Métodos Numéricos em Astronomia

Prof. Alex Cavaliéri Carciofi-USP

Janeiro 2014

carciofi@usp.br

Abstract

Neste breve curso será mostrado diversos problemas intrísecamente matemáticos que podem ser resolvidos com boa precisão usando técnicas e algoritmos voltadas à computação implementadas em Fortran90

Sumário

Blibliografia

- W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, 2007, Numerical Recipes, CUP, 3rd. edition
- D. Quinney, An Introduction to the Numerical Solution of Differential Equations, Research Studies Press and John Wiley & Sons, Revised ed., 1987.
- F. S. Acton, Numerical Methods that Work, Harper & Row, 1970.
- C.H. Asano e Eduardo Colli, Cálculo Numérico e Aplicações, http://www.ime.usp.br/ asano/LivroNumerico/LivroNumerico.pdf
- A.F.P. de C. Humes, I.S.H. de Melo, L.K. Yoshida, W. T. Martins, Noções de Cálculo Numérico, MacGraw Hill do Brasil, 1984.

Capítulo 1

Introdução

1.1 O cálculo numérico e a Astronomia

"(...) *l'Astronomie nous apprenait à ne pas nous effrayer des grands nombres.*"H. Poincaré [1]

No decorrer da história, a Astronomia teve várias vezes o importante papel de quebrar paradigmas da humanidade, ao ponto de Poincaré ter declarado que é ela a responsável por nos transformar em uma alma capaz de compreender a natureza. Para isso a Astronomia sempre desenvolveu, ou utilizou, as mais modernas técnicas numéricas e analíticas disponíveis em cada época, literalmente desbravando os limites do cálculo.

Não por acaso, atualmente a pesquisa em Astronomia tem se defrontado com problemas teóricos cada vez mais complexos e com um crescente volume de dados, que demandam recursos computacionais cada vez maiores e algoritmos cada vez mais inteligentes.

A resposta à pergunta "para que aprender a escrever um código numérico?" é, dessa forma, muito simples: grande parte da pesquisa moderna em Astronomia necessita de computadores. Alguns exemplos:

- *Simulações numéricas* modernas são capazes de resolver problemas que antes eram absolutamente impossíveis. Elas são usadas para se comparar modelos com observações e desta forma testar teorias físicas (Ex: simulação do milênio[2]).

- Dados astronômicos cada vez mais complexos requerem sofisticadas *ferramentas de visualização* para poderem ser estudados.

- A *análise de dados* é uma atividade rotineira em astronomia. Frequentemente essa análise implica no uso de técnicas numéricas para se desempenhar uma série de tarefas em um grande volume de dados.

O objetivo básico da pesquisa científica é confrontar teorias físico-matemáticas com as observações para se compreender os processos físicos que regem os sistemas naturais. Hoje em dia, o desenvolvimento eficiente de uma atividade de pesquisa requer:

- Uso frequente de técnicas numéricas para resolver problemas corriqueiros;

- Uso de novas ferramentas computacionais para tornar os códigos rápidos o suficiente. Ex: Paralelismo, vetorização, GPUs;

[1] "(...) a Astronomia nos ensinava a não temer os grandes números", H. Poincaré, in La Valeur de la Science

[2] http://www.mpa-garching.mpg.de/galform/virgo/millennium/

- Ferramentas numéricas para extração de informações dos dados observacionais;

- Implementação de ferramentas para automação de tarefas;

- Etc.

1.2 Resolução numérica de problemas

O motivo para escrever um programa, qualquer programa, é resolver um problema que é de mais rápida resolução em um computador ou, o que é o mais frequente, somente pode ser resolvido por um computador. A tarefa de escrever um problema pode ser dividida em três partes:

1. Especificar o problema de forma clara (parte mais difícil!);

2. Analisar o problema é reduzi-lo em seus elementos fundamentais;

3. Codificar o programa de acordo com o plano desenvolvido no passo 2;

Frequentemente há ainda um quarto passo:

4. Testar o problema exaustivamente, e repetir passos 2 e 3 até que o programa funcione corretamente em todas as situações.

1.3 Método numérico

Estende-se por método numérico um conjunto de regras estritas sob a forma de uma sequência de operações elementares que levam à solução do problema. Em seu nível mais fundamental (ver cap. 3), utiliza-se somente as quatro operações aritméticas $+$, $-$, $\times$, $\div$.

Esquematicamente, podemos representar a pesquisa em Astronomia através do diagrama da Fig. 1.1. Para ilustrar, consideremos os seguintes exemplos para os diversos elementos da Fig. 1.1:

- *Processo físico:* colisão de galáxias.

- *Observações:* medidas de velocidade radial de galáxias em colisão.

- *Teoria:* gravitação universal e modelos para a estrutura de galáxias.

- *Solução analítica:* equações diferenciais da gravitação.

- *Solução numérica:* emprego de algum método numérico para se resolver equações diferenciais.

O confronto entre teoria e observações frequentemente corresponde ao cerne do problema, e está sujeito à várias fontes de erro. Por um lado há os *erros observacionais*, frequentemente mal compreendidos e de difícil remoção. Esse tópico não será considerado neste curso.

Por outro lado, há os erros inerentes à criação de um modelo matemático (ver Fig. 1.1). Eles podem ser divididos em duas categorias:

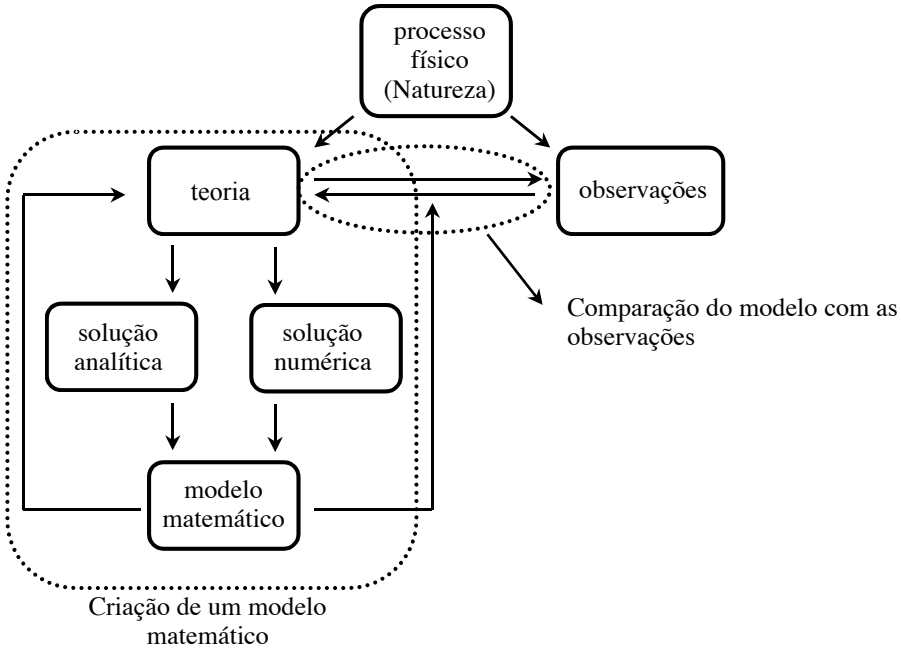


Figura 1.1: Pesquisa científica.

a. Simplificações no modelo matemático

Via de regra a descrição teórica do sistema estudado corresponde a uma simplificação do problema. Em geral há duas razões pelas quais se adotam simplificações:

- Simplificar o problema é a única forma de resolvê-lo;

- Adaptação do modelo ao sistema físico estudado.

Estas simplificações dão origem ao chamado *erro inerente* ao modelo matemático.

Um exemplo do segundo caso acima é aproximar-se a fotosfera solar por uma fotosfera plano paralela. As equações do transporte radiativo tomam uma forma mais simples quando escritas em uma simetria cilíndrica, e por isso frequentemente modelos de atmosferas estelares empregam essa forma das equações. Tais modelos são chamados de *modelos plano-paralelos*. No caso do Sol, a fotosfera tem ≈ 400 km de expessura enquanto o raio do Sol é $\approx 700\,000$ km, o que indica que efeitos da curvatura da fotosfera provavelmente são pequenos e que a aproximação plano-paralela é provavelmente adequada. O mesmo não é válido para a atmosfera de estrelas evoluídas (gigantes e supergigantes); nelas, a atmosfera corresponde a uma fração considerável do volume da estrela.

Outro aspecto ligado a essa questão são os erros instrínsecos na criação de um modelo matemático que advém do desconhecimento da física subjacente ao sistema estudado. Por causa disso, aspectos físicos importantes do modelos podem estar sendo omitidos, ou descritos de forma errada. Os exemplos são inúmeros:

- ao estudar-se espectros estelares, confronta-se linhas observadas com modelos atômicos. Entretanto, o conhecimento de átomos complexos, como Fe, ainda é incompleto;

- a turbulência é um fenômeno comum em astrofísica, e ocorre em situações tão diversas como no meio interestelar e em atmosferas estelares. Não existe ainda uma teoria física satisfatória para a turbulência. O mesmo vale, em grande medida, para fenômenos de convecção, que são muito importantes no interior estelar.

Tendo em vista as questões as questões acima, é importante desenvolvermos uma postura crítica em relação a modelos matemáticos.

b. Erros numéricos

Veremos no cap. 3 que existem três tipos distintos de erros numéricos que podem acometer os modelos matemáticos. Um deles, chamado de *erro de truncamento*, é inerente ao processo matemático de se atingir uma solução através de aproximações sucessivas. Os outros dois (erros de *representação* e *arredondamento*) estão associados ao *hardware* do computador utilizado.

Os três fatores combinados levam à chamada *perda de precisão numérica*, que pode ter resultados catastróficos (literalmente).

Capítulo 2

Elementos do Fortran 90

2.1 Introdução

Neste capítulo vamos apresentar os elementos básicos da linguagem de programação Fortran 90. Como exemplo, vamos considerar o seguinte problema: escrever um programa que pergunte ao usuário as coordenadas x e y de três pontos e que calcule a equação do círculo que passa por esses três pontos, a saber:

$$(x-a)^2 + (y-b)^2 = r^2$$

e então mostre as coordenadas (a, b) do centro do círculo e seu raio, r.

Este problema problema é aparentemente trivial, mas não é tão simples assim! O que ocorre se, por exemplo, os três pontos fornecidos estiverem ao longo de uma reta?

2.1.1 Análise do problema

Basicamente, a análise consiste em criar um *plano estruturado* que envolva níveis sucessivos de refinamento até **atingir um ponto em que o programador possa codificar cada passo individual**. Um exemplo de plano estruturado para o problema acima seria:

a) Ler os três conjuntos de coordenadas (*x1, y1*), (*x2, y2*), (*x3, y3*);

b) Resolver a equação do círculo;

c) Mostrar as coordenadas (a, b) e o raio r

As partes a) e c) são triviais, mas a parte b) requer uma análise mais aprofundada e será abordada no capítulo 4. No momento, lançaremos mão de um recurso comum (além de útil e importante) em programação, que é mover b) para uma **subrotina** (procedimento externo). Dessa forma, um possível código em Fortran 90 estruturado de acordo com o plano acima seria (arquivo `circulo.f90`)

```
PROGRAM circulo
  IMPLICIT NONE

!Esse programa calcula a equacao de um circulo passando por 3
!pontos fornecidos pelo usuario.
!Utiliza-se da subrotina calcula_circulo

!Declaracao de variaveis
  REAL :: x1, y1, x2, y2, x3, y3, a, b, r

!Passo 1: le coordenadas
  PRINT*,"Entre com a coordenada dos tres pontos"
  PRINT*,"na ordem x1,y1,x2,y2,x3,y3"

  READ*,x1,y1,x2,y2,x3,y3

!Passo 2: chama subrotina calcula_circulo
  CALL calcula_circulo(x1,y1,x2,y2,x3,y3,a,b,r)

!Passo 3: escreve resultado na tela
  PRINT*,"O Centro do circulo que passa por esses &
        &pontos eh (",a,",",b,")"
  PRINT*,"Seu raio eh ",r

END PROGRAM circulo
```

2.2 Estrutura Básica

Um programa em fortran 90 tem a seguinte estrutura básica:

```
PROGRAM nome
  IMPLICIT NONE

  [parte de especificação]
  [parte de execução]
  [parte de subprogramas]

END PROGRAM nome
```

2.2.1 Parte de Especificação

```
  REAL :: x1, y1, x2, y2, x3, y3, a, b, r
```

Na parte de especificação, logo após o `IMPLICIT NONE`, é onde se faz a **declaração de variáveis** (e outros tipos de declaração). Variáveis são basicamente endereços lógicos a trechos de memória onde uma determinada informação está armazenada.

Veremos adiante que o computador é capaz de manipular diferentes tipos de variáveis. Os tipos mais comuns são:

```
REAL ::
DOUBLE PRECISION ::
INTEGER ::
CHARACTER ::
LOGICAL ::
COMPLEX ::
```

O `IMPLICIT NONE` deve estar sempre presente! Ele força o programador a declarar explicitamente todas as variáveis, o que é muito importante pois evita uma série de possíveis erros de programação.

2.2.2 Comentários

Linhas cujo primeiro caractere não branco é ! são consideradas comentários e são ignoradas pelo compilador. Comentar bem o código é uma boa prática.

2.2.3 Parte de Execução

```
  PRINT *,"Entre com a coordenada dos tres pontos"
  PRINT *,"na ordem x1,y1,x2,y2,x3,y3"

  READ *, x1,y1,x2,y2,x3,y3

!Passo 2: chama subrotina calcula_circulo
  CALL calcula_circulo(x1,y1,x2,y2,x3,y3,a,b,r)
```

Na parte de execução estão todos os comandos que devem ser executados pelo computador (`PRINT`, `READ`, `CALL`, etc) e que perfazem o chamado **algoritmo**. Criar um algoritmo numérico que seja representável pelo conjunto de comandos disponíveis em uma determinada linguagem corresponde ao cerne do **método numérico**.

2.2.4 Caractere de continuação

```
PRINT *,"O Centro do circulo que passa por esses &
      &pontos eh (",a,",",b,")"
```

Se o último caractere de uma linha for `&`, isso diz ao compilador que o comando continua na próxima linha. Ele é usado em dois contextos:

- Quebrar uma string de caracteres longa. Se o & estiver dentro de uma string de caracteres, então deve haver outro & no inicio da linha seguinte;
- Quebrar linhas de código, de forma a torná-lo mais legível. Ex:

```
CALL calcula_circulo(x1,y1,x2,y2,x3,y3,&!quebrei a linha
                     a,b,r)
```

2.3 Compilando o código

Uma vez tendo o código pronto, devemos **compilá-lo**. Quando dizemos "compilar" um código nos referimos, na verdade, a dois procedimentos distintos.

O primeiro (compilação propriamente dita) é a análise do código pelo compilador, que vai ler e analisar cada linha válida de código (i.e., linhas não vazias e que não comecem com !), procurando verificar se há algum **erro de sintaxe**. Quando isso ocorre, o compilador retorna um ou mais **erros de compilação**. Por exemplo, o código abaixo retornará um erro de compilação:

```
PRINT, "Entre com a coordenada dos tres pontos"
```

Os erros de compilação não têm grande consequência, pois são facilmente detectados pelo compilador. Já **erros semânticos** podem resultar em **erros de execução** (divisão por zero, por exemplo) ou simplesmente podem passar desapercebidos pelo compilador, gerando um resultado incorreto.

Por exemplo, imaginemos um programa de computador capaz de analisar uma frase e verificar se a gramática está correta. Se lhe fornecermos a frase "O gato está sobre o mesa", o programa certamente apontará o erro gramatical ("**o** mesa"). Entretanto, se a frase analisada for "A mesa está sobre o gato", o programa dirá que a frase está gramaticalmente correta, apesar de semanticamente não fazer sentido.

A segunda parte do processo é chamada **linking**, em inglês. Nela, o compilador vai gerar o **código binário** que poderá ser executado pelo compilador.

2.4 Tipos de dados

Em matemática, assim como em programação, há dois tipos básicos de números: **os que são inteiros e os que não o são**.

Em programação, a diferença entre esses dois tipos é **absolutamente vital**, pois está ligada à maneira como os números são representados e armazenados na memória e manipulados pelo processador. Isso será visto em detalhes mais adiante (capítulo 3), no momento basta considerar que:

- Uma variável `INTEGER` é um número inteiro, sempre **armazenado de forma exata** na memória do computador, e tem um intervalo possível de valores relativamente baixo, tipicamente entre -2,1 $\times 10^9$ e 2,1 $\times 10^9$ em um computador de 32 bits;

- Um número `REAL`, por outro lado, é armazenado como uma **aproximação** com um **número fixo de algarismos significativos (7 ou 8)** e tem um intervalo bem maior, tipicamente entre -10^{38} e +10^{38} em um computador com 32 bits;

- Um número `DOUBLE PRECISION` é uma representação usando 64 bits com tipicamente 16 algarismos significativos e um intervalo entre -10^{308} e +10^{308}.

Existem várias funções intrínsecas do Fortran que retornam as propriedades dos números que podem ser representados por um dado processador: `EPSILON`, `HUGE`, `TINY`, etc. Isso será visto em detalhes no próximo capítulo, mas o código abaixo mostra como usar tais funções para mostrar os limites da representação numérica em um dado computador (arquivo `funcoes_numericas.f90`).

```
REAL :: x
DOUBLE PRECISION :: y
INTEGER :: i
INTEGER*8 :: j

PRINT*, &
"O menor numero positivo que somado a 1 retorna numero maior&
& que 1: "
PRINT*, EPSILON(x), EPSILON(y)

PRINT*, &
"O maior numero positivo que pode ser representado: "
PRINT*, HUGE(x),HUGE(y),HUGE(i),HUGE(j)

PRINT*, &
"O menor numero positivo que pode ser representado: : "
PRINT*, TINY(x),TINY(y)
```

Saída do programa `funcoes_numericas.f90`:

```
>>O menor numero positivo que somado a 1 retorna numero maior
>>que 1:
>>1.1920929E-07  2.220446049250313E-016

>>O maior numero positivo que pode ser representado:
>>3.4028235E+38  1.797693134862316E+308 2147483647 9223372036854
>>775807

>>O menor numero positivo que pode ser representado:
>>1.1754944E-38  2.225073858507201E-308
```

Saber os limites da representação numérica de um processador é muito importante. Se durante a execução de um programa uma operação resultar em um número fora do limite a ser representado, ocorre a chamada **exceção de ponto flutuante**. O que acontece depois depende da forma como o programa foi compilado.

2.5 Expressões artiméticas

Uma vez declaradas as variáveis, pode-se atribuir valores a elas de várias formas:

- Usando o `READ`;
- Atribuição direta. Ex: x = 2.2;
- Através de uma **expressão aritmética**.

  Ex: **a = b + c*d/e - f**g/h + i * j + k**

Os operadores aritméticos em Fortran são:

`+` `-` `*` `/` `**` (exponenciação)

Um ponto de fundamental importância é que o Fortran executa as operações acima de acordo com as prioridades abaixo:

1. exponenciação;
2. multiplicação e divisão;
3. adição e subtração.

Dentro do mesmo nível de prioridade, a conta será feita **da esquerda para a direita**, com exceção da exponenciação, que é feita da direita para a esquerda. Por exemplo, vamos estudar como o computador executa a seguinte expressão aritmética:

```
a = b + c*d/e - f**g/h + i * j + k
```

Os passos são:

1. Calcula f**g e salva em temp1
2. Calcula c*d e salva em temp2
3. Calcula temp2/e e salva em temp3
4. Calcula temp1/h e salva em temp4
5. Calcula i*j e salva em temp5
6. Calcula b+temp3 e salva em temp6
7. Calcula temp6-temp4 e salva em temp7
8. Calcula temp7+temp5 e salva em temp8
9. Calcula temp8+k e salva em a.

Em cálculo numérico todo o cuidado é pouco. Os exemplos a seguir ilustram os cuidados que devem ser tomados para se evitar erros semânticos que podem resultar em erros numéricos.

2.5.1 Exemplo 1

Qual o resultado do código abaixo?

```
INTEGER :: i,j

i = 1
j = 3

PRINT*, "Resultado = ", i/j

>>Resultado = 0
```

Por quê? `i` e `j` são inteiros, portanto o que foi feito é uma divisão entre números inteiros, cujo resultado é um inteiro truncado. Outros exemplos:
`5/2 = 2, 10/3 = 3, 1999/1000 = 1`, etc.

2.5.2 Exemplo 2

Qual o resultado do código abaixo?

```
REAL :: x
x = 1/3
PRINT*, "Resultado = ", x

>>Resultado = 0.000000
```

Por quê? Para o compilador, números sem o ponto (Ex: `1`, `3`) são inteiros. Dessa forma o que foi feito foi uma **divisão de inteiros** que resultou em um inteiro que então foi convertido em um número real. **Importante**: o computador sempre faz exatamente o que lhe é dito para fazer...

Para obter o resultado esperado, a sintaxe deveria ser:

```
x = 1./3.    ou   x = 1.0/3.0    ou x = 1.E0/3.E0
...
>>Resultado = 0.333333
```

2.5.3 Exemplo 3

Qual o resultado do código abaixo?

```
REAL :: x
x = 1/3.
PRINT*, "Resultado = ", x

>>Resultado = 0.333333
```

Por quê? Quando há diferentes tipos de dados em uma (sub)expressão, o compilador promove um tipo para outro de acordo com a seguinte prioridade:

`1 - INTEGER` (baixa), `2 - REAL`, `3 - DOUBLE`, `4 - COMPLEX` (alta)

Assim, na expressão acima, o 1, sendo inteiro, foi promovido a real, e a expressão foi calculada entre dois reais.

2.5.4 Exemplo 4

Qual o resultado do código abaixo?

```
REAL :: x,y
INTEGER :: i,j

i = 3.9
x = 3.9
y = 0.1
j = x+y
```

```
PRINT*, "Resultado = ", i,j
>>Resultado = 3  4
```

Porque? Inteiros são sempre truncados em direção ao zero. No caso de j, a operação de truncamento foi feita após a execução da expressão à direita.

