráfico.*

Definição : 2 *Hipergráfico de uma função. O hipergráfico de uma função f é o conjunto*

$$hiperg(f) = \{(x, y) \ ; \ y > f(x) \ \}$$

Definição : 3 *Função convexa*

f é uma função convexa se $hiperg(f)$ for um conjunto convexo.

Observação: 16 *Funções convexas univariadas*

As funções convexas têm uma importância significativa e tem propriedades que vou rapidamente descrever omitindo as demonstrações que o leitor interessado pode encontrar em [4, cap. I pag 48]

$\boxed{\textbf{Teorema:}}$ *2 Caracterização das funções convexas. Se uma função f for convexa em $[a, b]$, dados dois números reais positivos s, t ; $s + t = 1$ então $f(sa + tb) \leq sf(a) + tf(b)$.*

Este teorema é a tradução algébrica de que as retas tangentes se encontram fora do $hiperg(f)$. Uma soma da forma $sa+tb$; $s, t \geq 0$; $s+t \leq 1$ se chama uma combinação linear convexa de a, b. Se $s + t = 1$ o resultado de todas as somas deste tipo é a reta que une os pontos a, b e assim o teorema diz, também, que, o *$hiperg(f)$* contem a reta $sf(a) + tf(b)$, se contiver os dois pontos extremos $(a, f(a)), (b, f(b))$.

[7] Uma função é localmente convexa se houver um intervalo $[a, b]$ sobre o qual ela seja convexa.

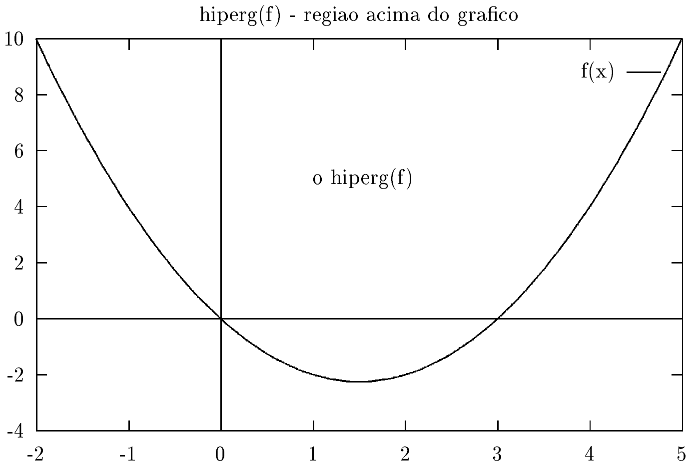

Figura 2.4: Hipergráfico é a região do plano que fica acima do gráfico de f.

Teorema: *3 Derivação das Funções convexas.*
As funções convexas são diferenciáveis en quase todos os pontos.

Observe que o "quase" tem sentido, a função módulo é convexa sendo diferenciável em quase todos os pontos porque não o é na origem, mas, é somente neste ponto que $f(x) = |x|$ não é diferenciável. É difícil uma demonstração deste teorema. Vamos nos restringir as funções convexas, f, deriváveis e vejamos o que podemos dizer de f'. Dados tres pontos sucessivos, $a < b < c$ no domínio de f, os segmentos de reta $(a, f(a)), (b, f(b)), (c, f(c))$ têm que estar contidos em hiperg(f) quaisquer que sejam a, c ; $a < b < c$. Pela convexidade, a derivada a direita em b tem que ser menor que a derivada a esquerda em b, (veja que isto vale mesmo para $f(x) = |x|$ com $b = 0$). A conclusão é:

Teorema: *4 A variação da derivada nas funções convexas. Se f for convexa então f' é crescente.*

Uma forma equivalente do teorema anterior é:

Teorema: *5 A variação da derivada nas funções convexas. Se f for convexa então f'' é positiva.*

Com freqüência se toma o último teorema como definição *de função convexa.*

Se f for convexa, $Hiperg(f)$ fica do mesmo lado de todas as retas tangentes ao gráfico de f. No gráfico (fig. 2.4), f é convexa, (seu hipergráfico é um conjunto convexo), e todas as tangentes ao gráfico de f ficam no *exterior* do conjunto $Hiperg(f)$, se o eixo OX cortar o $graf(f)$ necessariamente as raízes das tangentes se irão aproximar da raiz procurada. Vamos nos fixar em funções e raízes em pontos de concavidade, pontos em que uma função é *localmente convexa* ou *localmente côncava*. O argumento se adapta imediatamente a partir do caso convexo.

Suponhamos que $f(x_1) > 0$ e se tenha uma solução $f(a) = 0$ na vizinhança de x_1, então f decresce, logo $f'(x_1) < 0$ e a reta tangente tem coeficiente angular negativo logo se f for convexa o zero da reta tangente, x_2, é menor do que a solução da equação: $x_2 < a$, porque a tangente está fora do $hiperg(f)$. Encontramos assim um `zero aproximado`, $x_2 < a$; $f(a) = 0$. Como $x_2 < a$ podemos repetir o processo com $a := x_2$ uma vez que *necessariamente* $f(x_2) > 0$ pela convexidade.

Definimos assim, sucessivamente,

$$x_1, x_2, \cdots, x_{n-1}, x_n, \cdots \ ; \ x_{n-1} < x_n < a$$

uma sucessão crescente e limitada, então convergente

$$\lim_n x_n = c \leq a$$

Suponhamos que $c < a$, mas neste caso podemos definir mais um ponto usando o mesmo algoritmo, logo mais um ponto que também pertence a sucessão, e que é maior do que o limite c, *uma contradição*, logo

1. a sucessão não converge ou
2. $c \geq a$.

Assim, se o problema tiver solução, o limite $c = a$ é a raiz que procuravamos.

O que fizemos tem um processo dual: se $f'(x_1) > 0$ então a sucessão será decrescente, e a função côncava.

Demonstramos assim uma primeira redução do Teorema que afirma que o método de Newton-Raphson produz uma sucessão que converge para a raiz de f, se esta existir, e se a função for localmente convexa ou côncava:

$\boxed{\textbf{Teorema:}}$ **6** *Se f for convexa ou côncava num intervalo I e se tiver uma raiz neste intervalo então o método de Newton-Raphson produz uma sucessão que converge para a raiz de f.*

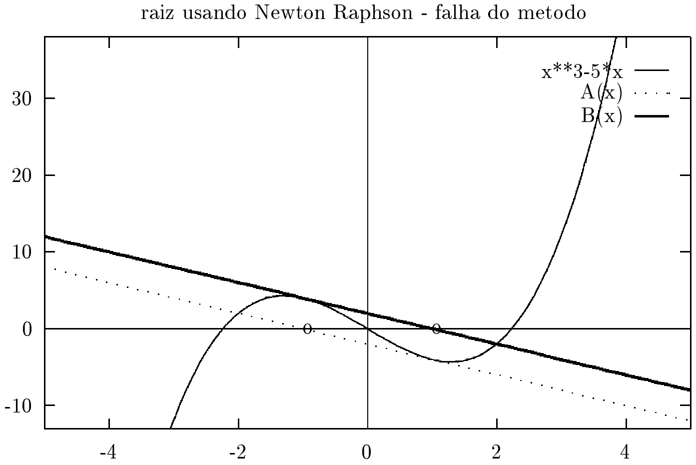

Figura 2.5: Com a reta tangente podemos obter sucessões divergentes, oscilando entre dois valores fixos.

Observação: 17 *Problemas sem solução*

O problema pode não ter solução: considere o gráfico da função convexa representada por uma parábola que não corte o `eixo dos x`. *Não se esqueça que,* a priori, *não sabemos se a função tem raízes... este é um exemplo simples de necessidade de monitoramente do programa pelo usuário.*

A (fig. 2.5) ilustra um caso de mudança de concavidade no gráfico da função produzindo uma sucessão oscilante que não converge com o presente método:

Exemplo: 24 *Função simétrica em torno de uma raiz.*

A equação da função em (fig. 2.5) é:

$$f(x) = \frac{5x}{4} - \frac{x^3}{4}.$$

e o ponto $x_1 = -1$ *que pode ser escolhido para produzir uma seqüência divergente é o ponto em que a concavidade muda. Sem muitas contas,*

$$f(1) = 1; f(-1) = -1; f'(-1) = f'(1) = \frac{1}{2}$$

conseqüentemente a raiz da tangente em $x = 1$ *será -1 e a raiz da tangente em* $x = -1$ *será 1, desta forma a sucessão das raízes das tangentes será* $1, -1, 1, -1, \cdots$*, uma sucessão divergente.*

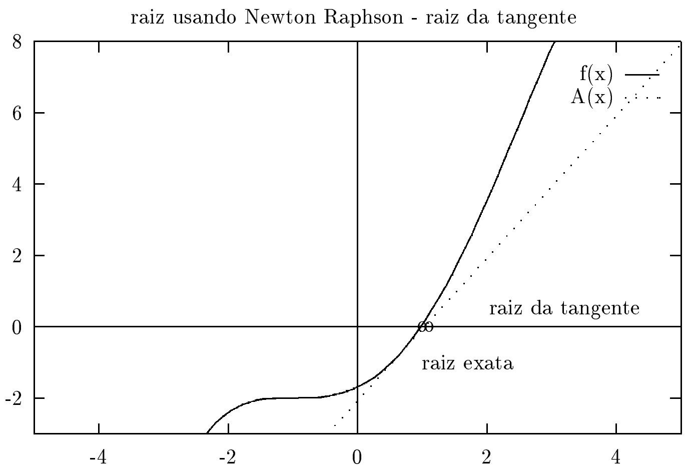

Figura 2.6: Uma função localmente convexa e uma raiz aproximada pelo método de Newton-Raphson.

O ponto central do resultado anterior é *as funções localmente convexas ou côncavas, tem o sinal da segunda derivada constante na vizinhança de quase todos os seus pontos.* Este é o quadro geral do método de Newton-Raphson.

Pela continuidade da segunda derivada[8], se f tiver raiz n'algum intervalo e na raiz, a, $f''(a) \neq 0$ então haverá um sub-intervalo em que o sinal de f'' é constante e portanto neste sub-intervalo, se $f(x) = 0$, tiver solução o método produzirá uma sucessão convergente para a raiz:

Teorema: 7 *Método de Newton-Raphson.*

Se f tiver uma raiz num intervalo no qual f'' tiver sinal constante, então o método de Newton-Raphson produz uma sucessão que converge para a raiz de $f(x) = 0$.

A (fig. 2.6) expõe a parte intuitiva do método:

A conclusão que tiramos sobre o método, inclusive frente ao exemplo correspondente à (fig. 2.5) com $f(x) = \frac{5x}{4} - \frac{x^3}{4}$ é: "o método de Newton-Raphson funciona quase sempre (o exemplo

[8]Foi uma hipótese a que aderi logo no início do capítulo.

da função simétrica em torno da raiz, representa um acidente raro mas que pode ser contornado pelo usuário uma vez que os programas devem mesmo ser monitorados).

Vou descrever, abaixo, duas variantes do método que tem uma explicação intuitiva simples. Nos exercícios você será conduzido a uma verificação destas variantes. Uma das variantes, `Newton-Raphson com peso`, funciona muito bem no caso do exemplo da função simétrica em torno da raiz:

$$f(x) = \frac{5x}{4} - \frac{x^3}{4},$$

analise por que. Entretanto nos exercícios você verá que em geral os seus resultado são mais pobres que o Newton-Raphson comum. Se o resultado é "pobre", por que apresentá-lo? A resposta para esta pergunta vem no início do capítulo quando eu disse que não me interessava pela procura de raízes em si mesma e sim por métodos de busca de dados. `Newton-Raphson com pesos` converge lentamente, mas, *busca usando pesos*, é um método que pode ser muito útil em outras circunstâncias. É o método que me interessa.

Outra conclusão que ainda se pode tirar reforçam os comentários que já fiz, o usuário tem que monitorar os programas como um instrumento potente em suas mãos. Se no exemplo 2.5 se usar $x_1 \neq -1$, o algoritmo funciona... portanto, quando o usuário encontrar uma sucessão periódica, deve mudar o ponto inicial de busca ou melhor, usar pesos.

Exemplo: 25 *Newton-Raphson com pesos.*

Consideremos a seguinte modificação do método de Newton-Raphson:

1. *Considera-se um intervalo $[a, b]$ em que f troca sinal. Para simplificar a linguagem consideremos: $f(a) > 0 > f(b)$.*

2. *Defina-se $s := |f'(a)|$; $t := |f'(b)|$; $med := (s*a+t*b)/(s+t)$*
   Se $f(a) * f(med) < 0$ então $a := med$
   Alternativamente $b := med$;

3. *Continua até que $|f(med)| < passo$.*

Este processo produz uma sucessão convergente em qualquer caso, em que, naturalmente, f tenha raiz. Se $f(x) = \frac{5x}{4} - \frac{x^3}{4}$

e $a = -b$ *então* $s = t$ *e* $med = \frac{a+b}{2} = 0$. *Se* a *e* b *não forem simétricos, mas estiverem em torno de zero, este Newton-Raphson com peso irá encontrar a raiz enquanto que o outro possivelmente não . A razão é bem simples: o método encontra pontos médios enquanto que o Newton-Raphson pode aceitar os extremos do intervalo. No caso ilustrado pela (fig. 2.5), Newton-Raphson com peso encontra logo o zero.*

2.3.4 Busca binária

`Busca binária` não é um método de determinação de raízes, especificamente. Se uma propriedade vale em forma dual nos extremos de um intervalo, em presença de continuidade, então esta propriedade irá valer em forma dual nos extremos de um dos sub-intervalos obtidos do primeiro pela divisão num ponto interior que pode ser encontrado através de sub-divisões sucessivas em pontos médios, (sub-divisão binária). Consideremos o nosso par de propriedades duais sendo

$$f(a) > 0 \; e \; f(b) < 0 \text{ no intervalo } [a, b]$$

em que f é contínua. Então a propriedade ainda vale em um sub-intervalo $[x_n, x_{n+1}]$ com, $a \leq x_n < x_{n+1} \leq b$ como conseqüência do Teorema do Valor Intermediário de Weirstrass.