Para se obter o inteiro mais próximo, o Fortran tem uma função específica para isso:

```
i = NINT(3.9)
>> Resultado = 4
```

2.5.5 Exemplo 5

Qual o resultado do código abaixo?

```
REAL :: a,b
INTEGER :: c,d

b = 100.
c = 9
d = 10.
a = b*c/d

PRINT*, "Resultado = ", a
a = c/d*b
PRINT*, "Resultado = ", a

>>Resultado = 90.000000
>>Resultado = 0.000000
```

Em cálculo numérico **não necessariamente valem várias propriedades das operações aritméticas** em matemática, tais como associatividade e distributiva (ver capítulo 3). Em outras palavras, **a ordem dos fatores altera o resultado!**

2.6 List-directed input and output

Vimos atrás a função dos comandos `READ` e `PRINT`.

```
REAL :: a, b
INTEGER :: c

READ*, a, b, c
PRINT*, a, b, c
```

O "* " significa que os dados impressos ou lidos serão formatados automaticamente de acordo com a lista de variáveis apresentadas (list-directed). O formato atribuído depende do tipo da variável

Ex: se a=1.1, b=2.2 e c=10, o `PRINT` acima retorna:

```
>>       1.100000     2.200000    10
```

O que ocorre se o input do programa for: `1.1 2.2 10.`? Ocorre um erro, pois o programa está esperando que o terceiro número seja um número inteiro.

2.7 Procedimentos, subprogramas e funções

Um procedimento é uma seção especial de um programa que é chamada, de alguma forma, sempre que necessário. Procedimentos podem ser intrínsecos (ou seja, parte do Fortran) ou escritos pelo programador ou por terceiros. Há uma categorização adicional, dependendo da forma como esses procedimentos são usados: funções e subroutinas.

Basicamente, o propósito de uma função é tomar um ou mais valores (ou argumentos) e criar um único resultado. O Fortran tem inúmeras **funções intrínsecas**:

```
!Argumento de funcoes trigonometricas devem estar em radianos!
SIN(x)
LOG(x)
LOG10(x)
SQRT(x)
EXP(x)
etc.
```

Uma lista das funções intrínsecas do Fortran 90 pode ser encontrada em *http://www.nsc.liu.se/ boein/f77to90/a5.html*.

2.8 Funções externas

A sintaxe básica para se criar uma função externa no Fortran é:

```
<tipo> FUNCTION  nome (arg1, arg2, ..., argn)
  IMPLICIT  NONE

  [parte de especificação]
  [parte de execução]

END FUNCTION  nome
```

Funções são usadas/criadas por vários motivos:

1. Organização do código: programação modular;
2. Evitar duplicação de código (funções podem ser chamadas várias vezes);
3. Uso de códigos de terceiros.

Vale lembrar que a filosofia por trás dos procedimentos é que o programa principal não precisa saber nada sobre os detalhes internos do procedimento: **compartimentalização do código**.

2.8.1 Exemplos

Função que calcula a raiz cúbica de um argumento (arquivo `raiz_cubica.f90`).

```
REAL FUNCTION raiz_cubica(x)
IMPLICIT NONE

!Declaracao do argumento da funcao
  REAL, INTENT(in) :: x

!Variaveis locais
  REAL :: log_x

!Calcula a raiz cúbica usando logaritmos
  log_x = LOG(x)
  raiz_cubica = EXP(log_x/3.)

END FUNCTION raiz_cubica
```

Note que:

- `log_x` é uma variável interna, não é vista pelo programa principal
- o atributo `INTENT(in)` diz ao computador que a variável x não pode ser modificada durante a execução.

Este outro exemplo mostra uma função que lê um valor de uma variável inteira. Note que essa função não tem argumento.

```
INTEGER FUNCTION le_valor()
IMPLICIT NONE

!Le numero
  PRINT*, "Por favor entre um numero inteiro: "
  READ*, le_valor

END FUNCTION le_valor
```

Tal função pode ser chamada pelo programa principal da seguinte forma, por exemplo:

```
soma = le_valor()*le_valor()
```

2.9 Subrotinas

A diferença entre uma subroutina e uma função está na forma como o programa se refere a ela e como os resultados (se houver) são retornados.

Subrotinas são acessadas pelo comando `CALL`, da seguinte forma:

```
CALL nome(arg1, arg2, ...)
```

O `CALL` causa uma **transferência de controle** para a subrotina, de forma que o programa é interrompido temporariamente e as instruções internas da subrotina são executadas.

A lista de argumentos de uma subrotina pode conter argumentos de entrada (input), de saída (output), ou nenhum argumento. A sintaxe básica é:

```
SUBROUTINE  nome (arg1, arg2, ..., argn)
  IMPLICIT  NONE
  [parte de especificação]
  [parte de execução]
END SUBROUTINE  nome
```

No arquivo `raizes.f90` encontra-se o exemplo abaixo, que é uma subrotina que calcula a raiz quadrada, cúbica, quádrupla e quíntupla de um número.

```
SUBROUTINE raizes(x,x2,x3,x4,x5)
IMPLICIT NONE

!Declaracao do input
  REAL, INTENT(in) :: x
!Declaracao do output
  REAL, INTENT(out) :: x2,x3,x4,x5
!Variaveis locais
  REAL :: log_x

!Calcula raiz quadrada
  x2 = SQRT(x)

!Calcula as outras raizes usando logaritmos
  log_x = LOG(x)

  x3 = EXP(log_x/3.)
  x4 = EXP(log_x/4.)
  x5 = EXP(log_x/5.)

END SUBROUTINE raizes
```

Exemplo de programa que chama a subrotina raizes:

```
PROGRAM exemplo
IMPLICIT NONE

  REAL :: arg, raiz2, raiz3, raiz4, raiz5

  PRINT*, "Entre com valor do argumento: "
  READ*, arg

!Chama a subrotina
  CALL raizes(arg, raiz2, raiz3, raiz4, raiz5)
```

```
  PRINT*, "Raizes = ", raiz2, raiz3, raiz4, raiz5
END PROGRAM
```

Rodando:

```
>>Entre com valor do argumento:
10.
>>Raizes = 3.162278    2.154435    1.778279    1.584893
```

Observação importante sobre subrotinas: *a ordem, tipo e número de argumentos usados no programa que chama a subrotina deve corresponder exatamente ao que foi definido na subrotina*. Coisas imprevistas podem acontecer se isso não for obedecido! Por exemplo, uma subrotina com os seguintes argumentos:

```
SUBROUTINE teste(a,b,c)
...
REAL :: a,b,c
...
```

chamada pelo seguinte programa:

```
...
DOUBLE PRECISION :: a,b,c
...
CALL teste(a,b,c)
...
```

resultará em erro.

Além do `INTENT(in)` e `INTENT(out)`, existe outro atributo que pode ser usado em uma subrotina:

```
  dum = INTENT(inout)
```

Esse argumento significa que a variável `dum` pode ser usada tanto para passar quanto receber informações da subrotina.

O uso dos `INTENT` *é muito importante, como uma salvaguarda para erros de programação.*

2.10 Controlando os passos do programa

O Fortran possui duas construções que podem ser usadas para tomadas de decisão durante o programa. São elas: `IF` e `CASE`. Antes de ver como usá-las, vamos estudar um pouco sobre variáveis `LOGICAL` e também sobre expressões lógicas.

2.10.1 variáveis e expressões lógicas

As variáveis lógicas são variáveis de 1 bit apenas: 0 significa falso e 1, verdadeiro (na verdade 1 bit é efetivamente utilizado, mas elas são armazenadas em palavras de 32 bits).

Declaração:

```
LOGICAL :: var1, var2, etc...
```

Para atribuir valores a elas, pode-se fazê-lo diretamente:

```
var1 = .TRUE.
var2 = .FALSE.
```

ou através de expressões lógicas, que fazem uso dos chamados operadores relacionais:

```
>       !maior
>=      !maior ou igual a
<       !menor
<=      !menor ou igual a
==      !igual a
/=      !diferente de
```

e dos operadores lógicos:

```
.NOT.   !nao logico
.AND.   !e logico
.OR.            !ou logico
.EQV.   !equivalencia logica
.NEQV.  !nao equivalencia logica
```

Exemplos de expressões lógicas:

```
var1 = 5 > 4
var2 = 2 > 2
var3 = SQRT(x) > 1. .AND. SQRT(x) <= 2.
var4 = a == b .OR. a /= c
var5 = .NOT. (a < b)
var6 = a == b .EQV. a /= c
```

2.10.2 A construção IF

As variáveis e expressões lógicas são usadas em conjunto com a construção `IF` para permitir a tomada de decisões ao longo de um programa. A forma básica dessa construção é

```
IF (expressão lógica 1) THEN
   comandos 1
ELSE IF (expressão lógica 2) THEN
   comandos 2
ELSE IF (expressão lógica 3) THEN
```

```
          comandos 3
ELSE
   comandos 4
END IF
```

O número de IFs depende da problema em questão, e pode ser apenas um. Nesse caso pode-se usar uma versão simplificada:

```
IF (expressão lógica 1) comando
```

Como exemplo, vamos fazer um programa que:

1. Leia o nome de um aluno;
2. Leia as notas dos 4 EPs de AGA0503;
3. Leia as notas das duas provas;
4. Calcule a média final;
5. Escreva a média final e se o aluno foi aprovado, reprovado ou ficou em recuperação.

Uma possível solução está no programa `media_final.f90`:

```
PROGRAM media_final
!Programa que le o nome do aluno, as notas dos
!EPs e das provas, calcula a media final
!e diz se o aluno foi aprovado ou nao
!Usa a funcao le_valor
IMPLICIT NONE
!Definicao de variaveis

        CHARACTER(LEN=10) :: nome,situacao
        REAL :: EP1, EP2, EP3, EP4
        REAL :: P1, P2
        REAL :: media

        REAL, EXTERNAL :: le_valor

!Le nome do aluno
        PRINT*, "Digite o nome do aluno:"
        READ*, nome

!Le as notas dos EPs
        EP1 = le_valor("EP1")
        EP2 = le_valor("EP2")
        EP3 = le_valor("EP3")
        EP4 = le_valor("EP4")

!Le as notas das provas
        P1 = le_valor("P1 ")
```

```
        P2 = le_valor("P2 ")

!Calcula a media final
        media = 0.4*(EP1+EP2+EP3+EP4)/4. + &
                    0.6*(P1+P2)/2.

!Determina situacao
        IF (media < 3.) THEN
                situacao = "reprovado"
        ELSE IF (media < 5.) THEN
                situacao = "recuperacao"
        ELSE
                situacao = "aprovado"
        ENDIF

!Escreve resultado
        PRINT*, "O aluno "//TRIM(nome)//" teve media = ", media
        PRINT*, "Sua situacao: "//situacao


END PROGRAM

REAL FUNCTION le_valor(oque)
IMPLICIT NONE

        CHARACTER(LEN=3), INTENT(in) :: oque

!Le numero
        PRINT*, "Digite a nota do "//oque//":"
        READ*, le_valor

END FUNCTION le_valor
```

2.10.3 SELECT CASE

Uma alternativa ao IF é o CASE:

```
SELECT CASE (expressão)
  CASE (seletor 1)
    comandos 1
  CASE (seletor 2)
    comandos 2
  CASE (seletor 3)
    comandos 3
  ............
  CASE (seletor n)
    comandos n
  CASE DEFAULT
    comandos padrao
```

```
END SELECT
```

Exemplo:

```
INTEGER :: mes

SELECT CASE (mes)
  CASE (1:2)
    PRINT*, "Ferias de verao!"
  CASE (3:6)
    PRINT*, "Aulas do primeiro semestre"
  CASE (7)
    PRINT*, "Ferias de inverno!"
  CASE (8:11)
    PRINT*, "Aulas do segundo semestre"
  CASE DEFAULT
    PRINT*, "Acho que estamos em dezembro..."
END SELECT
```

2.11 Repetindo partes de um programa

Uma grande parte das técnicas matemáticas baseia-se em alguma forma de **processo iterativo** em que a solução é atingida através de passos sucessivos.

Por exemplo, pode-se calcular o valor de π através da série

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \ldots \tag{2.1}$$

Assim, para calcular π usando a série acima (o que, na verdade, seria uma péssima idéia...), temos que somar os termos sucessivamente até atingirmos a precisão desejada.

Além disso, o **processamento de dados** em geral requer que a mesma ação seja aplicada repetidamente para cada dado. Assim, um dois mais importantes conceitos em programação é a habilidade de repetir uma sequência de comandos, seja um número pre-determinado de vezes seja até que determinada condição seja satisfeita.

2.11.1 Loop do DO com contador

```
DO contador = inicial, final, [incremento]

  comandos

END DO
```

Notas:

- O contador e os valores à direita do = devem ser variáveis ou expressões inteiras
- O incremento ao contador é optativo

- Quando o `DO` é iniciado, o **número de iterações** é calculado usando-se a seguinte expressão:

$$MAX\left(\frac{final - inicial + inc}{inc}, 0\right) \tag{2.2}$$

Quando o resultado acima é 0, o **loop não é executado**. A tabela abaixo mostra alguns exemplos do comportamento do `DO` para diferentes argumentos. Note que mesmo no caso em que nada é executado pelo loop (última linha da tabela, em que o número de iterações é 0) é atribuído um valor do contador (`i`).

`DO`	Número de iterações	Valores do contador
`DO i=1,10`	10	1,2,3,4,5,6,7,8,9,10
`DO j=20,50,5`	7	20,25,30,35,40,45,50
`DO j=20,-20,-6`	7	20,14,8,2,-4,-10,-16
`DO i=4,5,4`	1	4
`DO i=6,1`	0	6

Exemplo de uma função inteira que calcula o fatorial de um número inteiro (arquivo `fatorial.f90`):

```
INTEGER FUNCTION fatorial(arg)
IMPLICIT NONE

  INTEGER, INTENT(in) :: arg
  INTEGER :: i !Contador do loop

!Caso em que arg eh 0
  IF (arg == 0) THEN
    fatorial = 1
  ELSE
    fatorial = 1
    DO i=1,arg
      fatorial = fatorial*i
    ENDDO
  ENDIF
END FUNCTION fatorial
```

2.11.2 Comando EXIT

O `EXIT` permite terminar o loop quando alguma condição for satisfeita. Esse tipo de construção é muito útil quando se usa métodos iterativos para calcular algo com precisão pré-determinada

```
DO
  .
  .
  IF (precisao < epsilon) EXIT
  .
```

```
END DO
!Após o EXIT o programa segue seu curso normal
```

O uso do código acima é perigoso pois pode-se incorrer num loop infinito, caso a condição do `IF` nunca seja satisfeita. Para contornar este problema basta impor um número máximo de iterações, como no exemplo abaixo:

```
DO i=1,max_iteracoes
  .
  .
  IF (precisao < epsilon) EXIT
  .
END DO
```

2.11.3 Exercício

Escrever um programa que calcule o valor de π usando a série da Eq. (2.1), com a precisão definida pelo usuário e que imprima o resultado e o número de termos que foram somados.

Como definir se a precisão foi atingida?

2.12 Arrays

Em computação científica é frequentemente necessário manipular conjuntos de dados ordenados, tais como vetores e matrizes. Outro tarefa comum é repetir a mesma sequência de operações em um conjunto de dados.

Ambos requerimentos são atendidos em Fortran pelo uso de *arrays*, que permitem que um conjunto de dados seja acessado simultâneamente (como um único objeto).

Há duas formas de se declarar um array:

```
REAL, DIMENSION(50) :: a

REAL :: a(50)
```

Ambas as formas são equivalentes. Para imprimir os elementos 11 a 14 de a basta escrever:

```
PRINT*, a(11:14)
```

Por padrão o primeiro índice de um array é 1, mas isso pode ser modificado, como no exemplo abaixo:

```
REAL, DIMENSION(-10:1) :: a
```

Em Fortran, pode ocorrer confusão entre arrays e funções pois a sintaxe é semelhante. Considere o exemplo abaixo:

```
REAL, DIMENSION(10) :: primeiro
REAL :: segundo
INTEGER :: i
REAL :: a, b

...
a = primeiro(i)
b = segundo(i)
...
```

Os dois comandos têm a mesma forma, mas no primeiro o `(i)` é o índice de um array, então o compilador simplesmente atribui o i-ésimo valor de primeiro em `a`. Já no segundo comando, como não há o atributo de dimensão na variável `segundo`, o compilador assumirá que se trata de uma função real externa.

2.12.1 Inicializando um array

Arrays podem ser inicializados de várias formas. Atribuição direta:

```
arr_inteiro = (/1,2,3,4,5,6,7,8,9,10/)
```

Para arrays grandes, usa-se o DO implícito:

```
arr_inteiro = (/(i,i=1,10)/)
```

Essa é uma construção poderosa, que pode tomar várias formas:

```
arr_real = (/(SQRT(REAL(i)),i=1,1000)/)

arr_inteiro = (/((0,i=1,9),10*j,j=1,10)/)
```

Exercício: qual o valor do array acima?

2.12.2 Lendo e escrevendo um array

Quando aparecem em expressões sem referências aos índices, então implicitamente está se referindo ao array todo:

```
PRINT*, arr_inteiro
```

Subarrays também podem ser facilmente extraídos:

```
PRINT*, (p(i), i=1,99,2)
PRINT*, p(10:20)
```

Exemplo de uma construção perigosa:

```
READ*, primeiro, ultimo, (arr(i), i=primeiro,ultimo)
```

Por quê é perigosa? Se não tomar cuidado pode-se ter problemas de memória (**segmentation fault**, ou falha de segmentação). Por exemplo, se `arr` for um array de 1 a 10, e o valor de `ultimo` for 11, teremos problemas...

2.12.3 Arrays em expressões aritméticas

Definições:

1. Dois arrays são **conformes** se eles tiverem a mesma **forma** (igual número de dimensões e números de elementos em cada dimensão).
2. Um escalar é sempre conforme com um array.
3. Todas as operações intrínsecas são válidas entre dois arrays conformes.

Exemplos de expressões aritméticas envolvendo arrays:

```
REAL :: a(20),b(20),c(20),maximo, soma, media
REAL :: x(3),y(3)
REAL :: pe, co

a = b*c !Esse comando é equivalente à:
DO i=20
  a(i) = b(i)*c(i)
END DO

a = 10.*c
c = 2.
b = SIN(c)
maximo = MAX(a)          !Retorna o maior valor do array

! Calcula a soma de todos os elementos do array
soma = SUM(a)

! Calcula a média aritmética dos elementos do array
media = SUM(a)/SIZE(a)

! Calcula o produto escalar de dois vetores
pe = SUM(x*y)

! Calcular o comprimento de um vetor:
co = SQRT(SUM(x*y))
```

2.12.4 Arrays em procedimentos

O Fortran permite que arrays em procedimentos tenham um número de elementos indefinido. Esse número é definido ao longo da execução. Este recurso é extremamente útil, e deve ser adotado sempre, como uma boa prática de programação.

Exemplo:

```
SUBROUTINE exemplo(array1,array2)
IMPLICIT NONE

REAL, DIMENSION(:) :: array1, array2
INTEGER :: N
...
!Tamanho do array
N = SIZE(array1)
...
```

Se essa subrotina for chamada de um programa que tem as declarações:

```
REAL, DIMENSION(20) :: a,b
...
CALL exemplo(a,b)
```

então internamente as variáveis array1 e array2 terão dimensão 20.

2.13 Formatos: controlando o input e output

Vimos antes o list-directed input e output. Apesar de práticos, os comandos `READ*` e `PRINT*` não são suficientes na maioria das situações, pois frequentemente é necessário mais controle na forma da entrada e saída.

O fortran permite tal controle através de **strings de formatação**. Dessa forma, o `READ*` e o `PRINT*` são substituídos por comandos mais gerais, como abaixo:

```
READ(*, formato) arg1, arg2, ...

WRITE(*, formato) arg1, arg2, ...
```

sendo que `formato` se refere a uma string (ou uma variável string) que contém o código de formatação.

Para formatar **números reais**, usa-se os descritores `F` e `E`:

- `NFw.d`: escreve um número real nas próximas `w` letras com `d` casas decimais. Esse formato será repetido `N` vezes.
- `NEw.d`: escreve um número real em formato exponencial nas próximas `w` letras com `d` casas decimais. Lembrar que 4 caracteres são usados pelo expoente. Esse formato será repetido `N` vezes.

Para formatar caracteres usa-se o descritor A: `NAw`. Exemplo abaixo ilustra como se deve usar estes formatadores (ver arquivo `formatos.f90`)

```
WRITE(6,"(5A16)") "Argumento", "Raiz Quadrada", "Raiz &
&Cubica", "Raiz Quadrupla", "Raiz Quintupla"

WRITE(6,"(5F16.4)") arg,raiz2, raiz3, raiz4, raiz5

WRITE(6,"(5E16.4)") arg,raiz2, raiz3, raiz4, raiz5
```

A saída do código acima é:

```
>> Argumento    Raiz Quadrada      Raiz Cubica  Raiz Quadrupla
>>   10.0000           3.1623           2.1544          1.7783
>>0.1000E+02      0.3162E+01       0.2154E+01      0.1778E+01
>>Raiz Quintupla
>>         1.5849
>>     0.1585E+01
```

Para formatar números inteiros, usa-se o descritor `I`:

- `NIw`: escreve um número inteiro nas próximas `w` letras. Esse formato será repetido `N` vezes.

Exemplo:

```
WRITE(*,"(A,I1,A,F6.4)") "x**1/",2," = ",raiz2
```

Saída:

```
>>x**1/2 = 4.4721
```

2.14 Usando arquivos

Para poder ler e escrever dados de uma outra unidade lógica além do teclado (tipicamente um arquivo, mas pode ser outro dispositivo), deve-se inicialmente conectar-se a essa unidade. Isso é feito pelo comando `OPEN`. Ex:

```
OPEN(FILE="tabela.txt", UNIT=7, STATUS="NEW", ACTION="WRITE")
```

A `UNIT` representa uma unidade lógica, associada a um número inteiro. Usando-se diferentes unidades, pode-se acessar vários arquivos simultaneamente.

O `STATUS` é útil para se definir certas restrições ao uso de um dado arquivo. Por exemplo, podemos querer garantir que não escreveremos nada por cima de um arquivo. Há três opções principais

- `OLD`: nesse caso, o arquivo *já deve existir previamente* para poder ser aberto.
- `NEW`: nesse caso, o arquivo *não deve existir previamente* para poder ser criado.
- `UNKNOWN`: nesse caso, se o arquivo já existe, ele será tratado como `OLD`, caso contrário como `NEW`.

Podemos também especificar que tipo de ações de input/output são permitidas usando o `ACTION`, que pode tomar três valores:

- `READ`: nesse caso o arquivo é tratado como sendo *apenas para leitura*. Não é possível escrever nele.
- `WRITE`: nesse caso o arquivo é *apenas para escrita*. Comandos `READ` não são permitidos

- `READWRITE`: tanto input quanto output são permitidos.

Arquivos são acessados usando-se os comandos `READ` e `WRITE`. Por exemplo, se houver um arquivo associado à unidade 7, podemos escrever neste arquivo usando o seguinte comando:

```
WRITE(UNIT=7,FMT=<formato>) ...
```

ou simplesmente

```
WRITE(7,<formato>) ...
```

Quando o acesso ao arquivo não é mais necessário, deve-se fechar a conexão:

```
CLOSE(UNIT=7)
```

O exemplo `arquivos.f90` usa o código do programa `formatos.f90` e escreve o resultado num arquivo cujo nome é fornecido pelo usuário.

2.15 Exercícios

- Fazer um algoritmo que leia 10 valores reais e imprima uma tabela cuja primeira coluna seja formada por estes números, a segunda coluna apresente a parte inteira desses valores e a terceira coluna apresente estes valores em notação científica. Considere, por facilidade, valores de 0 a 100 com um máximo de três casas decimais.

- Desenvolver um programa que informe os números primos entre 1 e 1000.

Capítulo 3

Erros, Precisão Numérica e Ponto Flutuante

No capítulo anterior introduzimos o conceito de *variável* em programação. Uma variável é basicamente um nome usado para se referir a algum conteúdo na memória do computador. Além disso, vimos que ela contém também informações sobre o *tipo de dados* e, associado a esse tipo, há um conjunto de regras de como os dados devem ser manipulados. Por exemplo, vimos que uma divisão entre dois números reais (`REAL`, `DOUBLE PRECISION`) não é o mesmo que uma divisão entre inteiros (`INTEGER`).

Na prática, variáveis podem ser muito mais complexas, mas nesse capítulo descreveremos apenas como variáveis inteiras e reais são representadas pelo computador, e discutiremos as conseqüências dessa representação para o cálculo numérico.

3.1 Representação

Computadores *não* armazenam números com precisão infinita; usam, ao contrário, alguma aproximação que pode ser reduzida em um número fixo de *bits* (*binary digits*, em inglês) ou *bytes* (grupos de 8 bits). Quase todos os computadores permitem ao usuário uma escolha entre diferentes *representações* ou *tipos de dados*. Tipos de dados podem variar no número de bits utilizados, mas, mais fundamentalmente, na forma como o número é representado. Exemplos importantes são representações de inteiros e de ponto-flutuante (*floating-point*).

3.1.1 Representação de inteiros

Em computação há dois tipos de inteiros, os inteiros positivos (*unsigned*, sem sinal) e os inteiros negativos e positivos (*signed*, com sinal).

Inteiros positivos são simplesmente o número escrito em binário. Nesse caso, dados t dígitos binários, b_i ($i = 1,\ t-1$), a conversão entre o que está representado no computador e o sistema decimal é feita através da expressão

$$b_{t-1}2^{t-1} + b_{t-2}2^{t-2} + \cdots + b_2 2^2 + b_1 2^1 + b_0 2^0 \,. \tag{3.1}$$

O máximo inteiro positivo que pode ser representado é, portanto, $2^t - 1$. Por exemplo, com 8 bits (ou seja, 8 dígitos binários), só é possível representar os inteiros entre 0 a 255:

Decimal	Representação na memória
0	00000000
1	00000001
2	00000010
255	11111111

Os tipos mais comuns de inteiro aceitam negativos e são feitos dividindo-se os números representáveis aproximadamente à metade, com os positivos no começo e os negativos no final. Dessa forma, o maior valor negativo representável é -2^{t-1} e o maior positivo representável é $2^{t-1} - 1$. Por exemplo, com 8 bits, só é possível representar os números inteiros entre -128 e $+127$.

Decimal	Representação na memória
0	00000000
1	00000001
127	01111111
-128	10000000
-127	10000001
-126	10000010
-2	11111110
-1	11111111

Em Fortran os inteiros são em geral com sinal, mas existem algumas implementações que permitem a declaração de inteiros sem sinal. Tipicamente, um `INTEGER` é representado com 32 bits (4 palavras de 1 byte) e um `INTEGER*8` é representado com 64 bits. Assim, o maior valor de um `INTEGER` é $2^{31} - 1 = 2.147.483.647$ e o maior valor de um `INTEGER*8` é $2^{63} - 1 = 9.223.372.036.854.775.807$.

Vale lembrar que uma representação inteira é sempre exata. Expressões aritméticas envolvendo inteiros são também exatas, desde que

1. a resposta não esteja fora do intervalo representável, e,
2. divisão seja interpretada como uma *divisão inteira* que produz um resultado inteiro, desconsiderando qualquer resto.

O que ocorre quando se tenta atribuir a uma variável um inteiro fora do intervalo representável? O resultado pode ser um *overflow* ou um *rollover*. Por exemplo, se tentarmos compilar um código em Fortran que contenha a linha:

```
PRINT*, HUGE(i)+10
```

provavelmente teremos um erro de compilação (`arithmetic overflow`). Entretanto, o código abaixo não resultará em erro de compilação, e provavelmente o que o programa fará durante a execução é um *rollover*, ou seja, algo parecido com o que ocorre com um odômetro de carro quando ele atinge a máxima quilometragem. O código abaixo retorna o valor -2147483639.

```
i = HUGE(i)
PRINT*, i+10
```

Não considerar o tamanho dos números inteiros é um erro comum e com resultados em geral muito graves. Por exemplo, o código abaixo vai retornar um número negativo para o quadrado de um inteiro:

```
i = 50000
PRINT*, i**2
```

3.1.2 Representação de ponto-flutuante

Para representar números reais com o maior intervalo possível adota-se a chamada *representação de ponto-flutuante*. Nesta representação os números são armazenados com um número (aproximadamente) fixo de algarismos significativos e são escalados para cima ou para baixo usando-se um expoente. O termo ponto-flutuante refere-se ao fato de o ponto decimal (ou no caso de computadores, ponto binário) poder "flutuar", isto é, poder ser colocado em qualquer lugar relativo aos algarismos significativos do número. Isso é equivalente à chamada notação científica, que representa um número como a multiplicação de outro número e uma potência de 10.

Em ponto-flutuante, um número é representado internamente através de um bit de sinal, S (interpretado como mais ou menos), um expoente inteiro exato, E, e uma mantissa binária, M. Juntos, eles representam o número

$$S \times M \times 2^{E-e} \tag{3.2}$$

onde e é um viés do expoente, uma constante inteira fixa que depende da máquina usada e da representação implementada. Outra maneira de se escrever a representação de um numero real x com m dígitos binários na mantissa é

$$x = \pm .b_1 b_2 b_3 \ldots b_m \times 2^{E-e} \tag{3.3}$$

Note que o valor de b_1 é sempre não nulo. Tal representação, dita normalizada, maximiza o número de algarismos significativos e portanto a precisão do número representado.

Vejamos dois exemplos de números binários normalizados usando-se $m = 8$:

- $x_1 = 0.11100110 \times 2^2$.
  O correspondente na base 10 desde número é dado por:

$$x_1 = \left[2^{-1} + 2^{-2} + 2^{-3} + 2^{-6} + 2^{-7}\right] \times 2^2 = 3.59375.$$

- $x_2 = 0.11100111 \times 2^2$. O correspondente na base 10 desde número é dado por:

$$x_2 = \left[2^{-1} + 2^{-2} + 2^{-3} + 2^{-6} + 2^{-7} + 2^{-8}\right] \times 2^2 = 3.609375.$$

Note que, no sistema de representação usado, x_1 e x_2 são dois números consecutivos, isto é, não se pode representar nenhum outro número real entre x_1 e x_2. Por exemplo, o decimal 3.6 não teria representação exata nesse sistema. Essa é a origem de um tipo de erro em cálculo chamado *erro de representação*; esse erro é inerente ao sistema de representação adotado.

Em um computador moderno, os 32 bits correspondentes a uma variável tipo `REAL` são divididos em três partes: 1 bit para o sinal, 8 bits para o expoente e 23 bits para a mantissa, da seguinte forma:

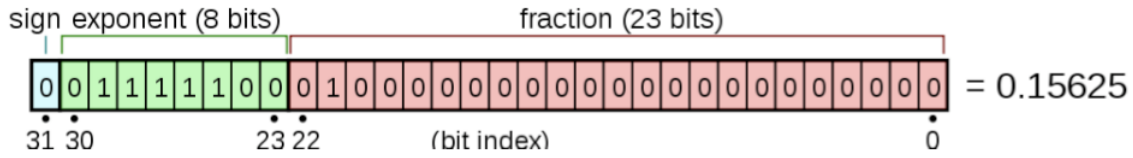


Figura 3.1: Distribuições dos bits em uma representação de ponto-flutuante

Nestes computadores, o termo *precisão simples* (single precision) corresponde a números com 32 bits (ou 4 bytes) e o termo *precisão dupla* (double precision) denota números com 64bits (ou 8 bytes). A tabela abaixo elenca algumas propriedades dos números de 32 e 64 bits.

A utilização de bits conforme a tabela acima permite a representação de números com os seguintes limites:

Precisão	simples	simples	dupla	dupla
Relação ao zero	mais distante	mais próximo	mais distante	mais próximo
Cálculo	$\pm(1-2^{-24})\times 2^{128}$	$\pm(1-2^{-24})\times 2^{-126}$	$\pm(1-2^{-53})\times 2^{1024}$	$\pm(1-2^{-53})\times 2^{-1022}$
Valor	$\pm 3,40\times 10^{38}$	$\pm 1,17\times 10^{-38}$	$\pm 1,79\times 10^{308}$	$\pm 2,22\times 10^{-308}$

Além dos números reais, o sistema de ponto-flutuante reservou alguns números especiais: +0.0, -0.0, $+\infty$, $-\infty$, e `NaN` (*not a number*, na sigla em inglês). Esses números especiais permitem que o processador lide com as chamadas exceções de ponto flutuante. Por exemplo:

```
x = HUGE(x)
PRINT*, x*2. !retorna +infinito
x = -1.
PRINT*, SQRT(x) retorna NaN
```

A ocorrência de `NaN` durante a execução é potencialmente grave, pois ela pode se propagar a outras variáveis. Isso ocorre porque:

- `NaN`*número = `NaN`

- `NaN`¡ número = `.FALSE.`

- `NaN` ¿ número = `.FALSE.`

- `NaN` == `NaN` = `.FALSE.`

Uma fonte interessante de informação sobre ponto-flutuantes pode ser encontrada na Wikipedia[1] sob os termos *IEEE 754-1985* ou *IEEE floating point arithmetic*.

3.2 Erros numéricos devido ao uso de computador digital

Os erros de representação, oriundo do fato de que nem todo número real pode ser representado exatamente no sistema binário, já foram descritos acima. Veremos agora os erros de arredondamento e truncamento.

[1] *http://www.wikipedia.org*

3.2.1 Erros de arredondamento

A aritmética entre dois números na representação de ponto-flutuante frequentemente não é exata, mesmo que os operandos possam ser representados exatamente. Para compreender isso é necessário considerar a maneira como dois números são somados. Isso é feito em dois passos: primeiro desloca-se para a direita a mantissa do menor número, dividindo-a por 2, enquanto que o expoente é aumentado, até que os dois operandos tenham o mesmo expoente, quando então os números podem ser somados. Esse procedimento descarta os algarismos menos significativos do menor operando, o que pode resultar em uma adição inexata.

Por exemplo, para somar os seguintes números binários normalizados em um sistema de ponto-flutuante que tem $m = 8$:

$$\begin{aligned} x_1 &= .11100110 \times 2^5 \\ x_2 &= .11001111 \times 2^2 \end{aligned}$$

Desloca-se a mantissa de x_2:

$$\begin{aligned} x_2 &= .01100111\mathbf{1} \times 2^3 \\ x_2 &= .00110011\mathbf{11} \times 2^4 \\ x_2 &= .00011001\mathbf{111} \times 2^5 \end{aligned}$$

e então faz-se a soma. Note que algarismos marcados em negrito serão descartados no processo!

O menor número que adicionado ao 1.0 produz um número diferente de 1.0 é chamado de precisão da máquina, ϵ_m, e é fornecido pela função intrínseca do Fortran `EPSILON`. Em outras palavras, o ϵ_m é a precisão relativa (ver abaixo) na qual números de ponto-flutuante podem ser representados, e corresponde a uma mudança do último dígito da mantissa. Praticamente toda operação aritmética em ponto-flutuante introduz um erro fracional de ao menos ϵ_m. Esse tipo de erro é chamado de erro de arredondamento (*roundoff*).

Erros de arredondamento se acumulam quando uma longa série de cálculos é feita. Se, para se obter um resultado, foram feitas N operações aritméticas, em geral o *mínimo* erro de arredondamento será da ordem de $\sqrt{N}\epsilon_m$ caso os erros ocorram de forma aleatória (a $\sqrt{N}$ vem do caminho aleatório). Se houver uma regularidade no sinal de ϵ_m, então o erro total será da ordem de $N\epsilon_m$.

Além disso, algumas ocasiões especialmente problemáticas favorecerão erros de arredondamento em uma simples operação. Em geral elas estão associadas à subtração de dois números muito próximos, dando um resultado cujos algarismos significativos são aqueles poucos algarismos em que os números diferiam.

Por exemplo, considere a expressão familiar para a solução da equação quadrática (Fórmula de Bhaskara)

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \tag{3.4}$$

A adição do numerador torna-se problemática sempre que $b > 0$ e $|4ac| \ll b^2$, pois nesse caso haverá uma subtração de dois números muito próximos. A solução para esse problema em particular é simples, e está mostrada no programa `cap03_bhaskara.f90`.

Vamos denotar $\oplus$, $\ominus$, $\otimes$ e $\oslash$ as operações em ponto-flutuante. Em geral:

$$\begin{aligned} x \oplus y &\neq x + y \\ x \otimes y &\neq x \times y \end{aligned}$$

Da mesma forma a associatividade e a distributiva também não são sempre válidas:

$$x \otimes (y + z) \neq x \otimes y + x \otimes z$$

3.2.2 Erros de truncamento

Erros de arredondamento são uma característica do hardware do computador. Existem um outro tipo diferente de erro que é uma característica do algorítimo usado e que é independente do hardware. Em geral ocorrem porquê muitos métodos computam *aproximações discretas* para uma quantidade contínua desejada.

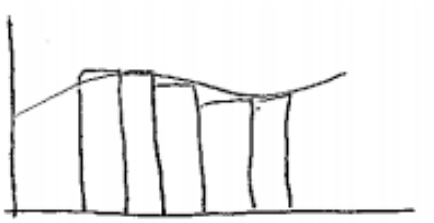
Exemplos:

- Calcular números que são o resultado de séries infinitas. Vimos anteriormente o exemplo de π, que pode ser calculado a partir da série:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \dots .$$

  O programa `cap03_pi.f90` ilustra a implementação numérica deste exemplo em particular.

- Aproximação de áreas por retângulos:



A discrepância entre o resultado correto e a resposta é chamado de erro de truncamento. Tal erro persistiria mesmo em um computador perfeito hipotético, com uma representação infinitamente acurada. A minimização de erros de truncamento corresponde ao cerne do campo da análise numérica!

3.3 Perda de precisão numérica

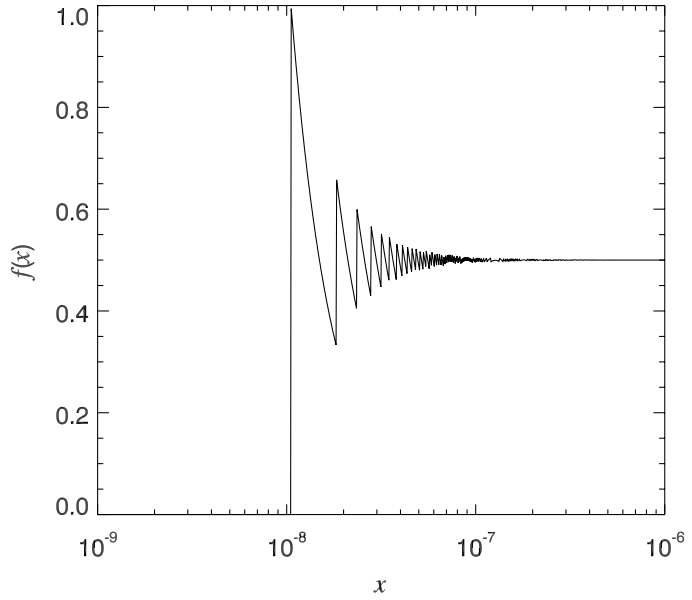
Em geral, quando um problema numérico é resolvido por um computador, os três tipos de erro descritos acima estão presentes e concorrem para a chamada *perda de precisão numérica*. Dependendo da magnitude desta perda de precisão, os resultados podem tornar-se completamente errados e sem sentido.

Como exemplo, considere a função

$$f(x) = \frac{1 - \cos(x)}{x^2} .$$

É possível mostrar (ver abaixo) que

$$\lim_{x \to 0} f(x) = \frac{1}{2} .$$



Entretanto, se calcularmos o valor de $f(x)$ *em dupla precisão*, obteremos o resultado mostrado na Figura 3.3.

Qual a origem do problema? Sabemos que $\lim_{x\to 0}\cos(x) = 1$. Assim, para valores pequenos de x a expressão $1 - \cos(x)$ tende a zero e começa a perder precisão, finalmente retornando o valor 0 quando $1 - \cos(x) < \epsilon_m$. Solução? Calcular $f(x)$ a partir de uma expansão em série. A expansão em série de $\cos(x)$ é bem conhecida:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots .$$

A partir dela, podemos escrever $f(x)$ como

$$f(x) = \frac{1}{2} - \frac{x^2}{4!} + \ldots .$$

Da expressão acima é evidente que $\lim_{x\to 0} f(x) = 1/2$. Para implementar corretamente uma função numérica que calcula $f(x)$, temos que usar ora a expansão acima, para valores pequenos de x, ora $f(x)$ propriamente dita, para valores maiores do argumento. O programa `cap03_precisao.f90` mostra uma possível implementação.

3.4 Erro Absoluto e Erro Relativo

Define-se erro absoluto, e_x, como:

$$e_x = x - \bar{x} \tag{3.5}$$

onde $\bar{x}$ é o valor verdadeiro da grandeza e x seu valor aproximado. O erro relativo, ϵ_x, é definido como

$$\epsilon_x = \frac{|e_x|}{\bar{x}} = \frac{|x - \bar{x}|}{\bar{x}} . \tag{3.6}$$

Deve-se notar que o valor do erro absoluto pode ser pequeno enquanto que o erro relativo é grande. Por exemplo:

$$\begin{aligned} \bar{x} &= 0.00052 \\ x &= 0.00061 \\ e_x &= 0.00009 \\ \epsilon_x = \frac{|e_x|}{\bar{x}} &= 0.17 = 17\% \end{aligned}$$

3.4.1 IFs envolvendo número reais

Se números reais são representados de forma aproximada por um computador, devemos ter cuidado com expressões do tipo:

```
REAL :: x, y
...
IF (x == y) THEN ...
```

Pode ocorrer a situação em que, *matematicamente*, `x` e `y` devem ser iguais um ao outro, mas que devido a erros de truncamento, arredondamento ou representação eles não mais sejam exatamente iguais. Por exemplo, considere o código:

```
REAL :: x,y
...
x = 2.
...
y = SQRT(x)
...
IF (y*y == x) THEN
...
```

A expressão lógica dentro do `IF` é matematicamente verdadeira, mas provavelmente será avaliada como falsa pelo computador.

A solução para esse problema é rescrever a expressão lógica de forma que a comparação entre dois reais x e y seja feita de seguinte forma:

$$\frac{|x-y|}{x} < \epsilon \tag{3.7}$$

onde ϵ é uma precisão previamente escolhida, que depende do problema em questão. Pode-se, por exemplo, escolher ϵ como um múltiplo da precisão da máquina, ϵ_m. Uma solução para o problema acima seria:

```
REAL :: x,y
...
x = 2.
...
y = SQRT(x)
....
IF (ABS(y*y-x)/x < 10.*EPSILON(x)) THEN
....
```

3.5 Exercícios

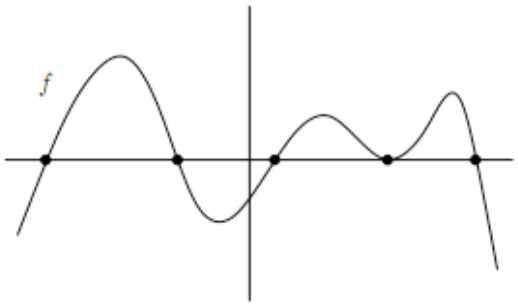
1.