Novamente, não nos interessa encontrar exatamente

$$x_n, x_{n+1},$$

aliás, "interessa" não é a palavra certa[9], é que em geral não se encontram soluções exatas. A idéia é a da construção de Cantor ou Dedekind, referida no capítulo 1 e usada implicitamente no método de Newton. Se divide o intervalo ao meio e se testa o que ocorre relativamente a propriedade no ponto central. Se este ponto coïncidir com a classe de $\underline{a}$ será escolhido como novo valor para $\underline{a}$, caso contrário, terá que coincidir com a classe de $\underline{b}$ e então será o novo valor para $\underline{b}$ e temos um novo intervalo $[a, b]$ para iterar o processo. Os valores da sucessão obtida serão sucessivamente pontos médios, donde o nome de busca binária.

[9] É como dizia raposa de Esopus, "ainda não estão maduras..." já que não podia alcançar as uvas...

Ora, isto produz uma sucessão de Cauchy porque a distância entre os membros da sucessão será menor que $\frac{b-a}{2^{n-1}}$.

Podemos usar `busca binária` em conexão com o método das secantes descrito anteriormente, ou simples junto com a propriedade dual representada pela troca de sinal de uma função contínua.

Demonstramos assim:

$\boxed{\textbf{Teorema:}}$ **8** *Raízes de funções contínuas que trocam de sinal.*

Seja $\{f(a) > 0$ *e* $f(b) < 0\}$ *no intervalo* $[a, b]$ *em que* f *é contínua então a sucessão* $(x_n)_n$ *dos pontos médios dos sub-intervalos em que a propriedade vale forma uma sucessão de Cauchy que converge para uma raiz de* f *em* $[a, b]$.

Aqui se aplica a mesma observação que fiz depois da demonstração do `Método da secante`, este método encontra uma raiz de f no intervalo $[a, b]$ mas apenas uma delas caso haja várias. Enfim estou apenas reforçando o que já disse anteriormente, os métodos que temos não funcionam sozinhos, necessitam de um monitoramento inteligente.

2.3.5 Programas utilizando os métodos descritos

Vou adotar a seguinte metodologia na discussão . Primeiro apresentarei um programa que determina raízes com razoável precisão . Depois discutirei os métodos envolvidos. Vou supor que você irá rodar o programa antes de lê-lo para ver como funciona. Neste caso verifique que no início se encontra a definição da função cujas raízes serão procuradas e troque-a por algumas equações diferentes ao rodar o programa.

O programa que apresento a seguir já atingiu um nível de sofisticação elevado, mas ainda tem imperfeições que indicarei posteriormente. Antes que você se lance à leitura de `RaizFn02` vou fazer algumas observações genéricas, e sugerir um método de leitura do programa.

Um bom programa em Pascal é uma *listagem de Rotinas*, `Proce-dures` ou `Functions`, que executam todo o trabalho e que foram construídas separadamente como programas isolados para executarem pequenas tarefas. Esta *listagem* forma o `corpo do programa`.

Comece a leitura a partir de onde se lê `Corpo do programa - Rotina Principal`. Não se importe *na primeira leitura* com o conteúdo das `Procedures e Functions`, procure entender a lógica do programa.

Depois faça uma segunda leitura do programa, neste caso leia as correspondentes `Procedures ou Functions` no momento em que elas forem sendo chamadas.

As versões deste programa que se encontram no arquivo `pas.zip` diferem desta, possivelmente. A do arquivo `pas.zip` é sempre a mais recente. Os programas RaizFinX; $X \in \{1, 2, 5, 6\}$ se encontram no `pas.zip`. A numeração dos programas foram mantidas por razões "históricas", você assim pode acompanhar o progresso do autor...

```
{  **  programa raizfin  versao 4 do programa ra\iagu zes.
   **  Analisa ra\iagu zes p/ secancia, p/ tangencia
   **  Raizfin2 joga resultados num arquivo em disco.
   **  por T Praciano-Pereira  - Matematica  -  URG  ***}

program raizfin02;

Uses Crt,printer;

Var inicio, fim, x,x1,x2, deltax : Real;
    Resposta : Char;

{** a funcao cujas raizes serao procuradas   *** }
Function f(x:Real):Real;
Begin
   f := (x*x*x +3*x*x+3*x+1)*sin(3*x);
End;

{**  calculo de derivada aproximadamente  ***}
Function difer(x,deltax:real):Real;
Begin
  difer := (f(x+deltax) - f(x));
End;

{** calculo de segunda derivada aproximadamente ***}
Function SegDifer(x:real):Real;
Begin
  SegDifer := difer(x+deltax,deltax)-difer(x,deltax);
End;

{**  detecta existencia de uma tangente horizontal ***}
Function TangenteHorizontal(x:Real):Boolean;
Var teste1, teste2 : Boolean;
Begin
 teste1:= (difer(x,deltax)*difer(x+deltax,deltax) < 0);
 teste2:=(difer(x,deltax)*difer(x+deltax,deltax)<deltax);
 TangenteHorizontal := teste1  or teste2 ;
End;
```

```
{1)**** Determinacao refinada de Raiz por secancia
Duas Procedures: NewtonRaphson  e  RaizSecante(varredura)
Newton-Raphson/ metodo  media aritmetica ponderada usando
como pesos as derivadas no ponto ****}
Procedure NewtonRaphson(x,deltax : Real);
Var med, ant, post,peso1,peso2,r : Real;
Begin
 med:=x;
 While Abs(f(med)) > deltax Do
  Begin ant:=med; peso1:=Abs(difer(ant,deltax)/deltax);
   post:=med - f(med)/(difer(med,deltax)/deltax);
   peso2:=Abs(difer(post,deltax)/deltax);
   med:= (1/(peso1+peso2))*(peso1*ant + peso2*post);
  End;
 WriteLn('raiz NR em  f(',med:2:9,') =  ', f(med):2:6);
End;

{**** Determinacao refinada de Raiz por secancia ****}
Procedure RaizSecante(x, deltax : Real);
Begin
 While f(x)*f(x+deltax) > 0 Do x:= x+deltax;
 WriteLn('raiz em Sec  f(',x:2:4,') =  ', f(x):2:4);
End;

{*** Determinacao refinada de raizes por tangencia ****}
Procedure VerificaRaizTangente(inicio,fim, deltax :Real);
Var x: Real;
Begin
  x := inicio;
  While (x <= fim) and (Abs(f(x)) < deltax) Do
x := x+deltax;
  If x > inicio Then
  Begin
    x:= 0.5*(inicio+fim);
    WriteLn('raiz tang. em f(',x:2:4,')= ', f(x):2:4);
  End;
End;

{*** detecta raiz por secancia: troca de sinal ****}
Procedure VerificaRaizSecante;
fat1, fat2 : Extended;
Begin
 fat1 := f(x)*f(x+deltax);
 fat2 := difer(x,deltax)*difer(x+deltax,deltax);
 If ( fat1 <= 0) and (fat2 > deltax)  Then
   Begin
    fat1 := difer(x,deltax)/deltax;
    fat2 := difer(x+deltax,deltax)/deltax;
   If Abs(fat1* fat2) > 1
       Then NewtonRaphson(x,deltax/1000)
        Else RaizSecante(x,deltax/1000);
End;
{**** Corpo do programa -  Rotina principal  *****}
Begin
Writeln('Inicio e fim do intervalo de pesquisa');
Write('Inicio: ');Readln(inicio);
```

```
Write('Fim: ');Readln(fim);
WriteLn('Precisao  deltax=  ');ReadLn(deltax);
Repeat
x := inicio; ClrScr;
  Repeat
    VerificaRaizSecante;  {** Raiz c/ derivada <> 0 **}
    If TangenteHorizontal(x) Then
    Begin
      {***  Analise fina   ***}
      VerificaRaizTangente(x-deltax/20,x+deltax,deltax/100);
    End;
    x := x + deltax;
  Until x > fim;
WriteLn;
WriteLn('Deseja refinar a pesquisa <s , n>');
Readln(Resposta);
If ( Resposta = 's' ) then
  Begin
    deltax := deltax/10;
    Writeln(' Anote os dados, tela vai ser limpa..');
    Writeln(' ... aperte uma tecla para continuar!');
  End;
Until Resposta = 'n'
End.
```

O exercício que segue é experimental, com ele você vai descobrir um defeito do Método de Newton-Raphson.

Exercício: 18 *Variações com Newton-Raphson.*

1. *No programa RaizFin01, substitua a rotina Newton-Raphson pela seguinte:*

```
Procedure NewtonRaphson(inicio,fim,deltax : Extended);
Var s,t,med,raiz,passo : Extended; contador : Byte;
Begin
     passo := deltax/1000;
     s:= Abs(difer(inicio,passo)/passo);
     t:= Abs(difer(fim,passo)/passo);
     med := (s*inicio + t*fim)/(s+t);contador := 1;
     If med > fim then med :=0.5*(inicio+fim);
     raiz := inicio - f(inicio)/( differ(inicio, passo)/passo);
     If f(raiz)*f(inicio) > 0 Then  inicio := raiz
        Else  fim := raiz;
     While  ( (Abs(f(med)) > passo) and (Abs(f(raiz)) > passo )
and (contador < 5) )
        Do
        Begin
          If f(med)*f(inicio) > 0 Then  inicio := med
             Else fim :=  med;
          s:= Abs(difer(inicio,passo)/passo);
          t:= Abs(difer(fim,passo)/passo);
          med := (s*inicio + t*fim)/(s+t);contador := 1;
          If med > fim then med :=0.5*(inicio+fim);
          WriteLn('pelo valor medio: f(',med:10:10,') =  ');
          Write( f(med):10:10);
          WriteLn('N-R puro f(',raiz:10:10,') =  ');
          Write( f(raiz):10:10);
          Inc(contador);
        End;
```

```
Mensagem := 'raiz NR em ';
SaidaDeDados(mensagem,med);
```

Rode o programa com $f(x) = ArcTan(x)$*. O resultado será uma listagem com a qual você poderá comparar a eficiência de* `NR com peso` *ou* `NR ''puro"`*. Qual o resultado ? Faça um gráfico e faça uma análise geométrica do que pode ter ocorrido.*

2. *Repita a questão anterior com* $f(x) = \frac{2x}{(x^2+1)^2}$.

3. *Mostre que é possível com qualquer das duas funções acima encontrar um paralelogramo em que o Método de Newton-Raphson produz um* loop infinito*.*

4. *Implemente numa rotina NR o seguinte algoritmo:*

   (a) *Considera-se um intervalo* $[a, b]$ *em que* f *troca sinal. Para simplificar a linguagem consideremos:* $f(a) > 0 > f(b)$.

   (b) *Defina-se* $s := |f'(a)|$; $t := |f'(b)|$; $med := (s * a + t * b)/(s + t)$ *Se* $f(a) * f(med) < 0$ *então* $a := med$ *Alternativamente* $b := med$;

   (c) *Continua até que* $|f(med)| < passo$.

   fazendo que sejam escritos os dados locais dentro da rotina para comparar com NR padrão .

$\boxed{\textit{Limitações do método de Newton-Raphson.}}$

A saída de dados correspondentes às duas questões acima é respectivamente:

```
1) Raizes de   f := ArcTan(x)
No intervalo [-5,5] passo=  0.1
======================================


2) Raizes de   f := 2*x/( x^2+1)^2 )
No intervalo [-5,5] passo=  0.1
===============================
```

Saiu apenas o rótulo da saída de dados, porque ambos os casos o programa não conseguiu encontrar raízes. O programa poderia ter caido num *loop infinito*, não é difícil construir um paralelogramo com tangentes de modo que a sucessão obtidas com as raízes das retas tangentes sempre produzam as duas tangentes paralelas. Conseqüentemente, a sucessão das `raízes ''aproximadas"`, neste caso, seria uma sucessão oscilante com dois valores diferentes apenas, portanto divergente.

A solução para evitar esta cilada consiste em incluir um "contador" na procedure NewtonRaphson com um limite de iterações permitidas. Veja a rotina NR- modificada nos exercícios acima em que esta técnica está sendo usada. Se o número de iterações ultrapassar este teto, uma mensagem deve ser emitida com as seguintes informações:

- há, possivelmente, uma raiz no intervalo;
- o número de iterações permitida foi ultrapassado.

Esta mensagem está de acordo com a afirmação que venho fazendo seguidamente, sugere ao usuário uma intervenção para alterar alguns parâmetros que alterem o comportamento do programa, como *troca dos extremos do intervalo*, *tamanho do passo*, uso de *pesos no cálculo de médias*.

Os programas Raizfin5/6 que você pode encontrar no disco, usam uma técnica de cálculo de derivadas aproximadamente que não foi discutida neste livro, "convolução".

Determinação de raízes aproximadas: sumário.

Os passos da análise de busca de raízes que segui foram os seguintes:

1. Procura `grossa`, isto é, com um passo grande, de pontos em que:

   (a) há trocas de sinal de $f; f(a) * f(a+\Delta) < 0$

   (b) $f(a)$ é pequeno.

   (c) há trocas de sinal de f' e neste caso com verificação simultânea de que $|f(a)|$ é pequeno.

   Estes dois itens tem o objetivo de classificar as raizes nas duas grandes classes: `raiz por secância`, quando $f'(a) \neq 0$, ou `raiz por tangência` quando $f'(a) = 0$.

2. A verificação de que $f(a)$ *seja pequeno* é complicada porque ela envolve uma relação de ordem de grandeza entre Δf e Δx. Uma alternativa consiste em analisar se $f(a) * f(a+\Delta)$ é pequeno comparado com $f(a)$. Se $f(a)$ for uma raiz aproximada então a ordem de grandeza do produto $f(a) * f(a+\Delta)$ é menor do que a de $f(a)$.

3. Varredura fina:
   detectada uma raiz, "uma rotina apropriada" volta a varrer um pequeno sub-intervalo com um passo muito pequeno para determinar, com alta precisão , o ponto $\underline{a}$, tal que $f(a)$ é aproximadamente zero. A "rotina apropriada" está ligada a uma das duas classes de raizes: por secância ou por tangência.