Capítulo 4

Zeros de Funções

4.1 Introdução

Considere o seguinte problema: *"dada uma função real f, achar suas* raízes*, isto é, os valores de x para os quais f(x)=0"*, como ilustra a figura abaixo (os pontos pretos indicam as raízes da função representada no desenho).



Este pode, a princípio, parecer um problema específico, mas ele aparece toda vez que tivermos uma *equação* a ser resolvida. Uma equação nada mais é do que uma expressão

$$f_1(x) = f_2(x)\,,$$

onde procuramos o(s) valor(es) de x que a satisfaça(m). Ora, mas isso é o mesmo que achar as raízes da função $f(x) = f_1(x) - f_2(x)$.

Além disso, o problema se relaciona com a *inversão de funções*. Por exemplo, temos uma função $g(x)$ conhecida, mas gostaríamos de determinar g^{-1} em certos pontos. Lembrando que $g^{-1}(y)$ é definido como sendo o valor x tal que $g(x) = y$ temos que, para um dado y, resolver a equação $g(x) = y$ é determinar $x = g^{-1}(y)$. Resolver a equação $g(x) = y$ é o mesmo que achar um zero da função $f(x) = g(x) - y$.

Nas próximas seções veremos alguns exemplos que ilustram o problema.

4.2 Exemplos de aplicação

4.2.1 Raiz cúbica de um número k

Suponha que queiramos achar um número $\bar{x}$ positivo tal que $\bar{x}^3 = k$. Esse número é o que denominamos a raiz cúbica de k, ou $\sqrt[3]{k}$.

Graficamente, encontramos $\bar{x}$ pela intersecção de $y = x^3$ com $y = 10$. Observe também que o problema é equivalente a resolver a equação

$$x^3 - k = 0\,,$$

ou seja, estamos procurando a raiz de $f(x) = x^3 - k$.

Um caso desta aplicação é dado na seção 4.4.1.

4.2.2 O cilindro deitado [1]

Considere um cilindro colocado horizontalmente sobre um plano, paralelo ao solo. O cilindro tem uma abertura, na parte superior, para a colocação da água (para dramatizar o exemplo, imagine um contêiner de petróleo, gigante, com esse formato e nessa posição). O problema é: como determinar uma escala com marcações que indiquem o volume de água dentro do cilindro (e não simplesmente a altura do nível da água)?

Para ver a relação entre essa questão e o problema de achar o zero de uma função, quantifiquemos um pouco mais o problema. Seja l o comprimento do cilindro e r o raio de uma seção transversal, perpendicular ao seu eixo. O volume total do cilindro é dado por

$$v = l \times \pi r^2$$

pois πr^2 é a "área da base"e l a "altura"do cilindro, embora ele esteja deitado.

Se ele estiver cheio até a altura h então o volume de água ali contido será l vezes a área preenchida pela água numa seção transversal qualquer, que chamaremos de $A(h)$. Note que h varia entre 0 e $2r$, e que $A(0) = 0$, $A(2r) = \pi r^2$ e $A(r) = \frac{1/2}{\pi} r^2$. Mas e os outros valores de h? Como achar a função $A(h)$?

Aqui podemos fazer um pouco de geometria: supomos que $h < r$ (o raciocínio será completamente análogo para $h > r$) e consideramos o ângulo θ formado entre a vertical e a linha L. A relação entre h e θ é simples: $r\cos\theta + h = r$, ou seja, $h = r(1 - \cos\theta)$.

Lembremos agora que a área de um setor de ângulo θ pode ser achada por regra de três, lembrando que para $\theta = 2\pi$ a área é πr^2

$$\begin{array}{ccc} \theta = 2\pi & \to \pi r^2 & \\ \theta & \to a(\theta) & \end{array} \quad \Rightarrow a(\theta) = \tfrac{1}{2}\theta r^2$$

Como mostra a figura, a área que queremos calcular é menor do que a área de dois setores de ângulo θ (perfazendo θr^2), e o excedente é a área de dois triângulos-retângulos.

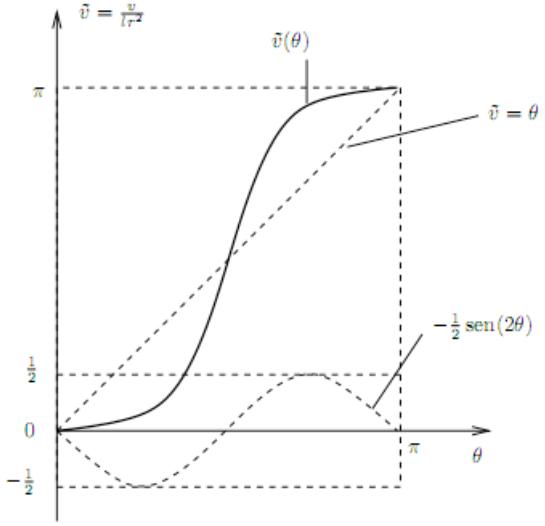
A área excedente é o produto de $d_1 d_2$, onde $d_1 = r\cos\theta$ e $d_2 = r\sin\theta$. Logo

$$A(h) = \theta r^2 - r^2 \sin\theta\cos\theta = r^2(\theta - \frac{1}{2}\sin 2\theta),$$

lembrando que θ depende de h pela relação $h = r(1 - \cos\theta)$. Essa conta sugere que talvez seja mais fácil fazer a escala ao longo do contorno do cilindro, parametrizado pelo ângulo θ, como se fossem as marcas de um relógio (pode-se fazer uma escala vertical, mas as contas ficarão mais complicadas).

É fácil ver que a mesma fórmula vale quando $h > r$ (verifique!). Resumindo, o volume $v(\theta)$ depende de θ pela fórmula

$$v(\theta) = lr^2(\theta - \frac{1}{2}\sin 2\theta),$$



onde θ varia entre 0 e π. O gráfico de $v(\theta)$ (na verdade, o gráfico de $\tilde{v} = v(\theta)/lr^2$) está esboçado na figura abaixo.

Na figura, colocamos na vertical a variável $\tilde{v} = \frac{v}{lr^2}$, de forma que o gráfico fique independente do raio r e do comprimento l do cilindro. As linhas pontilhadas indicam as duas funções (θ e $-\frac{1}{2}\sin 2\theta$) que somadas produzem a função $\tilde{v}(\theta) = \frac{v(\theta)}{lr^2}$.

A função $\tilde{v}(\theta)$ tem a derivada nula em $\theta = 0$ (e por simetria em $\theta = \pi$), pois

$$v'(\theta) = 1 - \frac{1}{2} \times 2\cos 2\theta$$

e

$$v'(0) = 1 - \cos(0) = 0.$$

Suponha agora que o volume total do cilindro seja da ordem de 10 litros e que queremos marcar, no contorno do cilindro, o valor de $\bar{\theta}$ correspondente a um volume de água de 3 litros. Isso corresponde, no gráfico, a achar o valor de $\bar{\theta}$ para o qual $v(\bar{\theta}) = 3$ (se o volume for medido em litros).

Esse é o problema de achar a raiz da função $v(\theta) - 3$. O mesmo procedimento pode ser adotado para se calcular as marquinhas correspondentes a outros valores do volume, de forma que toda a escala possa ser construída.

4.2.3 Cálculo da temperatura de uma camada circunstelar de poeira

Vamos considerar o seguinte problema astrofísico: uma estrela de raio R e temperatura efetiva $T_{\rm ef}$ circundada por uma camada esférica de poeira com raio interno R_i e raio externo R_e (Figura 4.1). A poeira é formada por grãos de raio a, com uma densidade numérica n (número de grãos por cm^3) e uma temperatura $T_{\rm poeira}$. Essa camada de poeira é opticamente fina (ou seja, tem baixa densidade, de forma que a fração da radiação estelar absorvida pela poeira é relativamente baixa). Esse sistema descreve, de forma aproximada, uma estrela do ramo assintótico das gigantes circundada por uma camada de matéria que foi ejetada ao longo desta etapa evolutiva.

Suponhamos que meçamos em um telescópio a magnitude dessa estrela nas bandas J e K, e que saibamos, a partir de outras fontes, a temperatura e o raio da estrela, as

[1] Extraído de Asano & Coli 2009

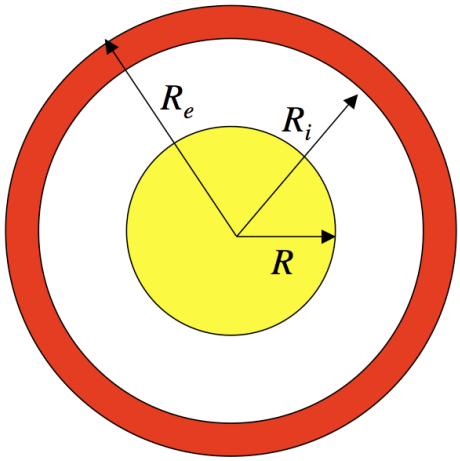


Figura 4.1: Estrela circundada por uma camada de poeira.

dimensões da camada de poeira e a temperatura dos grãos. Como podemos determinar, a partir dessas informações, a densidade numérica dos grãos?

Inicialmente, vamos lembrar da definição de magnitude. A diferença entre as magnitudes J e K guarda a seguinte relação com os fluxos luminosos nas respectivas bandas espectrais:

$$J - K = -2,5 \log \left(\frac{F_J}{F_K} \right) . \tag{4.1}$$

O fluxo F_λ em um comprimento de onda λ será a soma da luz emitida pela estrela com a luz emitida pela camada de poeira

$$F_\lambda = \frac{L_\star(\lambda) + L_{\text{poeira}}(\lambda)}{4\pi d^2} , \tag{4.2}$$

onde

- $L_\star$ é a *luminosidade* da estrela (i.e., sua energia radiante no comprimento de onda λ emitida por segundo);
- L_{poeira} é a luminosidade da camada de poeira; e
- d é a distância da estrela até nós.

A luminosidade da estrela relaciona-se com o seu raio e sua temperatura efetiva através da relação

$$L_\star(\lambda) = 4\pi R^2 B_\lambda(T_{\text{ef}}) , \tag{4.3}$$

onde B é a famosa *expressão para o espectro de um corpo negro.*

A expressão para a luminosidade da camada de poeira é um pouco mais complicada. Vamos inicialmente considerar a luminosidade de um único grão de poeira de raio a. Ela é exatamente equivalente à Eq. 4.3

$$L_{\text{grao}}(\lambda) = 4\pi a^2 B_\lambda(T_{\text{poeira}}) . \tag{4.4}$$

A luminosidade da camada de poeira será dada simplesmente pela multiplicação de $L_{\text{grao}}(\lambda)$ pelo número total de grãos na camada, ou seja

$$L_{\text{poeira}}(\lambda) = N L_{\text{grao}}(\lambda) = \frac{4\pi(R_e^3 - R_i^3)}{3} n L_{\text{grao}}(\lambda) . \tag{4.5}$$

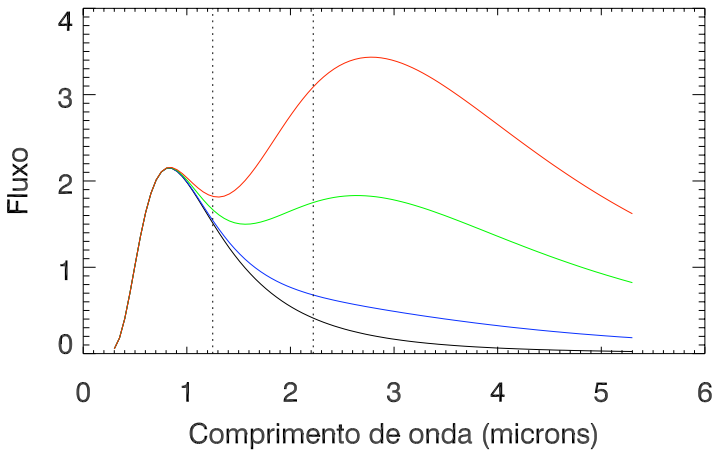


Figura 4.2: Espectro de uma estrela circundada por uma camada de poeira. Cada linha corresponde a uma densidade de grãos, a saber: preto: 0, azul: 10^{-4} cm^{-3}, verde: 5×10^{-3} cm^{-3}, vermelho: 10^{-3} cm^{-3}. O fluxo está mostrado em unidades arbitrárias.

A partir das Eqs. 4.1, 4.3 e 4.5 podemos escrever uma expressão que relaciona o índice de cor observado $(J-K)$ com as características físicas da estrela e da poeira

$$(J-K)+2,5\log\frac{R^2B_J(T_{\rm ef})+(4\pi a^2/3)(R_e^3-R_i^3)B_J(T_{\rm poeira})n}{R^2B_K(T_{\rm ef})+(4\pi a^2/3)(R_e^3-R_i^3)B_K(T_{\rm poeira})n}=0 \qquad (4.6)$$

onde os subscritos J e K indicam a função de corpo negro calculada para $\lambda=1.6$ μm e 2.2 μm, respectivamente. Note que a expressão final *não depende da distância à estrela*, pois o termo $4\pi d^2$ é cancelado na fração acima. A solução do nosso problema, isto é, qual a densidade numérica de grãos n que corresponde ao índice de cor observado, pode ser obtida encontrando-se a raiz da função acima.

Na Figura 4.2 mostramos o espectro emergente de um sistema com os seguintes parâmetros: $R=100$ $R_\odot$ e $T_{\rm ef}=3500$ K, $a=1$ μm, $R_i=900$ $R_\odot$, $R_e=1000$ $R_\odot$. Cada curva corresponde ao espectro para uma dada densidade de grãos, como indicado na figura. Para $n=0$, o espectro é dado simplesmente pelo espectro da estrela (não há poeira). À medida que aumentamos n, vemos que a poeira passa a contribuir cada vez mais para o espectro, gerando um *excesso de fluxo no infravermelho* (note que a temperatura dos grãos de poeira é bem menor que a temperatura da estrela). Eventualmente, para n muito grande a poeira vai dominar o espectro completamente.

Na Figura 4.3 vemos o índice de cor para o sistema descrito acima versus n. O índice de cor parte de um valor de negativo, pois a estrela emite mais luz na banda J do que na banda K, e chega a um valor positivo para altos valores de n, caso em que a poeira domina o espectro.

4.3 Algorítmos iterativos

Nesta e nas próximas seções, desenvolveremos métodos iterativos (não são *interativos*, atenção!) para a determinação de raízes de funções. Por *algoritmo iterativo* entende-se um

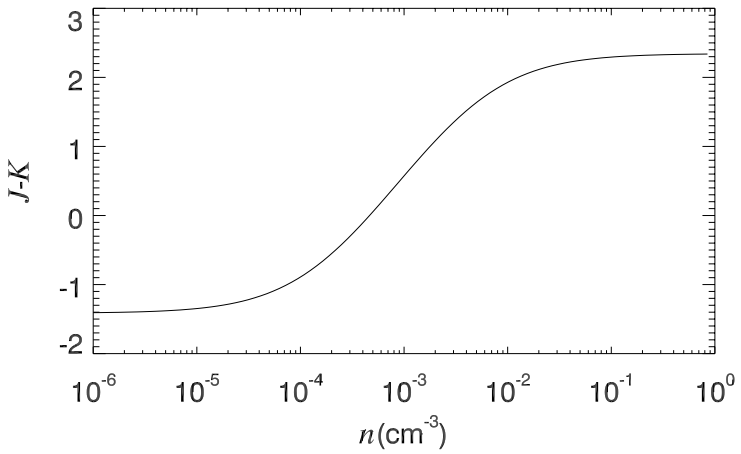


Figura 4.3: Índice de cor vs. n para a estrela mostrada na Figura 4.2.

processo que calcula uma sequência de aproximações x_1, x_2, x_3, ... da solução desejada. O cálculo de uma nova aproximação é feito utilizando as aproximações anteriores.

Diz-se que o processo iterativo *converge* para $\bar{x}$, a solução procurada, quando

$$\lim_{i \to N} x_i = \bar{x} \tag{4.7}$$

num número N de passos, que pode ser finito ou infinito, dependendo do algoritmo usado. Vamos estudar processos iterativos para calcular o valor aproximado para a raiz de uma função dada, que chamaremos de $\tilde{x}$. A diferença entre o valor exato da raiz, $\bar{x}$, e seu valor aproximado é o chamado erro. Como não podemos determinar o valor exato do erro, uma vez que não sabemos $\bar{x}$, vamos procurar delimitá-lo, ou seja, garantir que $|\bar{x} - \tilde{x}| < \epsilon$. Nesse caso, escreveremos $\bar{x} = \tilde{x} \pm \epsilon$ e diremos que que $\tilde{x}$ tem uma precisão ϵ.

Alguns dos métodos iterativos que veremos a seguir pressupõem um valor inicial dado, ou seja, uma estimativa inicial do valor da raiz. Dessa maneira, a determinação de uma raiz de uma dada função será feita em duas etapas:

1. determinar uma estimativa inicial para a raiz da função e/ou um intervalo $[a, b]$ onde exista somente uma única raiz;

2. determinar a solução ali contida através de um método iterativo, com uma precisão pré-fixada ϵ.

4.3.1 Localização de raízes isoladas

O primeiro passo acima, a localização dos zeros, poderá ser feita através de um gráfico ou de uma tabela da função. Estudando o comportamento da função, teremos condições de determinar um intervalo $[a, b]$ que contenha somente uma raiz da mesma. Para este intervalo, esboçaremos o gráfico da função ou faremos uma tabela de seus valores, determinando assim uma primeira aproximação da raiz.

Na pesquisa do intervalo que contenha ao menos um zero real, seja através de gráfico ou tabela, é muito útil o uso do Teorema de Bolzano. Esse teorema diz:

Se f for uma função contínua num intervalo [a,b] e trocar de sinal nos extremos desse intervalo, então existe pelo menos uma raiz real de f no intervalo [a,b].

Outro recurso que pode ser utilizado é transformar a equação $f(x) = 0$ numa equação equivalente da forma $g(x) = h(x)$ e buscar a intersecção do gráfico das duas funções.

Para os algoritmos que veremos a seguir, precisamos determinar uma aproximação inicial da raiz e/ou um intervalo que contenha apenas esta raiz. Assim, se a partir do gráfico suspeitarmos da existência de duas ou mais raízes próximas ou coincidentes, é aconselhável um estudo detalhado do comportamento da função no intervalo [a,b].

Para ilustrar, vamos pesquisar as raízes reais da função

$$f(x) = x\ln(x) - 3.2 \tag{4.8}$$

A função $f(x) = x\ln(x) - 3.2$ está definida somente para valores positivos de x. Tabelando-se $f(x)$ nos pontos x=1,2,3 e 4 escolhidos arbitrariamente, obtemos:

x	1	2	3	4
$f(x)$	-3,20	-1,81	0,10	2,36

Pelo Teorema de Bolzano concluímos que existe pelo menos uma raiz real no intervalo [2,3]. Além disso, vemos que $f(x) < 0$, para $x \leq 1$ e que $f(x)$ é monotônica estritamente crescente para $x > 1$. Desses fatos concluímos que existe uma única raiz real de $f(x)$, isolada no intervalo [2, 3].

Exercício

Pesquise a raiz reais da função $f(x) = 5\log(x) - 2 + 0,4x$ transformando-a na equação equivalente $5\log(x) = 2 - 0,4x$. Faça um esboço do gráfico das duas funções acima para determinar o valor aproximado da raiz.

Resposta: o intervalo [1, 2] contém uma única raiz e essa raiz é aproximadamente 1,8.

4.4 O método da dicotomia ou bissecção

O Teorema de Bolzano nos sugere um processo bastante simples para achar uma aproximação de uma raiz de uma função. Supondo que uma raiz da função f esteja isolada no interior do intervalo $[a, b]$ e, portanto, $f(a)f(b) < 0$, o processo consiste em dividir o intervalo dado ao meio e, por aplicação do Teorema de Bolzano aos subintervalos,

$$\left[a, \frac{a+b}{2}\right] \quad \text{e} \quad \left[\frac{a+b}{2}, b\right]$$

determinar qual deles contém a raiz. O processo é repetido para o novo subintervalo até que se obtenha uma precisão prefixada, isto é, o intervalo obtido seja menor ou igual a 2 vezes a precisão desejada. A Figura 4.4 detalha o método da bissecção. É importante reforçar que no intervalo $[a, b]$ deve haver uma *única* raiz da função, caso contrário o método pode convergir para uma solução indesejada.

4.4.1 Exemplos

Vamos determinar um valor aproximado da raiz quadrada de 5, com erro menor ou igual a 0,01, usando o método da dicotomia. Como vimos, determinar $\sqrt{5}$ é equivalente a determinar o zero positivo da equação $x^2 - 5 = 0$. Sabemos que o intervalo $[2, 3]$ contém esta raiz.

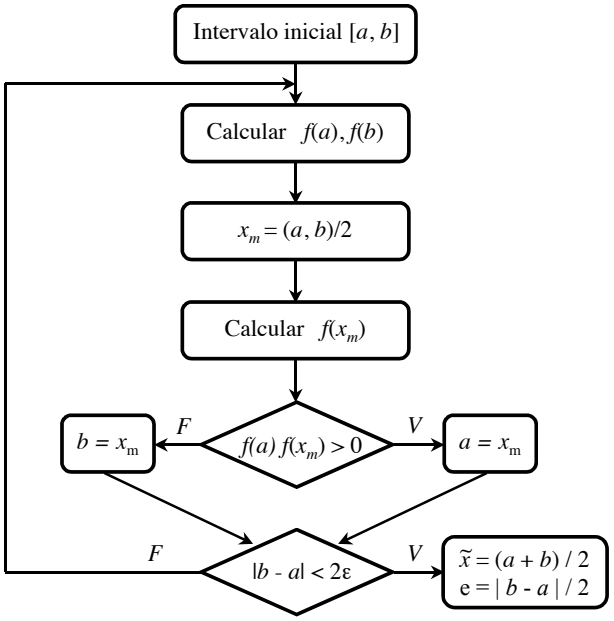


Figura 4.4: O método da bissecção.