4. Uma rotina para filtrar raízes aproximadas repetidas que consiste em produzir um salto de um passo sempre que se encontrou uma raiz, na pesquisa grossa.

5. Incluir contadores dentro das rotinas de refinamento para prevenir contra *refinamentos indefinidos.*

6. Sugerir ao usuário que existe raiz incluindo na sugestão aonde e de que tipo.

Exercício: 19 *Raízes de funções e outras buscas*

1. *Use um programa, com o método de Newton-Raphson, para encontrar todas as raízes de f e de f′ no intervalo [-7,7], para as seguintes equações:*

$$
\begin{array}{lll}
f(x) = x^2\sin(x); & f(x) = x^2\sin(2x); & f(x) = x^2\sin(x^2);\\
f(x) = x^2\cos(x); & f(x) = x^2\cos(2x); & f(x) = x^2\cos(x^2);\\
f(x) = \frac{x^2\sin(x)}{x^4+1}; & f(x) = \frac{x\sin(x^2)}{x^4+1}; & f(x) = \frac{x^2\cos(x)}{x^4+1};
\end{array}
$$

2. *Use os dados da questão anterior para traçar os gráficos das funções alí definidas.*

3. *Use um programa para encontrar todas as raízes de f, f' e f'' no intervalo [-7,7], para as equações abaixo. As derivadas devem ser calculadas por quociente de diferenças.*

$$
\begin{array}{lll}
f(x) = x^4 - 1; & f(x) = x^6 + 3x^2 + 1; & f(x) = \sin^2(x) + 3\sin(x) + 1;\\
f(x) = \frac{\exp(x)}{x^2+1}; & f(x) = \frac{\exp(x)}{0.2cos(2x)+1}; & f(x) = \frac{\exp(x)}{x};\\
f(x) = \frac{\exp(x)}{x+\sin(x)}; & f(x) = \frac{\exp(x)}{2x+x\cos(x^2)}; & f(x) = \frac{\exp(x)(x^2+1)}{2x};
\end{array}
$$

4. *Altere o programa usado anteriormente substituindo os valores de f', f'' por valores médios destes valores calculados com passo "ultrafino" nas vizinhanças de cada nó.*

5. *Refaça o programa anterior incluindo as derivadas formais e compare os resultados.*

6. *Modifique* `RaizFin05` *incluindo nele uma rotina gráfica para mostrar o gráfico da função cujas raízes se procuram nos exercícios anteriores.*

7. *Modifique algum programa da família* `RaizFin` *para que seja feito o gráfico da secante na determinação da raiz pelo método de Newton, (use algumas das funções do primeiro exercício).*

8. *Modifique algum programa da família* `RaizFin` *para que seja feito o gráfico da tangente na determinação da raiz pelo método de Newton-Raphson (use algumas das funções do primeiro exercício).*

9. *Estenda, diferenciavelmente, com um polinômio de grau 5 a função* $f(x) = x\cos(x)$ *fora do intervalo* $[-\pi, \pi]$. *Ver* Polinômio de Taylor *num livro de Cálculo.*

10. *Estenda com funções constantes a função* $f(x) = x\cos(x)$ *fora do intervalo determinado pelos seus extremos locais mais próximos de zero. Verifique se esta extensão tem derivadas contínuas e até que ordem.*

11. *Encontre solução aproximadas da equação* $x^4 - 2x^2 - 3xsen(x) = 1$ *no intervalo* $[-\pi, \pi]$.

12. *Modifique o programa* `Raizes02` *para calcular* $\sqrt{2}$ *com uma precisão de 8 casas decimais usando* `busca binária`.

Capítulo 3

Aplicações: modelagem computacional de processos

Usando o computador para discutir hipóteses

Introdução

Que é *modelagem de processos*? Um experimento para recriar um determinado fenômeno num ambiente artificial? é este o objetivo deste capítulo descrever um pouco como se faz isto. Advirto o leitor de que há livros inteiros dedicados ao assunto e este capítulo é apenas uma introdução. O fato de haver extensa literatura sobre um assunto, não me impede de escrever um capítulo introdutório sobre o mesmo. Era preciso terminar mostrando onde Pascal pode ser usado como uma ferramenta científica.

Vou mostrar, apoiado na idéia de comportamento assintótico do capítulo 1, como se podem formular hipóteses. Este capítulo é uma coleção de exemplos de modelos matemáticos e de como construí-los.

A modelagem tem dois objetivos particularmente importantes, "poupar recursos naturais" "garantir condições de segurança no estudo de objetos de risco".

Ao final do capítulo vou descrever dois problemas de larga envergadura e incluir bibliografia onde eles podem ser melhor analisados se o seu interesse por eles for excitado.

3.1 Alguns exemplos básicos de modelagem

Vou descrever duas classes de experimentos que seguem planejamentos distintos. Uma desses exemplos vem da Matemática, mas o leitor deve tentar abstrair este fato e tentar nele reconhecer uma outra situação em que se poderia aplicar o método, é o *teste de hipótese*. No segundo caso *se procura a hipótese* e o método consiste numa busca exaustiva para identificar uma equação, o caso estudado é da Física, mas novamente o leitor deve procurar identificar situações semelhantes onde a metodologia se aplique.

3.1.1 Quando se quer testar uma hipótese

Então, para começar, se tem uma hipótese.

Por alguma razão se sabe que algumas equações governam um determinado fenômeno e se procura encontrar *coeficientes* apropriados para ajustar esta classe de equações ao fenômeno que interessa. Se conhecem alguns efeitos do fenômeno e se a modelagem tiver sucesso estes efeitos podem vir a ser repetidos, confirmando a equação que se tinha. Neste caso uma mudança secundária de parâmetros pode produzir uma modelagem do fenômeno em outras circunstâncias mostrando que há boas probabilidades de que a modelagem descreva o fenômeno.

Nesta linha de idéias o projeto se descreve pelas seguintes etapas:

<u>Planejamento do Experimento</u>

1. **Equação**. Se parte de uma hipótese sobre uma equação ou um sistema de equações que governam um acontecimento.

2. **Tradução computacional**. Se descreve computacionalmente a equação, em Pascal isto tem que ser feito de forma numérica o que nos obriga a lembrar os problemas com a segunda derivada descritos no capítulo 1. (Há linguagens de programação em que a derivação se encontra implementada em forma algorítmica, Maple é um exemplo. Mas uma equação não precisa, necessariamente, ter derivadas segundas...)

3. **Saída de dados**.

   (a) **um módulo de interpretação gráfica** das funções envolvidas é importante. Um gráfico nada mais é que uma listagem de valores colocados em forma visual. Algumas vezes um gráfico ajuda a corrigir hipóteses.

   (b) **listagem numérica**, é preciso manter uma análise numérica dos dados também. Pequenas oscilações podem ser invisíveis num gráfico mas perfeitamente perceptíveis numa listagem numérica. A própria quantidade de zero depois do “ponto” decimal pode evidenciar a grandeza relativa de duas variáveis.

4. **Teste da hipótese** há duas maneiras muito usadas de verificar os resultados:

(a) probabilística: Se gera uma grande quantidade de modelos e se calcula a *probabilidade discreta* de um determinado *evento*.

(b) lógica: Se prova formalmente uma hipótese encontrada.