Em cada iteração i, i=0,1,2,..., denotaremos por a_i e b_i os extremos inferior e superior, respectivamente, do intervalo que está sendo considerado, por $\tilde{x_i}$ o valor aproximado da raiz e por ϵ_i o erro máximo cometido na i-ésima iteração. Esses valores estão dispostos na tabela abaixo. Inicialmente temos $f(a_0) = f(2,0) < 0$ e $f(b_0) = f(3,0) > 0$.

i	a_i	b_i	$\tilde{x_i} = \frac{b_i+a_i}{2}$	$\epsilon_i = \left\lvert\frac{b_i-a_i}{2}\right\rvert$	Sinal de $f(\tilde{x_i})f(a_i)$
0	2,0	3,0	2,5	0,5	-
1	2,0	2,5	2,25	0,25	-
2	2,0	2,25	2,125	0,125	+
3	2,125	2,25	2,1875	0,0625	+
4	2,1875	2,25	2,21875	0,03125	+
5	2,21875	2,25	2,234375	0,015625	+
6	2,234375	2,25	2,2421875	0,0078125	

Tabela 4.1: Método da Bissecção aplicado na obtenção de $\sqrt{5}$.

Portanto, $\sqrt{5}$=2,2421±0,0078.

4.4.2 Convergência

É possível mostrar matematicamente que

$$\lim_{i\to\infty} \tilde{x_i} = \bar{x}$$

ou seja, que o método da bissecção converge e que, na i-ésima iteração, a resposta $\tilde{x_i}$ tem precisão ϵ_i. Entretanto, os erros de arredondamento podem comprometer não somente a precisão, mas também a convergência do processo para a solução exata (ou seja, a acurácia). Para exemplificar este fato, vamos repetir o cálculo da raiz quadrada de 5 usando aritmética de ponto flutuante com 2 algarismos significativos. Obtemos os resultados da tabela a seguir.

i	a_i	b_i	$\tilde{x_i} = \frac{b_i+a_i}{2}$	$\epsilon_i = \left\lvert\frac{b_i-a_i}{2}\right\rvert$	Sinal de $f(\tilde{x_i})f(a_i)$
0	2	3	2,5	0,5	-
1	2	2,5	2,3	0,25	-
2	2	2,3	2,2	0,15	+
3	2,2	2,3	2,3	0,05	-
4	2,2	2,3	2,3	0,05	-

Tabela 4.2: Método da Bissecção aplicado na obtenção de $\sqrt{5}$ com apenas 2 dígitos de precisão.

Para i=3 e i=4 os valores da tabela 4.2 se repetem devido aos erros de arredondamento. Assim, 2,3 é a melhor aproximação em ponto flutuante com dois dígitos.

4.4.3 Propriedades do método da bissecção

O método da bissecção é à prova de falhas. Se o intervalo $[a, b]$ contiver mais de uma raiz, a bissecção encontrará uma delas (mas não necessariamente a raiz procurada). Se o intervalo contiver uma singularidade, o método convergirá para a singularidade. Por exemplo, se aplicarmos o método da bisseção na função

$$f(x) = \frac{1}{x - c}\,,$$

ele convergirá para c.

Como em cada passo a precisão do método aumenta por um fator de dois (ou seja, intervalos sucessivos são sempre a metade dos intervalos anteriores)

$$\epsilon_{i+1} = \epsilon_i/2\,,$$

podemos facilmente calcular o número de iterações requeridas para atingir um dado erro ϵ. Esse número será

$$n = \log_2 \frac{b-a}{\epsilon}\,.$$

Por exemplo, para obter uma precisão de 15 casas decimais são necessárias aproximadamente 50 iterações pois $2^{-50} \approx 10^{-15}$.

Exercício
Derive a expressão acima para o número de iterações.

4.4.4 Ordem da convergência

De uma forma geral, um método é dito de convergência de ordem m se na vizinhança da solução temos

$$|\epsilon_{n+1}| = \text{constante} \times |\epsilon_n|^m$$

ou

$$\lim_{n\to\infty} \frac{|\epsilon_{n+1}|}{|\epsilon_n|^m} = \text{constante}\,.$$

No caso da bissecção, constante $= 1/2$ e $m = 1$ e o método é dito de convergência linear. Quanto maior a ordem da convergência, mais eficiente o algoritmo. Algoritmos com $m > 1$ são ditos superlineares.

4.5 Método das substituições sucessivas

Neste método a sequência de aproximações da raiz $\bar{x}$ de uma função $f(x)$ é obtida através de uma *relação de recorrência* do seguinte tipo:

$$x_{i+1} = G(x_i), \qquad i = 0, 1, 2, \ldots \tag{4.9}$$

onde x_0 é uma aproximação inicial de $\bar{x}$ e $G(x)$ é uma função obtida transformando $f(x)$ numa expressão equivalente da forma $x = G(x)$. Como no caso do método da bissecção, é necessário que no intervalo $[x_{\min}, x_{\max}]$ haja uma única raiz de $f(x)$, onde $x_{\min}$ e $x_{\max}$ são os valores mínimos e máximos das estimativas de $\bar{x}$ feitas durante a iteração.

Por exemplo, a função $f(x) = x^2 + 0.96x - 2.08$ tem uma raiz positiva e outra negativa. As funções abaixo foram obtidas através manipulando-se $f(x)$

$$x = G_1(x) = x^2 + 1.96x - 2.08$$

$$x = G_2(x) = \frac{2.08 - 0.96x}{x}\,.$$

Vamos aplicar a relação de recorrência tomando $x = G_2(x)$. Ou seja

$$x_{i+1} = \frac{2.08 - 0.96x_i}{x_i}\,.$$

Partindo da aproximação inicial $x_0 = 0.5$, obtemos

$$\begin{aligned}
x_1 &= 1.4247, \quad \epsilon_1 = 0.9247\\
x_2 &= 0.8722, \quad \epsilon_2 = 0.5524\\
x_3 &= 1.1352, \quad \epsilon_3 = 0.2630\\
x_4 &= 0.9927, \quad \epsilon_4 = 0.1425\\
x_5 &= 1.0652, \quad \epsilon_5 = 0.0724\\
x_6 &= 1.0271, \quad \epsilon_6 = 0.0381\\
x_7 &= 1.0468, \quad \epsilon_7 = 0.0197\\
x_8 &= 1.0365, \quad \epsilon_8 = 0.0103\\
x_9 &= 1.0418, \quad \epsilon_9 = 0.0053
\end{aligned}$$

A relação de recorrência converge e após nove iterações tempos o resultado $\tilde{x} = 1.0418 \pm 0.0053$. Na Figura 4.5 mostramos uma representação gráfica da convergência usando $x = G_2(x)$.

Vamos agora aplicar a relação de recorrência tomando $x = G_1(x)$ e outra aproximação inicial ($x_0 = 1.05$). O resultado é o seguinte

$$\begin{aligned}
x_1 &= 1.0805, \quad \epsilon_1 = 0.0305\\
x_2 &= 1.2053, \quad \epsilon_2 = 0.1248\\
x_3 &= 1.7350, \quad \epsilon_3 = 0.5297\\
x_4 &= 4.3306, \quad \epsilon_4 = 2.5956\\
x_5 &= 25.1619, \quad \epsilon_5 = 20.8314
\end{aligned}$$

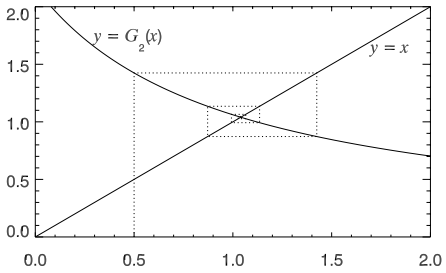


Figura 4.5: Interpretação geométrica da convergência pelo método das substituições sucessivas.

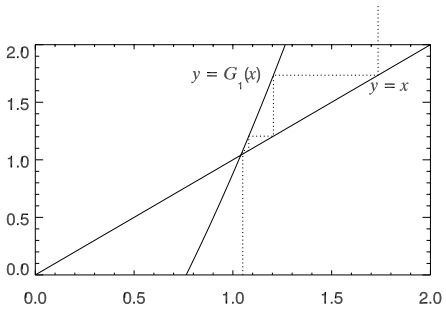


Figura 4.6: Interpretação geométrica da divergência da relação de recorrência quando usamos a função $G_1(x)$.

Esse exemplo mostra que dependendo da transformação $x = G(x)$ escolhida a relação de recorrência pode ser convergente ou divergente.

4.5.1 Estudo de convergência

Seja $\bar{x}$ o valor da raiz $f(x)$ no intervalo $[a, b]$. O erro cometido na n-ésima iteração é

$$\epsilon_{n+1} = x_{n+1} - \bar{x}\,.$$

Aplicando a relação de recorrência, Eq. 4.9, obtemos

$$\epsilon_{n+1} = G(x_n) - G(\bar{x}) = G(\bar{x} + \epsilon_n) - G(\bar{x})\,. \tag{4.10}$$

Pela expansão em séries de Taylor (Eq. 4.13), temos que

$$G(\bar{x} + \epsilon_n) = G(\bar{x}) + \epsilon_n G'(\bar{x})\,. \tag{4.11}$$

Substituindo a Eq. (4.11) na (4.10) obtemos

$$\epsilon_{n+1} = G'(\bar{x})\epsilon_n\,. \tag{4.12}$$

A relação acima indica uma condição suficiente para que a relação de recorrência seja convergente. Ela o será sempre que G', calculado na vizinhança da raiz $\bar{x}$, for menor do que 1, de forma que o erro da iteração $n + 1$ seja menor que o erro da iteração n.

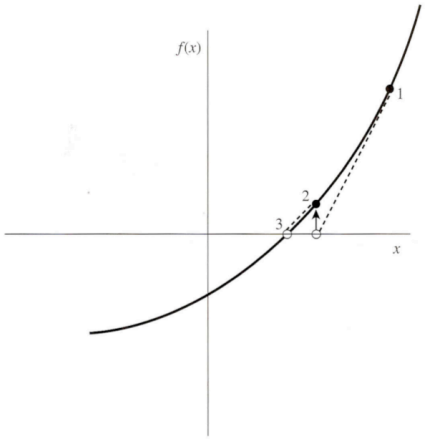


Figura 4.7: O método de Newton.

No exemplo estudado acima, vemos que tanto G_1' quanto G_2' são maiores que 1 próximo à raiz, o que ilustra a limitação da condição acima, que por ser apenas suficiente é útil para garantir a convergência das funções que a satisfazem mas não permite determinar os casos em que o método não converge.

A grande vantagem do método das substituições sucessivas é a sua grande facilidade de implementação, bastando para isso poucas linhas de código.

4.6 Método de Newton, Newton-Raphson ou das Tangentes

O método de Newton guarda alguma relação com o método das substituições sucessivas. Ele consiste em tomar a tangente da função em um ponto x_i (i-ésima estimativa da raiz) e prolongá-la até cruzar o zero. O ponto x_{i+1} onde a tangente cruza o zero passa a ser a nova estimativa da raiz. O método tem forte inspiração geométrica, como mostrado na Figura 4.7.

Vamos obter a relação de recorrência do método de Newton. Algebricamente, o método deriva da expansão em séries de Taylor de uma função na vizinhança de um ponto:

$$f(x+\delta) \approx f(x) + f'(x)\delta + \frac{f''(x)}{2}\delta^2 + \dots . \tag{4.13}$$

Para valores pequenos de δ, e para funções bem-comportadas, os termos quadrático e superiores são pouco importantes. Assim, se $x+\delta$ for uma raiz da função, $f(x+\delta) = 0$, e

$$\delta = -\frac{f(x)}{f'(x)}\,.$$

Dessa forma, a relação de recorrência é

$$x_{i+1} = x_i + \delta\,,$$

ou

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}\,. \tag{4.14}$$

Assim, uma diferença do método de Newton com relação aos anteriores é que ele necessita da primeira derivada da função.

Vamos agora analisar a convergência do método. Podemos escrever a relação de recorrência na forma $x_{i+1} = G(x_i)$, $G(x) \equiv x - f(x)/f'(x)$. De forma geral, podemos escrever o erro como

$$\epsilon_{i+1} = x_{i+1} - \bar{x} = G(x_i) - G(\bar{x}) = G(\bar{x} + \epsilon_i) - G(\bar{x})\,. \tag{4.15}$$

Expandindo $G(\bar{x} + \epsilon_i)$ em série de Taylor até o termo de segunda ordem, temos

$$G(\bar{x} + \epsilon_i) = G(\bar{x}) + \epsilon_i G'(\bar{x}) + \epsilon_i^2 \frac{G''(\bar{x})}{2}\,. \tag{4.16}$$

É fácil ver que a derivada de $G(x)$ em relação a x no ponto $\bar{x}$ é 0. Além disso,

$$G''(\bar{x}) = f''(\bar{x})/f'(\bar{x})\,. \tag{4.17}$$

Substituindo (4.16) and (4.17) em (4.15), temos que

$$\epsilon_{i+1} = -\frac{f''(x)}{2f'(x)}\epsilon_i^2\,, \tag{4.18}$$

ou seja, o método de Newton tem a importante característica de convergir *quadraticamente* para a solução desejada! Próximo a uma raiz, o número de algarismos significativos dobra (aproximadamente) em cada passo. Essa rápida convergência faz do método de Newton a melhor escolha para qualquer função cuja derivada possa ser calculada rapidamente, seja contínua e não nula na vizinhança da função.

Da Eq. (4.18) podemos obter uma condição para a ocorrência de uma convergência quadrática para o método de Newton. As iterações convergirão quando os erros sucessivos forem menores que os anteriores, o que ocorrerá quando

$$\left|\epsilon_0 \frac{f''(x)}{f'(x)}\right| < 2\,, \tag{4.19}$$

onde ϵ_0 é o erro inicial cometido ao estimarmos o primeiro valor da raiz.

O método, apesar de poderoso, tem os seus problemas. Por exemplo, se a estimativa inicial da raiz, x_0, estiver muito longe de $\bar{x}$ e o intervalo entre x_0 e $\bar{x}$ contiver um máximo ou mínimo local da função, o método provavelmente não convergirá. Se uma iteração colocar a estimativa da raiz próxima a esse mínimo ou máximo, a derivada ficará muito pequena e a próxima estimativa será jogada em direção a $-\infty$ ou ∞, com pequenas chances de recuperação. Isso é ilustrado na Figura 4.8. Uma solução simples para esse problema é forçar o confinamento da função em um intervalo, como no método da bissecção. Outro caso problemático para o método está ilustrado na Figura 4.9.

4.6.1 Exemplo

Vamos encontrar a solução da equação $e^{-x} = x$ usando o método de Newton. Temos

$$\begin{aligned} f(x) &= e^{-x} - x \\ f'(x) &= -e^{-x} - 1 \\ f''(x) &= e^{-x}\,. \end{aligned}$$

É fácil verificar que $|f''(x)/f'(x)| < 1$ para todos os valores de x, de forma que, segundo a Eq. (4.19), devemos ter uma convergência quadrática se $\epsilon_0 < 2$.

Aplicando o método para uma valor inicial $x_0 = 0.5$ obtemos a tabela abaixo:

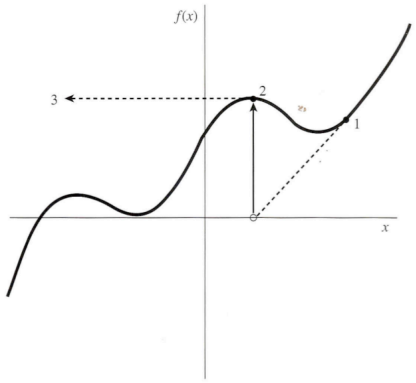


Figura 4.8: Ilustração de um caso em que o método de Newton pode falhar ao passar por um extremo local da função.

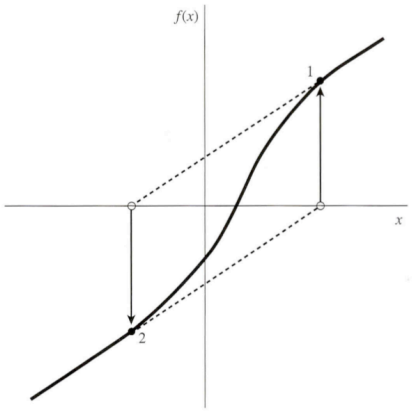


Figura 4.9: Outro caso em que o método de Newton pode falhar ao entrar em um ciclo não convergente.

i	x_i	$f(x)$	$f'(x)$	$\epsilon_i = \lvert x_{i+1} - x_i \rvert / x_i$
0	0.50000000000000	1.07×10^{-1}	-1.60653065971263	—
1	0.56631100319722	1.30×10^{-3}	-1.56761551300324	1.17×10^{-1}
2	0.56714316503486	1.96×10^{-7}	-1.56714336151533	1.47×10^{-3}
3	0.56714329040978	4.44×10^{-15}	-1.56714329040979	2.21×10^{-7}
4	0.56714329040978	-1.11×10^{-16}	-1.56714329040978	5.09×10^{-15}

Foi atingida uma precisão de quase 15 algarismos significativos em apenas 4 iterações!

4.7 Método das secantes

Para funções "suaves" próximo à raiz, o *método das secantes* e o *método da falsa posição* em geral convergem mais rápido que a bissecção.

Vamos inicialmente analisar o método das secantes. Neste método, assume-se que a função seja aproximadamente linear próxima à raiz, e a próxima estimativa da raiz é tomada como o ponto onde a secante que passa pela estimativa atual a pela estimativa

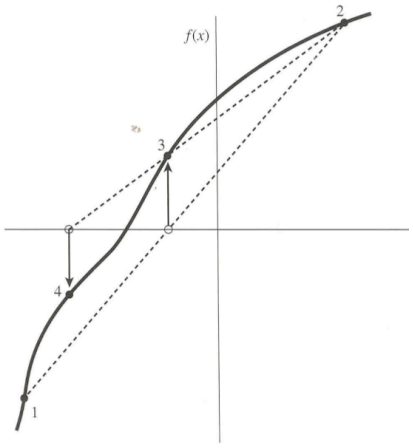


Figura 4.10: O método da secante.

anterior cruza eixo y. O método das secantes está ilustrado graficamente na Figura 4.10. Note que esse método necessita da estimativa de um intervalo $[x_1, x_2]$ inicial, e a primeira estimativa da raiz será o ponto por onde a secante formada pelos pontos $f(x_1)$ e $f(x_2)$ cruza o zero.

A relação de recorrência do método das secantes pode ser facilmente obtida da relação para o método de Newton se substituirmos $f'(x_i)$ por

$$\frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} .$$

Substituindo a expressão acima na Eq. (4.14) obtemos a relação de recorrência para o método das secantes

$$x_{i+1} = x_i - \frac{(x_i - x_{i-1}) f(x_i)}{f(x_i) - f(x_{i-1})} . \tag{4.20}$$

Pode-se mostrar que a ordem da convergência do método das secantes é dada pela *proporção áurea* 1.618, ou seja

$$\lim_{n\to\infty} |\epsilon_{n+1}| \approx \text{constante} \times |\epsilon_n|^{1.618}$$

O método das secantes tem, entretanto a desvantagem de que a função não necessariamente fica confinada no intervalo inicial $[a, b]$. Assim, para funções que não são suficientemente contínuas, não há garantia de que o método convergirá. Apesar da convergência mais lenta, o método das secantes apresenta uma grande vantagem com relação ao método de Newton pois não é necessário calcular a derivada da função, o que pode ser muito vantajoso quando não se conhece a derivada ou ela é muito complicada de ser calculada.

Sob o ponto de vista da eficiência, o método de Newton precisa avaliar duas funções por iteração (f e f'), então sua "ordem de convergência efetiva" é $\sqrt{2} \approx 1.414$. Já no método das secantes só é necessária uma avaliação de função a cada interação. Sob esse aspecto pode-se dizer que o método das secantes é mais eficiente do que o de Newton. Isto é evidenciado em funções muito complicadas, onde teremos um grande número de operações de ponto flutuantes para cada avaliação de função.

4.7.1 Exemplo

Para exemplificar o método das secantes vamos aplicá-lo à mesma função usada como exemplo para o método de Newton: $e^{-x} = x$. Partindo de um intervalo inicial $x_1 = 0.1$ e $x_2 = 1$, que contém a raiz, obtemos uma precisão semelhante à obtida pelo método de Newton com apenas duas iterações à mais:

i	x_i	$f(x)$	$\epsilon_i = \lvert x_{i+1} - x_i \rvert / x_i$
3	0.60408828086464	-5.75×10^{-2}	5.04
4	0.57046746094274	-5.21×10^{-3}	5.57×10^{-2}
5	0.56712120449306	3.46×10^{-5}	5.87×10^{-3}
6	0.56714330368783	-2.08×10^{-8}	3.90×10^{-5}
7	0.56714329040984	-8.32×10^{-14}	2.34×10^{-8}
8	0.56714329040978	0.00	9.36×10^{-14}

A função escolhida é um exemplo de função muito bem comportada. Mesmo partindo de um intervalo que está muito longe da raiz ($x_1 = 10$ e $x_2 = 20$), o método converge, como mostrado abaixo:

i	x_i	$f(x)$	$\epsilon_i = \lvert x_{i+1} - x_i \rvert / x_i$
3	0.00009079738617	1.00	1.00
4	0.90902712762098	-5.06×10^{-1}	1.00×10^{4}
5	0.60355282215108	-5.67×10^{-2}	3.36×10^{-1}
6	0.56502214899899	3.33×10^{-3}	6.38×10^{-2}
7	0.56715719192766	-2.18×10^{-5}	3.78×10^{-3}
8	0.56714329574713	-8.36×10^{-9}	2.45×10^{-5}
9	0.56714329040977	2.10×10^{-14}	9.41×10^{-9}
10	0.56714329040978	-1.11×10^{-16}	2.37×10^{-14}

4.8 Método da falsa posição

O método da falsa posição é uma variação do método das secantes. Nele, a função $f(x)$ é aproximada pela equação da reta que passa pelos extremos do intervalo $[x_1, x_2]$ em que deve existir uma e somente uma raiz da equação da função.