Aqui desejo sublinhar o caso probabilístico.

Uma observação se faz necessária, a enumeração que fiz acima, uma *departamentalização* da idéia, nada mais é que uma forma, um estilo do discurso. Se você quiser, pode considerar as etapas 3 e 4 como uma única, fique claro.

Exemplo: 26 *Determinação estocástica duma integral.*

Neste exemplo vou tratar da verificação probabilística *da hipótese.*

Quero calcular $\int\limits_a^b f(x)dx$ *e parto da hipótese de a função f tem uma integral que é difícil, longa de ser desenvolvida, ou simplesmente desconhecida*[1]. *As somas de Riemann*[2] *produzem boas aproximações para as integrais, se elas existirem: esta é a equação que vai ser testada.*

- *A soma de Riemann representa neste exemplo a (etapa ,1) do planejamento do experimento acima.*
- *Um programa como* `Riemann.pas` *representa sua representação computacional (etapa , 2)*
- *A saida de dados, (etapa 3b) vai ser o número que representa aproximadamente o valor da integral* $\int\limits_a^b f(x)dx$.
- *Podemos agora inserir o programa dentro dum laço para gerar sucessivos valores da Soma de Riemann é isto que faz o programa (exemplo , 18), página 73, no capítulo 1, estamos na (etapa, 4). Naquele exemplo não fiz nenhum gráfico, ele nos seria inútil. Saltamos direto para o o levantamento estatístico de dados e teste probabilístico da hipótese. O exemplo do cálculo de integrais estocásticas, fim do capítulo 1, completa este exemplo.*

[1] Observe, de passagem, que programas de Computação Algébrica como Maple ou Derive calculam formalmente um grande número de integrais e mesmo o fazem numericamente.

[2] Relembro o que já disse, somas de Riemann não são o método mais efetivo para calcular integrais computacionalmente.

Se forem verdadeiras as hipóteses que formulei sobre a dificuldade ou impossibilidade de calcular $\int_a^b f(x)dx$*, o método aqui proposto se torna uma alternativa válida para decidir sobre a integrabilidade de uma função e assim* formular uma hipótese. *Nunca é demasiado insistir no fato de que uma amostragem não pode demonstrar a integrabilidade. Sob este aspecto o exemplo cai na outra classe.*

Chamo sua atenção que o programa referido acima representa apenas um começo de construção neste sentido, tome um livro de estatística e veja como completá-lo para produzir uma boa análise estatística automática. Este assunto, sozinho, já poderia compor um livro.

3.1.2 Quando se quer encontrar uma hipótese

O que estou chamando de hipótese quer dizer uma equação.

Ao longo da história da ciências naturais, da Física, se foram constituindo equações que governariam os fenômenos. É difícil conceber o esforço mental que nossos antepassados fizeram para concluir por certas equações diferenciais como modelos. Hoje temos meios rápidos para produzir este modelos, ou equações. Uma *equação* tem uma curva que a representa graficamente, de acordo com a nossa hipótese 1. Se encontrarmos a curva, temos a equação. Isto pode requerer mais alguns módulos de trabalho formal e não computacional.

<u>Planejamento do Experimento</u>.

No exemplo que vou descrever usarei a palavra "variável" num sentido comum entre os que aplicam matemática. É o sentido que lhe dão os estatísticos, e que para os matemáticos significa "função". Ou seja, no texto que segue, *variável* significa *função.*

1. Determinação das variáveis a serem medidas. Uma equação depende de variáveis: tempo, temperatura, elasticidade etc... Algumas vezes existe *correlação* entre elas como função de uma delas, chamada *variável independente*, as outras *dependem* desta.

2. Escolha da variável independente. Se escolhe uma variável para

ser a *independente* e se produzem as curvas das outras relativamente a esta.

3. Verificação de correlação. O que os estatísticos chamam de correlação é uma coleção de fatores multiplicativos entre duas ou mais curvas definindo uma combinação linear entre elas, se houver mais de duas curvas. Suponha que as variáveis independentes sejam: Y_1, Y_2, dependentes do tempo t. Se houver *correlação* , então existe uma constante $\underline{c}$ tal que $Y_1(t) = \underline{c}Y_2(t)$ sendo $\underline{c}$ o fator de correlação. Claro que podem haver vários fatores de correlação entre distintas variáveis. Não havendo uma correlação vísivel, se retoma o experimento com outra variável considerada *independente*, descoberta alguma correlação, se descobriu uma pista para o modelo do fenômeno. Observe que a existência de *correlação* reduz o número de variáveis[3].

Vou estudar um exemplo simples, um caso de duas variáveis não correlatas entre si, mas dependentes do tempo.

Exemplo: 27 *Interferência mecânica. Ver [5, Vol. III - Quantum Mechanics]. Quero determinar a superposição da energia contida nas ondas mecânicas produzidas por uma determinada fonte, sonora, por exemplo.*

Aparato experimental*:*

1. *uma fonte sonora;*
2. *uma parede de material absorvente com dois buracos, A, B;*
3. *um detector colocado num suporte móvel ao longo de uma linha paralela com a que passa no centro dos buracos.*

Uma das variáveis é o impulso que passa pelo buraco A, outra é impulso que passa pelo buraco B, e finalmente a terceira é soma dos impulsos que passam pelos dois buracos. Melhor dizendo, as variáveis são a intensidade destes impulsos. O detector digitaliza a intensidade dos impulsos sonoros recebidos enquanto se movimenta e um programa conta os impulsos recebidos:

1. *Com o buraco A fechado, $A(x)$.*
2. *Com o buraco B fechado, $B(x)$.*

[3]Em Álgebra Linear *correlação* se chama de *dependência linear* e o número de variáveis é a dimensão do problema.

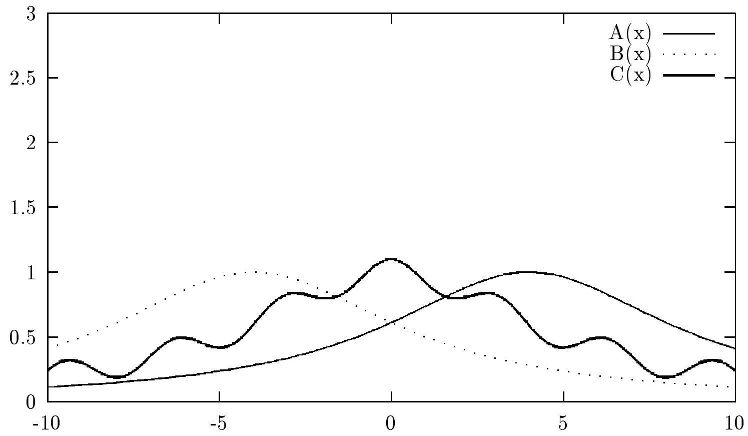

Figura 3.1: Interferência de ondas sonoras. A figura é apenas uma ilustração.