Assim, o método da falsa posição é muito semelhante ao método da bissecção, mas em vez de se determinar o ponto intermédio do intervalo $[x_1, x_2]$, é determinado um ponto x_3, tal que:

$$x_3 = x_1 + \frac{x_1 - x_2}{f(x_2) - f(x_1)} f(x_1)\,. \tag{4.21}$$

O intervalo $[x_1, x_2]$ é então substituído pelo intervalo limitado por x_3 e pelo extremo x_1 ou x_2 em que a função tem sinal contrário a $f(x_3)$. Isso quer dizer que nesse método a raiz está *sempre confinada ao intervalo atual*; portanto, tal como o da bissecção, o método da falsa posição converge sempre. A Figura 4.11 ilustra graficamente o método.

A ordem de convergência do método da falsa posição é difícil de ser estimada, mas ela será certamente menor que no caso das secantes.

4.9 Exercícios

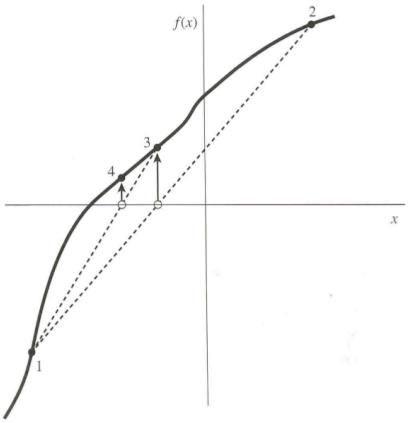


Figura 4.11: O método da falsa posição.

Capítulo 5

Matrizes e Sistemas lineares

Neste capítulo estudaremos alguns métodos para calcular a solução de sistemas de equações lineares. Apenas nos preocuparemos com sistemas quadrados, isto é, aqueles em que o número de equações é igual ao número de incógnitas. Supõe-se que as noções básicas de álgebra matricial, como adição e multiplicação de matrizes, matriz inversa e identidade, determinante de uma matriz etc., sejam conhecidas do leitor.

5.1 Introdução

Um sistema de equações algébricas de ordem n, que é um conjunto de n equações com n incógnitas,

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n & = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n & = b_2 \\ \vdots & \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n & = b_n \end{cases}$$

pode ser representado através de uma equação matricial

$$Ax = b\,,$$

onde

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \text{ é matrix dos coeficientes,}$$

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \text{ é o vetor colunar das incógnitas,}$$

$$b = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} \text{ é o vetor dos termos independentes.}$$

Em todo o texto, salvo menção em contrário, sempre indicaremos um sistema linear genérico de ordem n por $Ax = b$. Para facilidade de notação usaremos indistintamente

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \text{ ou } x = (x_1, \ldots, x_n)\,.$$

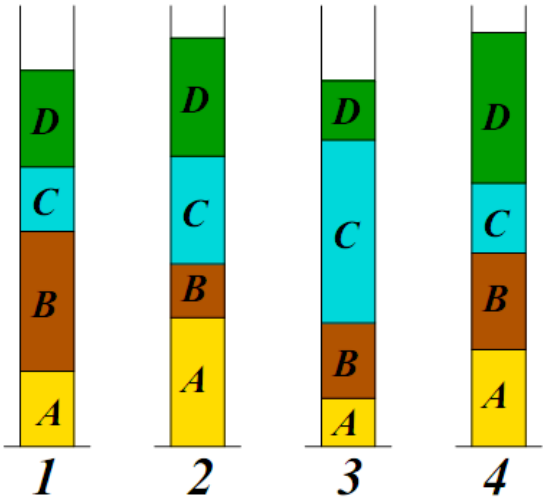
5.2 Exemplos de Aplicação

5.2.1 Provetas [1]

Considere o seguinte problema: quatro tipos de materiais particulados estão distribuídos por quatro provetas, e em cada proveta os materiais são dispostos em camadas, não misturadas, de modo que seja possível medir facilmente o volume de cada material em cada uma delas. Dado que possamos medir a massa total de cada proveta, e que saibamos a massa da proveta vazia, queremos calcular a densidade de cada um dos materiais.

Para colocar o problema em termos matemáticos, chamemos os materiais de A, B, C e D, e suas densidades respectivas de ρ_A, ρ_B, ρ_C e ρ_D. Essas são as *incógnitas* do problema, números que queremos descobrir.

Entre os dados disponíveis para resolvê-lo estão a massa conjunta dos quatro materiais em cada uma das provetas (numeradas de 1 a 4), que chamaremos de m_1, m_2, m_3 e m_4, já descontada a tara das provetas.



Além disso, temos o volume de cada um dos materiais em cada uma das provetas. Chamaremos de v_{1A}, v_{1B}, v_{1C} e v_{1D} o volume dos materiais A, B, C e D na Proveta 1, v_{2A}, v_{2B}, v_{2C} e v_{2D} o volume dos materiais A, B, C e D na Proveta 2, e assim por diante.

Como a densidade é a razão entre massa e volume, a massa do material A na Proveta 1 é $v_1 \times \rho_A$. Estendendo esse raciocínio para os demais materiais, obtemos que a massa total m_1 contida na Proveta 1 é

$$v_{1A} \times \rho_A + v_{1B} \times \rho_B + v_{1C} \times \rho_C + v_{1D} \times \rho_D \,.$$

Considerando as quatro provetas, obteremos quatro equações:

$$\begin{cases} v_{1A} \times \rho_A + v_{1B} \times \rho_B + v_{1C} \times \rho_C + v_{1D} \times \rho_D &= m_1 \\ v_{2A} \times \rho_A + v_{2B} \times \rho_B + v_{2C} \times \rho_C + v_{2D} \times \rho_D &= m_2 \\ v_{3A} \times \rho_A + v_{3B} \times \rho_B + v_{3C} \times \rho_C + v_{3D} \times \rho_D &= m_3 \\ v_{4A} \times \rho_A + v_{4B} \times \rho_B + v_{4C} \times \rho_C + v_{4D} \times \rho_D &= m_4 \end{cases}$$

Trata-se de um sistema linear de quatro equações e quatro incógnitas.

[1] Extraído de Asano & Coli 2009

Uma possível aplicação em geologia seria a seguinte. Uma sonda faz o papel das provetas, e uma coluna de material é retirada, contendo materiais diferentes dispostos em camadas (pode ser até uma sonda coletando material congelado). A sonda permitiria medir a dimensão de cada camada, mas não poderíamos desmanchar a coluna para medir a densidade de cada material isoladamente, sob o risco de alterar a compactação.

5.2.2 Resolução do Círculo

Vamos agora concluir o exemplo iniciado no Capítulo 2. Nosso problema era o seguinte: dadas as coordenadas de três pontos quaisquer, (x_1, y_1), (x_2, y_2), (x_3, y_3), resolver a equação do círculo que passa por estes três pontos,

$$(x-a)^2 + (y-b)^2 = r^2\,,$$

de forma a determinar o centro do círculo (a, b) e seu raio, r. Temos três incógnitas, de forma que são necessárias três equações para resolver o problema. São elas

$$\begin{cases} (x_1-a)^2 + (y_1-b)^2 &= r^2 \\ (x_2-a)^2 + (y_2-b)^2 &= r^2 \\ (x_3-a)^2 + (y_3-b)^2 &= r^2\,. \end{cases}$$

Vamos inicialmente manipular a primeira equação. Expandindo os termos quadráticos obtemos

$$x_1^2 + a^2 - 2ax_1 + y_1^2 + b^2 - 2by_1 - r^2 = 0\,.$$

Definindo

$$k \equiv a^2 + b^2 - r^2$$

obtemos

$$2x_1a + 2y_1b - k = x_1^2 + y_1^2\,.$$

Manipulando as demais equações da mesma forma, obtemos o seguinte sistema de equações lineares

$$\begin{cases} 2x_1a + 2y_1b - k &= x_1^2 + y_1^2 \\ 2x_2a + 2y_2b - k &= x_2^2 + y_2^2 \\ 2x_3a + 2y_3b - k &= x_3^2 + y_3^2\,. \end{cases}$$

que, escrito em forma matricial, fica

$$\begin{bmatrix} 2x_1 & 2y_1 & -1 \\ 2x_2 & 2y_2 & -1 \\ 2x_3 & 2y_3 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \\ k \end{bmatrix} = \begin{bmatrix} x_1^2 + y_1^2 \\ x_2^2 + y_2^2 \\ x_3^2 + y_3^2 \end{bmatrix}\,. \tag{5.1}$$

O problema resume-se, agora, em resolver o sistema acima para obter a, b e k.

5.2.3 Calculando as populações do H em uma região H II

5.3 Método de Cramer

Um método para resolver sistemas lineares, talvez já conhecido do leitor, é o método de Cramer. Nele a solução do sistema $Ax = b$ é dada por

$$x_i = \frac{\det(A_i)}{\det(A)}, i = 1, 2, \ldots, n$$

onde $\det(A)$ é o determinante da matriz A, e A_i é a matriz obtida de A substituindo a sua i-ésima coluna pelo vetor b dos termos independentes.

O determinante de uma matriz A de ordem n pode ser calculado através do desenvolvimento por linhas (regra de Laplace):

$$\det(A) = \sum_{j=1}^{n} (-1)^{i+j} a_{ij} \det(A_{ij})$$

onde i é o índice de uma linha qualquer e A_{ij} é a matriz obtida de A retirando-se a i-ésima linha e a j-ésima coluna.

Observe que se $\det(A) \neq 0$ então o sistema $Ax = b$ tem uma única solução. Se $\det(A) = 0$ então podem ocorrer dois casos:

1. o sistema não possui solução (sistema inconsistente);

2. o sistema possui infinitas soluções (sistema indeterminado).

Por exemplo, no caso de um sistema linear de ordem 2, cada equação representa uma reta. Resolver o sistema significa determinar a intersecção das duas retas. Se as duas retas forem coincidentes, então há infinitos pontos de interseção. Se forem paralelas, não há nenhum ponto de interseção. Neste texto nos preocuparemos com sistemas lineares que tenham uma única solução.

Uma das propriedades do determinante é que se uma das linhas da matriz for uma combinação linear de outra (ou outras), então o determinante será zero. Por exemplo, a matriz abaixo tem determinante zero

$$\begin{bmatrix} 1 & 2 \\ 3 & 6 \end{bmatrix}.$$

Sistemas que possuem equações que são combinações lineares são ditos degenerados ou singulares. Como vimos acima, eles podem ser ou inconsistentes ou indeterminados.

Sistemas não singulares (em que o $\det(A) \neq 0$) possuem sempre uma solução. Entretanto, duas questões numéricas podem impedir que a solução seja obtida

1. embora não sejam combinações lineares exatas de outras, algumas equações podem ser tão próximas a combinações lineares que erros de arredondamento as tornem linearmente dependentes em algum estágio da solução. Neste caso, o procedimento numérico irá falhar.

2. Erros de arredondamento cumulativo podem impedir que a solução seja obtida.

Ao longo deste capítulo discutiremos formas de lidar com estas duas questões.

A utilização do método de Cramer para resolver sistemas lineares pode ser inviável, pois o número de operações aritméticas que devem ser efetuadas aumenta consideravelmente com um pequeno aumento na ordem do sistema.

Para estimar o número de operações necessárias para a regra de Cramer, vamos considerar o caso de um sistema com $n = 20$. Para resolvê-lo, precisamos calcular 21 determinantes de ordem 20. Mas, para calcular um determinante de ordem 20, usamos a regra de Laplace, que decompõe o determinante em uma soma envolvendo 20 determinantes de ordem 19. Se extrapolarmos o processo até chegarmos em determinantes de ordem 2, teremos que o número de operações aritméticas será da ordem de $21! \approx 5 \times 10^{19}$. Para um sistema de ordem n, temos que o número de operações será da ordem de $(n+1)!$.

Em um computador pessoal de 30 Gflops[2] estas 10^{20} operações levariam 3.3×10^9 s ou aproximadamente 100 anos! Na prática, a situação é ainda pior, pois estamos considerando apenas o tempo para efetuar as operações aritméticas, e não o acesso à memória.

No novo super-computador do IAG, que terá uma capacidade teórica de 20 Tflops, esta conta levaria "apenas" 57 dias. Embora útil para sistemas de ordem menor, o método de Cramer é impraticável para sistemas maiores, e outros métodos devem ser empregados neste caso. Outro aspecto negativo do método de Cramer é que como ele necessita de muitas operações aritméticas, ele potencialmente gerará mais erros de arredondamento.

Exercício 1: use a regra de Cramer para obter uma solução analítica para o problema do círculo (Eq. 5.1).

5.4 Tarefas da álgebra linear computacional

Há muito mais na álgebra linear do que resolver um único sistema de equações lineares. Abaixo listamos os principais tópicos abordados neste capítulo.

- Solução para a equação matricial $Ax = b$ para um vetor colunar desconhecido, x.
- Solução para mais de uma de uma equação matricial, $Ax_j = b_j$, para um conjunto de vetores x_j, $j = 0, 1, \ldots$, cada um correspondendo a um dado vetor de termos independentes, b_j. Nesta tarefa a simplificação chave é que a matriz A é mantida constante, enquanto que os termos independentes variam.
- Cálculo da matriz inversa A^{-1}, que obedece à equação matricial $AA^{-1} = I$, onde I é a matriz identidade.
- Cálculo do determinante de uma matriz quadrada A.
- Melhora iterativa da solução de um sistema.

5.5 Sistemas de acordo com as propriedades das matrizes

Tipicamente, podemos ter dois tipos de sistemas lineares, os sistemas *cheios* e *esparsos*. Nos sistemas cheios, todos, ou ao menos a grande maioria, dos elementos da matriz A é diferente de zero. Nos sistemas esparsos, uma parte importante dos elementos de A é nula. Um caso importante são sistemas com matrizes tridiagonais, como ilustrado na Figura 5.1.

Sistemas esparsos possuem soluções particulares e mais rápidas que os sistemas cheios. Vamos inicialmente estudar os métodos de solução para matrizes cheias.

5.6 Método da Eliminação de Gauss

5.6.1 Sobre o método

É o método mais simples para solução de um sistema de equações. O método de Gauss possui várias características que o tornam interessante, mesmo que haja métodos mais eficientes.

Uma característica interessante do método é que quando aplicado para resolver um conjunto de equações lineares, a eliminação de Gauss produz tanto a solução das equações

[2]Flops significa número de operações de ponto flutuante por segundo.

Figura 5.1: Exemplos de matrizes esparsas.

(para um ou mais vetores de termos independentes) quanto a inversa da matriz A (esta última é obtida quando empregamos uma variante do método, chamada de método de Gauss-Jordan, seção 5.7). Uma de suas características mais importantes é que o método é tão estável quanto qualquer outro método direto (direto, aqui, é usado em contraposição aos métodos iterativos mostrados no fim do capítulo), desde que seja empregado o pivotamento (seções 5.6.4 e 5.7.2)

Algumas deficiências do método são

1. se a matriz inversa não for desejada, o método de Gauss é tipicamente 3 vezes mais lento que a melhor alternativa disponível (decomposição LU, seção 5.10).

2. quanto o empregamos para mais de uma equação matricial ($Ax_j = b_j$), todos os vetores de termos independentes devem ser armazenados na memória e manipulados simultaneamente.

A deficiência 1) acima pode suscitar questionamentos, afinal, se temos a matriz inversa, podemos calcular as incógnitas de um sistema $Ax_j = b_j$ através de:

$$x_j = A^{-1} b_j \, .$$

Isto realmente funciona, mas este procedimento resulta em uma resposta muito suscetível a erros de arredondamento, e deve ser evitado.

5.6.2 Procedimento

Vamos ilustrar o procedimento do método de eliminação de Gauss com um exemplo simples. O objetivo consiste em transformar o sistema $Ax = b$ em um sistema triangular equivalente. Para isso, usarmos a seguinte propriedade da Álgebra Linear.

Propriedade: A solução de um sistema linear não se altera se subtrairmos de uma equação outra equação do sistema multiplicada por uma constante.

Considere o seguinte sistema de equações:

$$\begin{cases} 2x + \ y + \ z = \ 7 \\ 4x + 4y + 3z = 21 \\ 6x + 7y + 4z = 32 \end{cases}$$

Multiplicando a primeira equação por (-2) e somando na segunda, e multiplicando a primeira equação por (-3) e somando na terceira temos

$$\begin{cases} 2x + \ y + z = \ 7 \\ \quad\ \ 2y + z = \ 7 \\ \quad\ \ 4y + z = 11 \end{cases}$$

Multiplicando a segunda equação por (-2) e somando na terceira temos

$$\begin{cases} 2x + \ y + z = \ 7 \\ \quad\ \ 2y + z = \ 7 \\ \quad\ \ \ \ -z = -3 \end{cases}$$

Da terceira equação temos $-z = -3 \Rightarrow \boxed{z = 3}$.
Substituindo na segunda equação temos $2y + 3 = 7 \Rightarrow \boxed{y = 2}$.
Substituindo na primeira equação temos $2x + 2 + 3 = 7 \Rightarrow \boxed{x = 1}$.

Vemos que o método de Gauss é uma forma sistemática de triangularizar um sistema linear. A solução é obtida em dois passos:

1. Eliminação (*forward elimination*): triangularização propriamente dita.
2. Substituição (*back substitution*): obtenção da solução final (vetor x).

Se usarmos a notação matricial, estamos resolvendo a equação

$$\begin{bmatrix} 2 & 1 & 1 \\ 4 & 4 & 3 \\ 6 & 7 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 7 \\ 21 \\ 32 \end{bmatrix}$$

transformando-a em

$$\underbrace{\begin{bmatrix} 2 & 1 & 1 \\ & 2 & 1 \\ & & -1 \end{bmatrix}}_{\substack{\text{matriz} \\ \text{triangular superior}}} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 7 \\ 7 \\ -3 \end{bmatrix}.$$

Entretanto, podemos trabalhar somente com os números sem escrever as equações. Para tanto é conveniente escrever a chamada *matriz aumentada*

$$\left[\begin{array}{ccc|c} 2 & 1 & 1 & 7 \\ 4 & 4 & 3 & 21 \\ 6 & 7 & 4 & 32 \end{array}\right].$$

Como antes multiplicamos a primeira equação por 2 e subtraímos da segunda; multiplicamos a primeira equação por 3 e subtraímos da terceira:

$$\left[\begin{array}{ccc|c} 2 & 1 & 1 & 7 \\ 0 & 2 & 1 & 7 \\ 0 & 4 & 1 & 11 \end{array}\right].$$

Então multiplicamos a segunda equação por 2 e subtraímos da terceira

$$\left[\begin{array}{ccc|c} 2 & 1 & 1 & 7 \\ 0 & 2 & 1 & 7 \\ 0 & 0 & -1 & -3 \end{array}\right].$$

5.6.3 Estimativa do número de operações realizadas

Vamos estimar o número de operações realizadas na obtenção da solução x. Estimaremos separadamente o número de operações feitas durante a eliminação e a substituição.

1) Processo de eliminação

Para estimar o número de operações realizadas durante a triangulação da matriz, calcularemos quantas adições e multiplicações são necessárias em cada etapa do processo. Por exemplo, para eliminarmos a primeira coluna, temos $(n-1)$ linhas onde para cada uma delas são calculadas $n+1$ multiplicações e n adições.

eliminação da:	multiplicações	adições
1^{a} coluna	$(n-1)(n+1)$	$(n-1)n$
2^{a} coluna	$(n-2)n$	$(n-2)(n-1)$
$\vdots$	$\vdots$	$\vdots$
$(n-1)^{\text{a}}$ coluna	$(1)(3)$	$(2)(1)$
Total	$\sum_{i=1}^{n-1} i(i+2)$	$\sum_{i=1}^{n-1}(i+1)i$

O total de multiplicações é

$$\sum_{i=1}^{n-1} i(i+2) = \sum_{i=1}^{n-1} i^2 + 2\sum_{i=1}^{n-1} i\,.$$

Avaliando cada uma das somatórias

$$\sum_{i=1}^{n-1} i^2 = \sum_{i=1}^{n} i^2 - n^2 = \frac{n(n+1)(n+2)}{6} - n^2 = \frac{n^3}{3} - \frac{n^2}{2} + \frac{n}{6}$$
$$\sum_{i=1}^{n-1} i = \sum_{i=1}^{n} i - n = \frac{n(n+1)}{2} - n = \frac{n^2}{2} - \frac{n}{2}\,,$$

que implica

$$\sum_{i=1}^{n-1} i^2 + 2\sum_{i=1}^{n-1} i = \frac{n^3}{3} - \frac{n^2}{2} + \frac{n}{6} + 2\left(\frac{n^2}{2} - \frac{n}{2}\right) = \frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6}\,.$$

O número total de adições pode ser obtido de forma análoga:

$$\sum_{i=1}^{n-1}(i+1)i = \frac{n^3}{3} - \frac{n}{3}\,.$$

Obtemos, assim, que o número total de operações de ponto flutuante para o processo de eliminação é

$$N_{\text{elim}} = \frac{2n^3}{3} + \frac{n^2}{2} - \frac{7n}{6}\,.$$

Para um valor de n suficientemente grande, temos que os termos n^3 dominam nas expressões acima, de forma que o total de operações na eliminação será $\mathcal{O}(2n^3/3)$.

2) Processo de substituição

Vamos agora estimar quantas operações de ponto flutuante são feitas durante o cálculo da solução final a partir da matriz triangularizada (*back substitution*).

passo	multiplicações	adições
linha n	1	0
linha $n-1$	2	1
	$\vdots$	$\vdots$
linha 1	n	$n-1$
Total	$\sum_1^n i$	$\sum_1^{n-1} i$

Obtemos que o número de operações para esta fase

$$N_{\text{subst}} = n^2 \,.$$

Chegamos, assim, ao o número total de operações necessárias para resolver um sistema de ordem n pelo método de Gauss

$$N_{\text{Gauss}} = \frac{2n^3}{3} + \frac{3n^2}{2} - \frac{7n}{6} \,.$$

Concluímos que para valores altos de n o processo de eliminação necessita de um número muito maior de operações que a substituição e que, neste caso, o total de operações é

$$N_{\text{Gauss}} \approx \frac{2n^3}{3} \,.$$

Por exemplo, um sistema matricial de 20×20 implica em aproximadamente $2 \cdot 20^3/3 \approx 5 \cdot 10^3$ flop. Com um PC de 30 Gflops o problema será resolvido em

$$t = \frac{5 \cdot 10^3 \,\text{flop}}{30 \cdot 10^9 \,\text{flops}} \approx 2 \cdot 10^{-7} \,\text{s!}$$

Esta estimativa é muito otimista, pois consideramos que cada operação de ponto flutuante é efetuada em um ciclo da CPU. Isto é válido para adições, mas não para multiplicações, que tipicamente requerem da ordem de dez ciclos de CPU. Além disso não consideramos fatores como a perda de eficiência devido ao acesso à memória. De qualquer maneira, vemos que o método de Gauss é imensamente mais eficiente que o método de Cramer.

5.6.4 Pivotamento parcial

Seja o sistema

$$\begin{bmatrix} 10 & -7 & 0 \\ -3 & 2.099 & 6 \\ 5 & -1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 3.901 \\ 6 \end{bmatrix} \tag{5.2}$$

cuja solução é $x = [0, -1, 1]$.

Vamos considerar que nosso operador de ponto flutuante tenha apenas *5 algarismos significativos*, e vamos resolver o sistema acima pelo método de Gauss.

Multiplicando a 1ª equação por 0.3 e somando na 2ª; multiplicando a 1ª equação por -0.5 e somando na 3ª, obtemos

$$\left[\begin{array}{ccc|c} 10 & -7 & 0 & 7 \\ 0 & -0.001 & 6 & 6.001 \\ 0 & 2.5 & 5 & 2.5 \end{array}\right] . \tag{5.3}$$

Multiplicando a 2ª equação por $-2.5/-0.001 = 2500$ e somando na 3ª equação

$$\left[\begin{array}{ccc|c} 10 & -7 & 0 & 7 \\ 0 & -0.001 & 6 & 6.001 \\ 0 & 0 & 15005 & 15004 \end{array}\right] .$$

Note que, devido à restrição de 5 algarismos significativos, tivemos que truncar as seguintes operações

$$\begin{aligned} 6.001 \times 2500 &= 15002.5\not{0} \\ 15002.5\not{0} + 2.5 &= 15004.5\not{0} = 15004 . \end{aligned}$$

Ao efetuarmos a substituição obteremos

$$\begin{aligned} x_3' &= \frac{15004}{15005} = 0.99993 \\ x_2' &= \frac{6.001 - 6 \times 0.99993}{-0.001} = \frac{6.001 - 5.9995\not{8}}{-0.001} = -1.5 \\ x_1' &= \frac{7 + 7 \times (-1.5)}{10} = 7 - 10.510 = -0.35 \end{aligned}$$

Comparando este vetor $x' = (0.99993, -1.5, -0.35)$ obtido com o vetor $x = (1, -1, 0)$ solução, vemos o quão grande foi o erro gerado pela restrição de 5 algarismos significativos!

O que causou este problema? O primeiro elemento da linha que está sendo usada para eliminar os termos das demais é chamado de *pivô*. Na primeira etapa da eliminação acima (Eq. 5.3), o pivô, (-0.001), tornou-se muito pequeno em relação aos outros coeficientes, resultando num enorme multiplicador (2500) que fez aparecerem erros de arredondamento. Estes erros por sua vez são ampliados na fase de substituição, onde apareceram subtrações de números muito próximos divididas por números muito pequenos, o que amplifica enormemente o erro (por exemplo, veja o cálculo de x_2', acima).

Uma solução simples e eficiente para este problema é empregar *pivotamento parcial* no método de Gauss, que consiste em trocar linhas de forma que tenhamos sempre o maior valor absoluto possível para o pivô. Isto garantirá multiplicadores $\lesssim 1$ em módulo.

No exemplo acima, empregamos o pivotamento parcial já na primeira etapa

$$\left[\begin{array}{ccc|c} 10 & -7 & 0 & 7 \\ 0 & -0.001 & 6 & 6.001 \\ 0 & 2.5 & 5 & 2.5 \end{array}\right] \begin{array}{c} \text{pivotamento} \\ \longrightarrow \\ \text{parcial} \end{array} \left[\begin{array}{ccc|c} 10 & -7 & 0 & 7 \\ 0 & 2.5 & 5 & 2.5 \\ 0 & -0.001 & 6 & 6.001 \end{array}\right] .$$

O multiplicador será $(-0.001)/(-2.5) = 0.0004$. Multiplicando a 2ª equação por este valor e somando na 3ª equação, obtemos a matriz estendida

$$\left[\begin{array}{ccc|c} 10 & -7 & 0 & 7 \\ 0 & 2.5 & 5 & 2.5 \\ 0 & 0 & 6.002 & 6.002 \end{array}\right] ,$$

que resulta na solução exata $x' = (1, -1, 0)$.

Uma regra importante a ser seguida: o *pivotamento parcial sempre deve ser empregado no método de Gauss!*

5.6.5 Solução simultânea de várias equações matriciais

Vimos na Seção 5.4 uma tarefa corriqueira da álgebra linear é resolver um conjunto de equações matriciais, $Ax_j = b_j$, $j = 1, \ldots, m$. Neste conjunto as equações matriciais compartilham a matriz A e possuem cada uma um dado dado vetor de termos independentes, b_j. Neste caso, em vez de fazer a mesma eliminação m vezes, podemos "guardar" a sequência de operações aplicadas na triangulação da matriz A para depois aplicar em b_j, $j = 1, \ldots, m$.

Por exemplo, seja a matriz

$$A = \begin{bmatrix} 2 & 6 & -2 \\ 1 & 3 & -4 \\ 3 & 6 & 9 \end{bmatrix}$$

e o vetor de pivotamento que contém o número da linha que foi pivotada,

$$p = \begin{bmatrix} \ \\ \ \\ \ \end{bmatrix}.$$

Com o pivotamento da 3ª linha, temos

$$\begin{bmatrix} 3 & 6 & 9 \\ 1 & 3 & -4 \\ 2 & 6 & -2 \end{bmatrix}; \quad \begin{bmatrix} 3 \\ \ \\ \ \end{bmatrix}.$$

Fazendo 1ª linha $\times (-1/3) +$ 2ª linha e 1ª linha $\times (-2/3) +$ 3ª linha,

$$\begin{bmatrix} 3 & 6 & 9 \\ \boxed{-1/3} & 1 & -7 \\ \boxed{-2/3} & 2 & -8 \end{bmatrix}; \quad \begin{bmatrix} 3 \\ \ \\ \ \end{bmatrix}.$$

Nesta operação, preservamos os *multiplicadores* que foram utilizados para eliminar os primeiros coeficientes das linhas que não eram o pivô. Para concluir a triangularização da matriz, novamente utilizamos o pivotamento da 3ª linha:

$$\begin{bmatrix} 3 & 6 & 9 \\ \boxed{-1/3} & 2 & -8 \\ \boxed{-2/3} & 1 & -7 \end{bmatrix}; \quad \begin{bmatrix} 3 \\ 3 \\ \ \end{bmatrix}.$$

Fazendo 2ª linha $\times (-1/2) +$ 3ª linha,

$$\begin{bmatrix} 3 & 6 & 9 \\ \boxed{-1/3} & 2 & -8 \\ \boxed{-2/3} & \boxed{-1/2} & -3 \end{bmatrix}; \quad \begin{bmatrix} 3 \\ 3 \\ \ \end{bmatrix}.$$

Para ilustrar como utilizar os multiplicadores armazenados e o vedor de pivotamento, vamos encontrar a solução para o vetor $b = [4, -7, 39]$.

- 1º passo: trocar a linha 1 com a linha $p(1) = 3 \rightarrow b = [39, -7, 4]$;
- 2º passo: multiplicar a 1ª linha por $\boxed{-1/3}$ e somar na 2ª linha; multiplicar a 1ª linha por $\boxed{-2/3}$ e somar na 3ª linha, $\rightarrow b = [39, -20, -22]$;

- 3º passo: trocar a linha 2 com a linha $p(2) = 3 \to b = [39, -22, -20]$;

- 4º passo: Multiplicar a 2ª linha por $\boxed{-1/2}$ e somar na 3ª linha, $b = [39, -22, -9]$.

Assim, o vetor x com a solução do sistema será dado por

$$\begin{bmatrix} 3 & 6 & 9 \\ 0 & 2 & -8 \\ 0 & 0 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 39 \\ -22 \\ -9 \end{bmatrix},$$

que pode ser facilmente resolvido por substituição

$$\begin{aligned} x_3 &= -9/(-3) = 3 \\ x_2 &= [-22 + 8 \times 3]/2 = 1 \\ x_1 &= [39 - 6 \times 1 - 9 \times 3]/3 = 2\,. \end{aligned}$$

.

O procedimento ilustrado pode ser repetido para um número arbitrário de vetores b_j. Uma sugestão para uma implementação eficiente do método de Gauss é fazer uma subrotina para a eliminação, que retorna a matriz triangularizada com os coeficientes de eliminação, segundo procedimento acima, e outra para a substituição.

Abaixo delineamos um possível algoritmo para implementar a eliminação de Gauss computacionalmente, mantendo os multiplicadores para uso posterior:

- de $i = 1$ até $n - 1$ faça
  - determine o índice do pivoteamento $l \geq i$
  - troque as linhas i e l, das colunas i até n
  - registre o i-ésimo pivotamento: $p(i) = l$
  - laço
    - de $j = i + 1$ até n faça
    - calcule o multiplicador para a linha j
    - guarde-o no lugar do elementos eliminado
    - adicione múltiplos da linha l à linha j
  - fim do laço

5.6.6 Cálculo do determinante de uma matriz A

Pelas propriedades do determinante, o determinante não se altera se somarmos um múltiplo de uma linha da matriz à outra, ou seja, se efetuarmos uma combinação linear entre as linhas. Assim, a eliminação de Gauss para se obter uma matriz triangular superior não afeta o valor do determinante

$$\det \underbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}}_{A} = \det \underbrace{\begin{bmatrix} a'_{11} & a'_{12} & \cdots & a'_{1n} \\ 0 & a'_{22} & \cdots & a'_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & a'_{nn} \end{bmatrix}}_{U}$$

Mas o determinante de uma matriz triangular é o produto dos elementos da matriz. Portanto,

$$\det A = \det U = a'_{11} a'_{22} \ldots h a'_{nn}\,.$$

Atenção: cada operação de pivotamento troca o sinal do determinante! Assim, se para implementar a eliminação de Gauss foram realizados n_p pivotamentos, o determinante será

$$\det A = (-1)^{n_p} a'_{11} a'_{22} \dots a a'_{nn} \, .$$

Exemplo: calcule o determinante da matriz abaixo usando o método de Gauss:

$$A = \begin{bmatrix} 2 & 1 & 1 \\ 4 & 4 & 3 \\ 6 & 7 & 4 \end{bmatrix} .$$

Usando a eliminação de Gauss sem pivotamento, obtemos

$$U = \begin{bmatrix} 2 & 1 & 1 \\ 0 & 2 & 1 \\ 0 & 0 & -1 \end{bmatrix} \rightarrow \det A = \det U = 2 \times 2 \times (-1) = -4 \, .$$

Exercício 2: Calcule, usando o método de Gauss com pivotamento parcial a solução do sistema

$$\begin{bmatrix} 4 & 3 & 2 & 2 \\ 2 & 1 & 1 & 2 \\ 2 & 2 & 2 & 4 \\ 6 & 1 & 1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 8 \\ 3 \\ 1 \end{bmatrix}$$

5.7 Método de Gauss-Jordan

Este método é uma variante do método de Gauss, onde são eliminados todos os elementos acima e abaixo do pivô. O resultado da eliminação de Gauss-Jordan é uma matriz diagonal. Para ilustrar método, vamos aplicá-lo à solução de um sistema $Ax = b$ e à obtenção simultânea da matriz inversa de A.

Considere a equação matricial

$$A \cdot [x_1 \vee x_2 \vee Y] = [b1 \vee b2 \vee I] \, . \tag{5.4}$$

onde A e Y são matrizes quadradas, x_i e b_i são vetores colunares, I é a matriz identidade, o operador $(\cdot)$ significa um produto de matrizes e o operador $\vee$ significa o aumento de matriz, ou seja, a remoção dos parênteses das matrizes para fazer uma matriz mais larga. É fácil perceber, da equação acima, que os x_1 e x_2 são simplesmente a solução das equações matriciais

$$A \cdot x_1 = b_1 \, ,$$
$$A \cdot x_2 = b_2 \, ,$$

e que a matriz Y é a inversa de A, ou seja

$$A \cdot Y = I \, .$$

Para simplificar, mas sem perda de generalidade, vamos podemos escrever explicitamente a Eq. (5.4) usando matrizes de ordem 3

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \cdot \left\{ \begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \end{bmatrix} \vee \begin{bmatrix} x_{12} \\ x_{22} \\ x_{32} \end{bmatrix} \vee \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & y_{32} & y_{33} \end{bmatrix} \right\} = \left\{ \begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \end{bmatrix} \vee \begin{bmatrix} b_{12} \\ b_{22} \\ b_{32} \end{bmatrix} \vee \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right\}$$

Usando a notação de matrizes aumentada, o sistema acima fica

$$\left[\begin{array}{ccc|c|c|ccc} a_{11} & a_{12} & a_{13} & b_{11} & b_{12} & 1 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & b_{21} & b_{22} & 0 & 1 & 0 \\ a_{31} & a_{32} & a_{33} & b_{31} & b_{32} & 0 & 0 & 1 \end{array}\right]$$

5.7.1 Exemplo de aplicação: inversão de matrizes

$$A \cdot [x \vee Y] = [b \vee I]\,. \tag{5.5}$$

$$\left[\begin{array}{ccc|c|ccc} a_{11} & a_{12} & a_{13} & b_1 & 1 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & b_2 & 0 & 1 & 0 \\ a_{31} & a_{32} & a_{33} & b_3 & 0 & 0 & 1 \end{array}\right]$$

$$\left[\begin{array}{ccc|c|ccc} 2 & 1 & 1 & 7 & 1 & 0 & 0 \\ 4 & 4 & 3 & 21 & 0 & 1 & 0 \\ 6 & 7 & 4 & 32 & 0 & 0 & 1 \end{array}\right]$$

$$\left[\begin{array}{ccc|c|ccc} 2 & 1 & 1 & 7 & 1 & 0 & 0 \\ 0 & 2 & 1 & 7 & -2 & 1 & 0 \\ 0 & 4 & 1 & 21 & -3 & 0 & 1 \end{array}\right]$$

A partir daqui, elimina-se os elementos superiores e inferiores ao pivô:

$$\left[\begin{array}{ccc|c|ccc} 2 & 0 & 1/2 & 7/2 & 2 & -1/2 & 0 \\ 0 & 2 & 1 & 7 & -2 & 1 & 0 \\ 0 & 0 & -1 & -3 & 1 & -2 & 1 \end{array}\right]$$

$$\left[\begin{array}{ccc|c|ccc} 2 & 0 & 0 & 2 & 5/2 & -3/2 & 1/2 \\ 0 & 2 & 0 & 4 & -1 & -1 & 1 \\ 0 & 0 & -1 & -3 & 1 & -2 & 1 \end{array}\right]$$

Finalmente, normaliza-se a matriz, de forma que à esquerda ficamos com uma matriz identidade. Obtém-se, assim, o vetor solução do problema e a matriz inversa de A.

$$\underbrace{\left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right]}_{\equiv X} \underbrace{\begin{array}{c} 1 \\ 2 \\ 3 \end{array}}_{\equiv X} \underbrace{\left[\begin{array}{ccc} 5/4 & -3/4 & 1/4 \\ -1/2 & -3/2 & 1/2 \\ -1 & 2 & -1 \end{array}\right]}_{\equiv A^{-1}}\,.$$

5.7.2 Pivotamento total

Vimos acima (seção 5.6.4) um exemplo em que o pivotamento parcial foi usado para evitar erros de arredondamento que podem aparecer quando o multiplicador fica muito pequeno. Partindo do princípio que os erros de arredondamento são tão menores quanto maiores forem os multiplicadores, em geral obtém-se melhores resultados empregando-se o *pivotamento total*, em que se pivotam colunas, além das linhas, de forma a sempre manter o maior termo de uma data linha como pivô.

O pivotalmento total pode ser empregado devido à seguinte propriedade da Álgebra Linear:

Propriedade: a solução de um sistema linear não é alterada quando trocamos de lugar duas colunas i e j de A, *desde que se troquem as duas linhas correspondentes nos vetores x e na matriz Y* (Eq. 5.5).

Desta forma, vemos que as operações de troca de colunas embaralham o(s) vetor(es) das incógnitas e a matriz inversa, de forma que para empregar o pivotamento total devemos manter um registro das trocas de colunas efetuadas para podermos colocar a solução final na ordem correta. Para exemplificar, vamos resolver novamente o sistema da Eq. (5.2). Após a eliminação da primeira coluna, obtemos

$$\left[\begin{array}{ccc|c} 10 & -7 & 0 & 7 \\ 0 & -0.001 & 6 & 6.001 \\ 0 & 2.5 & 5 & 2.5 \end{array}\right] .$$

Vamos agora trocar de lugar as colunas 2 e 3, de forma que o coeficiente 6 será o pivô

$$\left[\begin{array}{ccc|c} 10 & 0 & -7 & 7 \\ 0 & 6 & -0.001 & 6.001 \\ 0 & 5 & 2.5 & 2.5 \end{array}\right] .$$

Eliminando o segundo termo da terceira linha, obtemos

$$\left[\begin{array}{ccc|c} 10 & 0 & -7 & 7 \\ 0 & 6 & -0.001 & 6.001 \\ 0 & 0 & 2.50083 & 2.50083 \end{array}\right] ,$$

que pode ser facilmente resolvido para obtermos $x_p = (0, 1, -1)$. Como fizemos uma troca de colunas, a solução final é obtida trocando-se de lugar as linhas 2 e 3 de x_p, ou seja, $x = (0, -1, 1)$.

5.8 Refinamento da Solução

Seja o sistema

$$Ax = b\,. \tag{5.6}$$

Resolvendo por Gauss ou Gauss-Jordan, obtemos a solução $x^{(0)}$. Sabemos que erros de arredondamento podem ocorrer quando se resolve um sistema linear pelo método de eliminação, podendo comprometer o resultado obtido. Mesmo utilizando pivoteamento total, não se pode assegurar que a solução obtida seja exata.

Inicialmente, notemos que é trivial verificarmos se a solução de um sistema está correta, para isso basta multiplicarmos a matriz A pela solução obtida, $x^{(0)}$, e o resultado deve ser b. Numericamente, esta verificação deve ser feita impondo-se um critério de convergência do tipo:

$$\left| \frac{b_i^{(0)} - b_i}{b_i} \right| < \epsilon, i = 1, \ldots, n\,,$$

onde $b^{(0)}$ é o vetor obtido do produto $Ax^{(0)}$.

O que fazemos quando o resultado obtido $x^{(0)}$ não passa pelo critério de convergência? Uma possibilidade é fazermos o refinamento da solução, como delineado a seguir. Vamos chamar de erro a diferença entre o valor verdadeiro, x, e o valor obtido, $e^{(0)} = x - x^{(0)}$. Substituindo no sistema (5.6), temos

$$A(x^{(0)} + e^{(0)}) = b \tag{5.7}$$

$$Ae^{(0)} = b - Ax^{(0)} \equiv r^{(0)}\,. \tag{5.8}$$

Resolvendo o sistema (5.8) determinamos $e^{(0)}$, a partir do qual podemos fazer uma nova estimativa para a solução: $x^{(1)} = x^{(0)} + e^{(0)}$. Caso $x^{(1)}$ obedeça ao critério de convergência estipulado, teremos encontrado nossa solução. Caso contrário, podemos refinar novamente a solução obtendo uma estimativa para o erro de $x^{(1)}$ resolvendo o sistema

$$Ae^{(1)} = b - Ax^{(1)} \equiv r^{(1)} \,. \tag{5.9}$$

Este processo pode ser executado quantas vezes desejarmos. É fundamental que as operações envolvidas nos cálculos dos resíduos sejam feitas em dupla precisão.

Importante: Como o refinamento envolve a resolução de vários sistemas que compartilham a mesma matriz A, podemos empregar o procedimento descrito na seção 5.6.5 para tornar o processo mais eficiente.

5.9 Sistemas mal-condicionados

Estes sistemas também são conhecidos pelo termo mal-condicionados (*"ill conditioned"*, em inglês). Vejamos um exemplo.

$$\begin{cases} x + \ y = \ 1 \\ 99x + 100y = 99.5 \end{cases}$$

A solução deste sistema é única e exata: $x = 0.5$, $y = 0.5$. Agora considere o sistema

$$\begin{cases} x + \ y = \ 1 \\ 99.4x + 99.9y = 99.2 \end{cases} \,,$$

cuja solução única e exata é $x = 1.4$, $y = -0.4$, muito diferente da do sistema anterior, apesar dos coeficientes serem parecidos.

Graficando as retas no plano (x, y) vemos porque isto acontece: as retas correspondentes às equações são quase paralelas, ou seja, as equações são quase linearmente dependentes.

Uma maneira de se "medir" o condicionamento de uma traz é calcular seu determinante. Entretanto, o valor do determinante depende da norma (módulo) de cada um dos seus vetores. Assim, devemos normalizar os vetores para depois calcular o determinante. Isto produzirá um determinante com valor (em módulo) no intervalo $[0, 1]$. Se o módulo do determinante for próximo de zero, então o sistema é mal-condicionado.