3. *Com ambos os buracos abertos, C(x).*

finalmente o programa traça tres curvas:

A soma dos impulsos, obtida quando os dois buracos estiverem destampados, *é uma curva com máximos e mínimos. Nos pontos de máximo as ondas se encontram em* ressonância *ou ainda se diz que elas se encontram em* fase. *Neste caso as variáveis A e B, não são correlatas, visivelmente, uma é uma translação da outra, nem entre si, nem com a soma, (superposição, como dizem os físicos),$C = A+B$. O resultado é "desconcertante" em primeira instância. Parecia óbvio que com os dois buracos destampados haveria uma soma da energia que emana das duas fontes. O pesquisador foi conduzido a procurar uma variável que ainda não tivesse sido considerada e descobriu a* fase, *ou seja a onda sonora tem* intensidade *e* fase *e existe um objeto matemático que tem duas coordenadas:* os números complexos, *e assim se chegou a uma hipótese:* as ondas se descrevem com números complexos e o gráfico (fig. 3.1) traz apenas a intensidade não podendo apresentar a soma da energia emitida pelas fontes A, B.

Modernizei o experimento que pode ser encontrado em [5, local citado] e onde o leitor poderá encontrar a descrição de *outros fenômenos de interferência, eletromagnéticos, por exemplo. É claro que "cheguei" a uma hipótese já conhecida o que justifica que foi necessário apenas meia página de experimento.*

Observação: 18 *Soma ou Superposicao de ondas*

O gráfico da energia obtida com a soma dos sinais não corresponde a soma das funções A e B por uma razão simples e que é interessante de sublinhar aqui: A melhor forma de modelar ondas é com variáveis complexas porque elas guardam duas informações: módulo=*intensidade* e argumento = *fase, enquanto que as variáveis reais guardam apenas a informação sobre o módulo.*

A conclusão sobre a importância do uso das variáveis complexas é a grande descoberta de hipótese que se pode salientar neste exemplo.

Vejamos outros experimentos que podem ser classificados como de *busca da hipótese*.

Exemplo: 28 *Cardiogramas e diagnóstico automático.*

A idéia básica é seguinte: uma "máquina" em funcionamento normal tem um "espectro" com um determinado modelo, naturalmente obtido experimentalmente. É o caso de cardiogramas *que, em princípio, um médico analisando a olho nú o consegue perceber anomalias no funcionamento do músculo cardíaco.*

Um banco de cardiogramas de pessoas sadias *poderia servir de* banco de dados *comparativo para diagnósticos automáticos.*

Aqui poderiamos ir ao ponto mais sofisticado designado por comparação de padrões*, ou ficar com uma metodologia mais simplificada se consistindo de uma digitalização dos cardiogramas e sua correspondente comparação numérica.*

Exemplo: 29 *Análise de vibrações mecânicas.*

*O exemplo dos cardiogramas nos sugere um outro exemplo simples e muito usado na indústria. Uma máquina mecânica, que tenha por base de funcionamento eixos em rotação, produz vibrações. Porém, quando nova, (*a comparação aqui se dá com o cardiograma da pessoa sadia*), tem um espectro de vibrações que com o tempo se altera, na medida em que os mancais que suportam os eixos, ou os rolamentos, se estragam.*

Sensores colocados em diversos pontos da máquina captam as vibrações e um gráfico pode ser feito tendo-se como variável independente a velocidade angular de rotação do eixo. As vibrações são função desta velocidade. A análise da intensidade destas vibrações, o estudo do espectro das frequências contidas na vibração, determina quando a máquina deve ser parada para manutenção antes que entre numa fase crítica que produza uma ruptura.

Exemplo: 30 *Um chofer e as vibrações do seu carro.*

Um chofer que envelheça com seu carro consegue escutar as vibrações diferentes *em diversos pontos do veículo, em função da velocidade do carro, (que é correlata com a velocidade angular do motor relativamente a cada uma das marchas)*

Uma determinada vibração, a uma velocidade específica, lhe diz que os rolamentos da roda chegaram ao fim da vida e que é preciso trocá-los. Outra vibração lhe indica que alguma peça da lataria se encontra frouxa ou quebrada, (neste caso a correlação pode ser com os buracos da ruas de onde o IPVA foi desviado).

3.2 Uma metodologia geral?

Não existe uma *metodologia geral*, como também não existe um programa genérico que sirva para tudo e nem sequer a melhor linguagem de programação com objetivos gerais para resolver qualquer problema. Pelo menos não a conheço, *mas tem gente tentando descobrir este sistema universal...*

Mas as duas classes de exemplos que apresentados na seção anterior são bem genéricos e mostram um caminho inicial para uma vasta classe de problemas:

1. testar uma equação conhecida.
2. procurar funções e suas correlações com o objetivo:
   (a) para compará-las com um modelo existente.
   (b) para estabelecer um modelo.

Se desenvolveram, junto com a crescente estruturação dos dados e com a modularidade das linguagens de computação, duas técnicas de programação chamadas:

1. **top-down design**: *planejamento de cima para baixo*, significando planejar a solução "algorítmica" de um problema a partir das idéias mais gerais e caminhando para baixo, depois, na especificação dos detalhes: os pequenos módulos.
2. **down-top design**: *planejamento de baixo para cima*, significando o inverso da outra, primeiro se busca a identificação das pequenas rotinas internas que o problema contém e a partir delas se vai gerando o planejamento geral.

3. **top-down-top**: A prática nos ensina que um ponto médio entre as duas técnicas muitas vezes é o método aconselhável: se inicia em geral com o *planejamento de cima para baixo* especificando os módulos do processo. Depois uma análise mais acurada destes módulos o modulariza ainda mais ou algumas vezes elimina módulos se produzindo uma crítica da análise inicial.

Neste contexto exigem equipamentos, computadores velozes, com grande capacidade de memória para manipular a grande quantidade de dados. Pascal é uma linguagem de programação limitada para este efeito.

3.3 Outros exemplos de modelagem

Vou descrever alguns experimentos computacionais descritos na literatura científica. A esperança é de que esta descrição possa ser útil a quem tenha algum equipamento onde possa gerar experimentos semelhantes. Vou citar algumas fontes onde outros experimentos possam ser analisados e possivelmente reproduzidos. Alguns são bem simples e se encontram inteiramente ao seu alcance. Outros se encontram aqui para indicar desenvolvimentos futuros ou pelo menos o um pouco do que existe.

3.3.1 Exemplos mais simples

Aceleração da gravidade, com mouse

Se você tiver um *ratinho*[4] suficientemente livre de atrito e se o jogar num plano inclinado para cima, ele voltaria a cair. Os sinais podem ser recuperados e o gráfico da curva traçada pelo *ratinho* pode ter seu gráfico feito.

A segunda lei de Newton diz que a única força atuando sobre o *ratinho* é da gravidade, uma vez que por hipótese não há atrito. Então o movimento não pode ser retilíneo, quer dizer que a velocidade muda. A velocidade sendo a derivada da curva da distância terá por equação $v = gt$ portanto a distância deve ter por equação:$s = \frac{1}{2}gt^2$, uma parábola. Este deve ser o gráfico da curva descrita pelo mouse, se você conseguir gerar o experimento.

[4] *Ratinho* é nome brasileiro do mouse...

Se tentar, não se esqueça de, inicialmente, reduzir a sensibilidade do *ratinho* para aumentar a amplitude da curva.

A única vantagem do *ratinho* é que você játem os dados da curva digitalizados em seu computador.

Correlação entre temperatura e pressão

Se você tiver um aparelho para medir pressão e outro para medir temperatura podendo digitalizar os dados e cedendo o impulso para um computador, será possível fazer o gráfico da evolução da temperatura e da pressão simultâneamente de um gás submetido a uma fonte de calor e encerrado num ambiente fechado. Será possível verificar que o formato do depósito em que está encerrado o gás, influencia na relação evolutiva entre temperatura e pressão : *condição de fronteira*

Será possível encontrar a função de correlação entre estas variáveis calculando o quociente entre elas:

$$c = \frac{P(t)}{T(t)}$$

considerando ambas como função do tempo. Gráficos sugerem algumas vezes uma correlação : quando houver extremos ou inflexões do mesmo tipo em dois gráficos nos mesmos pontos há uma forte razão de suspeitar de correlação. O teste consiste em calcular a fração acima, se ela ficar próxima de uma constante se encontrou o fator de correlação.

Atração da gravidade em economia

Será possível estabelecer uma lei que descreva a atração dos consumidores em função :

1. da área ocupada por uma loja;
2. da distância média da loja e o consumidor em seu trajeto casa-trabalho.

O resultado seria uma lei semelhante a da gravitação de N corpos com massas distintas?

Dados necessários para uma tal pesquisa:

1. distribuição de freqüência de fregueses a diversas lojas e "shopping centers".
2. valor médio das distâncias desses centros de compra aos clientes, por cliente no seu caminho casa-trabalho. Este valor médio é uma integral de linha.
3. área ocupada pelos "shopping centers".
4. Relação de itens atrativos para uma determinada freguesia.

Uma outra alternativa de análise poderia ser a substituição da *área ocupada pela loja* por *número de produtos em exposição*. Uma análise comparativa dos dois estudos pode sugerir aos comerciantes que é melhor ter mais espaço para atrair fregueses do que diversidade de produtos...

Um outro estudo pode responder a pergunta: "A propaganda pode alterar o efeito da gravidade?" Tem a propaganda algum efeito.

Correlação entre fenômenos que dependem do tempo

Alguns fenômenos que dependem do tempo podem ser estudados com o estabelecimento de uma correlação entre eles.

1. Tempo de percurso casa-trabalho de carro.
2. Tempo de percurso casa-trabalho em transporte coletivo.
3. Tempo de percurso casa-trabalho em bicicleta.

Paralelamente se pode estudar cada um desses percursos frente as variáveis:

1. Custo de cada um destes percursos.
2. Segurança média relativamente a acidentes.

O resultado pode ser interessante do ponto de vista ecológico e de planejamento urbano, certamente não será nada interessante do ponto de vista do capitalismo selvagem.

3.3.2 Dois exemplos bem mais complicados

Vou descrever aqui, para finalizar este capítulo, dois exemplos que mostram a importância e o alcance da modelagem apesar de que eles fiquem consideravelmente fora do alcance do conteúdo deste livro. O objetivo é apenas o de mostrar o que pode se encontrar um pouco além do horizonte.

Circulação sanguínea

A círculação sanguínea oferece duas grandes dificuldades:

1. O meio natural em que ela pode ser estudada é em animais vivos e hoje há um consenso predominante sobre o respeito à natureza que nos impede de maltratar ou sacrificar vidas animais.
2. A medida que o indivíduo envelhece a elasticidade dos seus tecidos diminue junto com aumentar o volume de entulhos nos vazos sanguíneos criando perturbações ao fluxo normal do sangue.

O segundo aspecto dos dois discutidos acima é simplesmente o mais dramático: quando o indivíduo se torna *interessante* pela quantidade de eventos críticos, é exatamente quando ele deve menos ser exposto a experiências porque precisa ser amparado com soluções.

Nos encontramos assim no centro de uma matéria importantíssima do ponto de vista de modelagem e complicada porque envolve Física, Matemática, Química, Biologia e Computação. Nos Estados Unidos, no Courant Institute um grupo trabalho no assunto, mas no Brasil, na UFRGS, Departamento de Matemática Aplicada, outro grupo também trabalha nesta direção.

Processamento paralelo.

Um supercomputador Cray, funcionando em dos centros de pesquisa da NASA, Goddard Space Flight Center, analisa as informações sobre o *tempo* que lhe chegam de sensores espalhados pelo planeta, no ar, no mar e em terra. Um outro computador constituído com uma arquitetura de multiprocessamento, (512 processadores) passou também a executar um algoritmo semelhante para as análises de informações sobre o *tempo*.

O desempenho do computador com 512 processadores se revela mais eficiente, como era esperado, mas também evidencia a existência de um problema que vem ocupando os estudiosos de processamento paralelo, *o fluxo de dados*, ou, exposto de forma mais intuitiva, um *problema de tráfico...*

Estes dois assuntos foram tirados de dois números da revista *siam news* da SIAM (Society for Industrial and Applied Mathematics). Esta revista pode ser lida livremente na Internet no endereço http://www.siam.org. Você pode ler mais a respeito e inclusive obter informações bibligráficas sobre estes assuntos ou outros mais no endereço eletrônico indicado.

Como eu dizia no início do capítulo, o conteúdo deste capítulo serve apenas para instigar a curiosidade de quem estiver se iniciando em *programação*, em *modelagem matemática...*

Agora, se você chegou até aqui e aprendeu Pascal, talvez esteja na hora de escolher outra linguagem de programação para continuar o seu desenvolvimento... há muitas para escolher.

Bibliografia

[1] Borland *Reference guide to TurboPascal- 1990* - Borland - 1992

[2] Borwein, Jon - *NMBRTH@@LISTSERV.NODAK.EDU 28/07/1997* WWW.CECM.SFU.CA/personal/jborwein/Kanada_50b.html jborwein@@cecm.sfu.ca

[3] Cláudio, D. M. e Marins, J.M. *Cálculo Númerico Computacional* - 1994 editora atlas - 2ª Edição

[4] Dieudoné *Calcul Infinitesimal* - 1968 - Collection Méthodes - Herman - Paris.

[5] Feynman, R.P. Leighton, R.B. Sands, M. *Feynman lectures in Physics* - Addison-Wes;ey - 1965

[6] Knuth, D.E. *The Art of Computer Programming, Vols 1,2.* - Addison-Wesley-Reading Ma 1968

[7] Praciano-Pereira, T. *Cálculos numéricos e gráficos usando Pascal.* - Notas de aula policopiadas - BCE - UEM - 1993

[8] Praciano-Pereira, T *Cálculo Numérico Computacional* Notas de aula policopiadas - Biblioteca Central - Univ. do Rio Grande - 1996

[9] Praciano-Pereira, T e Gerônimo, J.R. *Cálculo Diferencial e Integral com apoio computacional* Vol I,II,III - Notas de aula policopiadas - BCE - UEM - 1991

[10] Praciano-Pereira, T. Linux Extreme Pre-Print - Dep. de Matemática - Universidade do Rio Grande - 1998. Ver também http://cart.cheapbytes.com/cgi-bin/cart/LinuX

[11] *Python* - http://www.python.org

[12] *SciLab* - http://www.inria.fr/Scilab

Índice remissivo