Por exemplo, vamos considerar o primeiro exemplo acima. Normalizando os vetores da matriz

$$\begin{bmatrix} 1 & 1 \\ 99 & 100 \end{bmatrix} \,,$$

obtemos

$$\begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 99/140.716 & 100/140.716 \end{bmatrix} \,.$$

O módulo do determinante da matriz normalizada é

$$\left| \frac{1}{\sqrt{2}} \frac{100}{140.716} - \frac{1}{\sqrt{2}} \frac{99}{140.716} \right| \approx 5 \times 10^{-3} \,,$$

o que demonstra, quantitativamente, que a matriz original é mal-condicionada.

Há outras medidas do condicionamento de uma matriz, assim como há fórmulas que relacionam o erro cometido no método de Gauss ou Gauss-Jordan com essas medidas e o número de algarismos significativos utilizado. Isto, porém, está além do escopo destas notas. Veja, por exemplo, a seção sobre *singular value decomposition* no Numerical Recipes.

5.10 Decomposição LU

Suponhamos que se possa escrever a matriz A como o produto de duas matrizes

$$A = L \cdot U \,, \tag{5.10}$$

onde L é uma matriz triangular inferior (tem elementos somente na diagonal e abaixo) e U é uma matriz diagonal superior (com elementos somente na diagonal e acima). Para o caso de uma matriz 4×4, Eq. (5.10) ficaria

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \underbrace{\begin{bmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{bmatrix}}_{\substack{L \\ \text{triangular} \\ \text{inferior}}} \underbrace{\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix}}_{\substack{U \\ \text{triangular} \\ \text{superior}}} .$$

Pode-se usar esta decomposição para resolver o conjunto de equações lineares

$$Ax = (LU)x = L(Ux) = b \,.$$

Inicialmente resolvemos para o vetor y tal que

$$Ly = b \tag{5.11}$$

e depois resolvemos

$$Ux = y \,, \tag{5.12}$$

para obter a solução final.

Qual a vantagem em quebrarmos um sistema em dois sistemas sucessivos? A vantagem é que a solução de um sistema triangular é trivial. Dessa forma, o sistema (5.11) é resolvido por substituição para frente:

$$y_1 = \frac{b_1}{l_{11}} \tag{5.13}$$

$$y_i = \frac{1}{l_{ii}} \left[b_i - \sum_{j=1}^{i} l_{ij} y_i \right] , \; i = 2, 3, \ldots, n \,, \tag{5.14}$$

e o sistema (5.12) por substituição para trás:

$$x_n = \frac{y_n}{u_{nn}} \tag{5.15}$$

$$x_i = \frac{1}{u_{ii}} \left[y_i - \sum_{j=i+1}^{n} u_{ij} x_j \right] , \; i = n-1, n-2, \ldots, 1 \,. \tag{5.16}$$

5.10.1 Efetuando a Decomposição LU

Como podemos achar L e U dado A? Abaixo vamos delinear um algoritimo bastante utilizado, que pode ser estudado em mais detalhes no Numerical Recipes. Vamos escrever explicitamente o componente $i.j$ da Eq. (5.10). Este componente é sempre uma soma que começa co,

$$l_{i1}u_{1j} + \cdots = a_{ij}\,.$$

O número de termos da soma depende se $i < j$, $i > j$, ou $i = j$. De fato, temos os três casos acima

$$i < j : l_{i1}u_{1j} + l_{i2}u_{2j} + \cdots + l_{ii}u_{ij} = a_{ij} \tag{5.17}$$

$$i = j : l_{i1}u_{1j} + l_{i2}u_{2j} + \cdots + l_{ii}u_{jj} = a_{ij} \tag{5.18}$$

$$i > j : l_{i1}u_{1j} + l_{i2}u_{2j} + \cdots + l_{ij}u_{jj} = a_{ij} \tag{5.19}$$

As Eqs. (5.17) — (5.19) perfazem n^2 equações para $n^2 + n$ incógnitas (note que a diagonal está representada duas vezes). Trata-se, assim, de um sistema indeterminado. Para resolver este sistema, deve-se, assim, *especificar arbitrariamente valores para n incógnitas*. Um procedimento muito usado para resolver a decomposição é o Algoritmo de Crout, que resolve de forma trivial as equações acima para todos os l's e u's simplesmente rearranjando as equações em determinada ordem. O algoritmo é como se segue:

- Faça l_{ii}, $i = 1, \cdots, n$ (de forma a reduzir o número de incógnitas para n^2);
- Para cada $j = 1, \cdots, n$, faça os dois procedimentos seguintes:
  - Primeiramente, para $i = 1, \cdots, j$, use Eqs. (5.17) e (5.18), e a condição acima, para determinar os u_{ij}, ou seja

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}\,. \tag{5.20}$$

    Quando $i = 1$ a soma anterior é tomada como zero.
  - Em segundo lugar, para $i = j+1, \cdots, n$ use (5.17) para achar os l_{ij}, da seguinte maneira

$$l_{ij} = \frac{1}{u_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj}\right)\,. \tag{5.21}$$

Certifique-se de executar ambos os procedimentos antes de passar para o próximo j!

A chave para compreender o procedimento acima é que os l's e u's que ocorrem no lado direito das equações (5.20) e (5.21) já estão sempre determinados no momento em que são necessários (por isso que o método de Crout é basicamente uma ordem em que as equações devem ser resolvidas). Vemos, também, que cada a_{ij} é usado apenas uma vez durante o processo. Isso significa que os l's e u's podem ser armazenados nos lugares que os termos a's ocupavam na memória do computador. Ou seja, o método de Crout substitui a matriz original A por uma matriz combinada dos elementos de L e U:

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ l_{21} & u_{22} & u_{23} & u_{24} \\ l_{31} & l_{32} & u_{33} & u_{34} \\ l_{41} & l_{42} & l_{43} & u_{44} \end{bmatrix}\,.$$

O pivotamento, tal como no caso dos métodos de Gauss e Gauss-Jordan, é essencial para a estabilidade do método de Crout. Quando se emprega o pivotamento parcial, na realidade não se decompõe a matriz A na sua forma LU, mas sim uma permutação das linhas de A. Para ver como efetuar o pivotamento no método de Crout, consulte o capítulo 2 do Numerical Recipes.

Qual a vantagem da Decomposição LU sobre o método de Gauss? Como listado na seção 5.12, o número de operações necessárias para efetuar a decomposição é da ordem de $1/3\,n^3$, exatamente o mesmo número de passos necessários para fazer a eliminação de Gauss. Na literatura, frequentemente cita-se uma vantagem da Decomposição LU que é o fato de que uma vez tendo-se L e U é trivial obter a solução para um número arbitrário de vetores de termos independentes (ou seja, resolve-se facilmente um conjunto de sistema de equações lineares). Entretanto, o mesmo procedimento pode ser feito de forma igualmente eficiente à partir do procedimento delineado na seção 5.6.5.

Conclusão: o método de Gauss e o método da Decomposição LU são igualmente eficientes quando se trata de resolver um sistema de equações lineares, ou um conjunto de sistemas de equações lineares.

5.10.2 Um caso especial: decomposição LU de matrizes tridiagonais

Um caso particular em que a decomposição LU oferece uma solução eficiente é no caso de matrizes tridiagonais. Suponha que A seja uma matriz na forma

$$A = \begin{bmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & a_n & b_n \end{bmatrix},$$

ou seja, apenas os elementos da diagonal principal e das diagonais imediatamente acima e abaixo são não nulos. Neste caso, a decomposição LU então tem uma forma simples

$$L = \begin{bmatrix} 1 & & & & \\ l_2 & 1 & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & l_n & 1 \end{bmatrix},$$

$$U = \begin{bmatrix} u_1 & v_1 & & & \\ & u_2 & v_2 & & \\ & & \ddots & \ddots & \\ & & & \ddots & v_{n-1} \\ & & & & u_n \end{bmatrix}.$$

É fácil demonstrar que a determinação dos l's, u's e v's é feita através da seguintes relações de recorrência:

$$\begin{cases} u_1 = b_1 \\ l_j = a_j/u_{j-1} \\ u_j = b_j - l_j c_{j-1},\ j = 2, 3, \ldots, n \end{cases}$$

Note que nas relações acima está implícito que $v_i = c_i$.

Tendo determinado as matrizes L e U através das relações acima, o procedimento para determinar o vetor solução x do sistema $Ax = d$ (note que aqui usamos d para evitar confusão com os b's da matriz tridiagonal) é simples. Inicialmente calcula-se a solução do sistema $Ly = d$ através da substituição para frente:

$$\begin{cases} y_1 = d_1 \\ y_i = d_i - l_i y_{i-1}, \ i = 2, \ldots, n \end{cases}$$

Calcula-se, então, o vetor solução x resolvendo-se o sistema $Ux = y$ por substituição para trás:

$$\begin{cases} x_n = \frac{y_n}{u_n} \\ x_k = y_k - c_k \frac{x_{k+1}}{u_k}, \ k = n-1, \ldots, 1 \end{cases}$$

O procedimento descrito acima é muito eficiente do ponto de vista computacional e pode ser implementado com facilidade em duas subrotinas, uma para o cálculo da decomposição e outra para a solução do sistema. Note que o fato de que a decomposição LU de uma matriz tridiagonal também é tridiagonal simplifica muito as substituições para frente e para trás. Veremos no Cap. 6 que esta solução para um sistema tridiagonal será muito útil para calcular os coeficientes da interpolação por spline cúbica.

5.11 Forma alternativa para o cálculo da matriz inversa

Denotemos a matriz inversa de A por B, tal que:

$$AB = I \,,$$

onde I é a matriz identidade. Como usar a decomposição LU ou o método descrito na seção (5.6.5) para encontrar B? Isso é feito simplesmente escrevendo a equação matricial acima para cada uma das colunas de B, ou seja,

$$A \begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \\ b_{41} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} ,$$

$$A \begin{bmatrix} b_{12} \\ b_{22} \\ b_{32} \\ b_{42} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} ,$$

e assim por diante. Ou seja, o cálculo da matriz inversa reduz-se a resolver um conjunto de n sistemas lineares em que os vetores de termos independentes são as diferentes colunas da matriz identidade.

5.12 Comparando Gauss, Gauss-Jordan, e Decomposição LU

Concluímos esta parte do capítulo comparando a eficiência dos três métodos diretos estudados. Temos no quadro abaixo a comparação do número de operações empregadas em cada método para as diferentes tarefas da álgebra linear.

	método	operações
Solução de	Gauss	$1/3\,n^3$
Sistemas	Gauss-Jordan	$1/2\,n^3$
Lineares	LU	$1/3\,n^3$
Inversão	Gauss	$5/6\,n^3$
de	Gauss-Jordan	n^3
Matriz	LU	$5/6\,n^3$
m	Gauss	$\frac{1}{3}n^3 + \frac{1}{2}mn^2$
lados	Gauss-Jordan	$\frac{1}{2}n^3 + mn^2$
direitos	LU	$\frac{1}{3}n^3 + \frac{1}{2}mn^2$

5.13 Métodos Iterativos

Os métodos que vimos até agora (Gauss, Gauss-Jordan, Decomposição LU) são conhecidos como métodos diretos, pois a solução é obtida através da manipulação direta das equações do sistema. Tais métodos podem se tornar ineficientes quando o número de equações fica muito grande ($n \gtrsim 100$), pois o número de operações de ponto-flutuante é $\mathcal{O}(n^3)$. Mais detalhes em Blum(1972), p.131.

Nos métodos ditos iterativos (também chamados de métodos indiretos), arbitra-se um vetor inicial $x^{(0)}$ para a solução e calcula-se uma nova estimativa da solução, $x^{(1)}$ como função de $x^{(0)}$ e assim sucessivamente, ou seja,

$$x^{k+1} = g(x^k)\,,$$

onde k é a k-ésima iteração e g representa uma função qualquer. O processo é repetido até obter a precisão desejada, que se traduz em uma diferença muito pequena entre x^{k+1} e x^k.

Nota: não confunda métodos iterativos com a melhora iterativa da solução, apresentada na seção 5.8.

5.13.1 Método de Jacobi

Seja um sistema linear de ordem n

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = & b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = & b_2 \\ \vdots & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = & b_n \end{cases}$$

Podemos reescrevê-lo na seguinte forma

$$\begin{cases} x_1 &= \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3 - \ldots - a_{1n}x_n) \\ x_2 &= \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3 - \ldots - a_{2n}x_n) \\ \vdots & \\ x_n &= \frac{1}{a_{nn}}(b_n - a_{n1}x_1 - a_{n2}x_2 - \ldots - a_{n,n-1}x_{n-1}) \end{cases}$$

No método de Jacobi, escolhemos arbitrariamente um vetor inicial $x^{(0)}$ e substituímos no lado direito das equações acima obtendo um novo vetor $x^{(1)}$. Repetindo-se o processo k vezes, vemos que a k-ésima estimativa da solução é obtida da seguinte relação de recorrência:

$$\begin{cases} x_1^{(k+1)} & = \frac{1}{a_{11}}(b_1 - a_{12}x_2^{(k)} - \ldots - a_{1n}x_n^{(k)}) \\ x_2^{(k+1)} & = \frac{1}{a_{22}}(b_2 - a_{21}x_1^{(k)} - \ldots - a_{2n}x_n^{(k)}) \\ \vdots & \\ x_n^{(k+1)} & = \frac{1}{a_{nn}}(b_n - a_{n1}x_1^{(k)} - \ldots - a_{n,n-1}x_{n-1}^{(k)}) \end{cases}$$

ou, de forma mais compacta,

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^{(k)} \right) .$$

Dizemos que o processo iterativo converge se, para a sequência de aproximações gerada, dado $\epsilon > 0$, existir um j tal que para todo $k > j$ e $i = 1, 2, \ldots, n$, $|x_i^k - \bar{x}_i \leq \epsilon$, onde $\bar{x}$ é a solução do sistema. Como na prática não a conhecemos, torna-se necessário um critério de parada para o processo iterativo. Um possível critério é impor que a variação relativa entre duas aproximações consecutivas seja menor que ϵ. Dado $x^{(k+1)}$ e $x^{(k)}$, tal condição é escrita como

$$\max \left\{ \left| \frac{x_i^{(k+1)} - x_i^{(k)}}{x_i^{(k)}} \right|, i = 1, \ldots, n \right\} \leq \epsilon \, . \tag{5.22}$$

Exemplo: considere o seguinte sistema de equações

$$\begin{cases} 4x_1 + 2x_2 + x_3 & = 11 \\ -x_1 + 2x_2 & = 3 \\ 2x_1 + x_2 + 4x_3 & = 16 \end{cases} ,$$

cuja solução é $x = (1, 2, 3)$. Rescrevendo as equações como

$$\begin{cases} x_1 & = \frac{11}{4} - \frac{1}{2}x_2 - \frac{x_3}{4} \\ x_2 & = \frac{3}{2} + \frac{1}{2}x_1 \\ x_3 & = 4 - \frac{1}{2}x_1 - \frac{1}{4}x_2 \end{cases} ,$$

temos que as relações de recorrência, pelo método de Jacobi, são

$$\begin{aligned} x_1^{(k+1)} &= \frac{11}{4} - \frac{1}{2}x_2^{(k)} - \frac{x_3^{(k)}}{4} \, , \\ x_2^{(k+1)} &= \frac{3}{2} + \frac{1}{2}x_1^{(k)} \, , \\ x_3^{(k+1)} &= 4 - \frac{1}{2}x_1^{(k)} - \frac{1}{4}x_2^{(k)} \, . \end{aligned} \tag{5.23}$$

Começando com um vetor arbitrário $x^{(0)} = [1, 1, 1]$ obtemos

$$\begin{aligned} x_1^{(1)} &= \frac{11}{4} - \frac{1}{2} \cdot 1 - \frac{1}{4} \cdot 1 = 2 \\ x_2^{(1)} &= \frac{3}{2} + \frac{1}{2} \cdot 1 = 2 \\ x_3^{(1)} &= 4 - \frac{1}{2}.1 - \frac{1}{4} \cdot 1 = \frac{13}{4} \end{aligned}$$

Substituindo $x^{(1)}$ do lado direito do sistema (5.26) obtemos

$$x_1^{(2)} = \frac{11}{4} - \frac{1}{2}\cdot 2 - \frac{1}{4}\cdot\frac{13}{4} = \frac{15}{16}$$
$$x_2^{(2)} = \frac{3}{2} + \frac{1}{2}\cdot 2 = \frac{5}{2}$$
$$x_3^{(2)} = 4 - \frac{1}{2}.2 - \frac{1}{4}\cdot 2 = \frac{5}{2}$$

Na tabela abaixo listamos os resultados para as 5 primeiras iterações. Vemos que a sequência converge, e atinge uma precisão de aproximadamente 5% em 5 iterações.

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$\max\{\lvert(x_i^{(k)} - x_i^{(k-1)})/x_i^{(k)}\rvert, i = 1,\ldots,n\}$
0	1	1	1	-
1	2	2	13/4	9/13
2	15/16	5/2	5/2	3/10
3	7/8	63/62	93/32	17/63
4	133/128	31/16	393/128	7/131
5	519/512	517/256	767/256	21/517

5.13.2 Convergência do Método de Jacobi

(REVISAR)

Vamos escrever a matriz A como

$$A = L + D + U$$

onde L é a *"lower triangular matrix"* (sem diagonal); U *"upper triangular matrix"* (sem diagonal); e D a matriz diagonal.

Desta forma,

$$Ax = (L + D + U)x = b$$
$$Dx = -(L + U)x + b$$
$$x = D^{-1}[-(L + U)x + b]$$
$$x = Jx + c$$

onde $J = -D^{-1}(L + U)$ e $c = D^{-1}b$. Aplicando o método iterativo teremos

$$x^{(k+1)} = Jx^{(k)} + c, \text{ onde } J = -\begin{bmatrix} 0 & \frac{a_{12}}{a_{11}} & \frac{a_{13}}{a_{11}} & \cdots & \frac{a_{1n}}{a_{11}} \\ \frac{a_{21}}{a_{22}} & 0 & \frac{a_{23}}{a_{22}} & \cdots & \frac{a_{2n}}{a_{22}} \\ \vdots & & 0 & & \vdots \\ \vdots & & & 0 & \vdots \\ \frac{a_{n1}}{a_{nn}} & \frac{a_{n2}}{a_{nn}} & \cdots & \frac{a_{n,n-1}}{a_{nn}} & 0 \end{bmatrix}$$

Partindo de $x^{(0)}$ e fazendo sucessivamente a iteração temos

$$x^{(k)} = \underbrace{J^k}_{\substack{\text{elevado} \\ \text{a } k}} x^{(0)} + [1 + J + J^2 + \ldots + J^{k-1}]c \tag{5.24}$$

Para que convirja, requer que

$$\lim_{k\to\infty} J^k = [0]$$

O que implica que $\lim_{k\to\infty}[1+J+J^2+\ldots+J^{k-1}] = (1-J)^{-1}$. Assim quando (5.24) é satisfeita, $x = \lim_{k\to\infty} x^{(k)}$ existe e $x = 0+(1-J)^{-1}c$, isto é, $(1-J)x = c$ ou $x = Jx+c$.

Mas a condição (5.24) é válida se e somente se todos os auto valores da matriz J forem em módulo < 1.

Seja $\rho_s = \max|\lambda_1|, |\lambda_2|, \ldots, |\lambda_n|$ onde $|\lambda_i|$ são os autovalores da matrix J. ρ_s é também chamado de raio espectral (*"spectral radius"*).

Então para atingir precisão p após k iterações devemos ter

$$\rho_s^k \approx 10^{-p} \;\rightarrow\; \boxed{k \approx -\frac{p\ln 10}{\ln \rho_s}}$$

Assim se ρ_s estiver próximo de 1 a convergência será muito lenta. Existem métodos de aceleração. Ver *Quinney* e *NR* seção 19.5.

Determinar os auto valores da matriz J requerirá outro algoritmo, em geral. Na prática, muitas vezes é mais fácil testar numericamente a convergência.

Critério das linhas

Uma condição mais simples de convergência , porém apenas suficiente, é que o sistema possua diagonal principal estritamente dominante, ou seja,

$$|a_{ii}| > \sum_{\substack{j=1\\ j\neq i}} |a_{ij}|, i = 1, \ldots, n \tag{5.25}$$

que é chamado de critério das linhas. Note que por este critério, os elementos da diagonal principal nunca podem ser nulos.

Exercício: mostre que a matriz do sistema

$$\begin{bmatrix} 4 & 2 & 1 \\ -1 & 2 & 0 \\ 2 & 1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 3 \\ 16 \end{bmatrix}$$

satisfaz o critério das linhas.

Por ser um critério apenas suficiente, um sistema que não satisfaz o critério das linhas pode convergir. Além disso, alterando a ordem das linhas ou colunas pode-se tornar um sistema convergente em divergente e vice-versa.

5.13.3 Método de Gauss-Seidel

O método de Gauss-Seidel é muito semelhante ao método de Jacobi, mas em geral apresenta uma convergência mais rápida. Neste método, aproveita-se os valores já calculados em uma iteração (ex:, $x_1^{(k+1)}$) para a estimativa dos termos seguintes.

As relações de recorrência tomam a seguinte forma

$$\begin{cases} x_1^{(k+1)} &= \frac{1}{a_{11}}(b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - \ldots - a_{1n}x_n^{(k)}) \\ x_2^{(k+1)} &= \frac{1}{a_{22}}(b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)} - \ldots - a_{2n}x_n^{(k)}) \\ x_3^{(k+1)} &= \frac{1}{a_{22}}(b_3 - a_{31}x_1^{(k+1)} - a_{32}x_2^{(k+1)} - \ldots - a_{2n}x_n^{(k)}) \\ \vdots & \\ x_n^{(k+1)} &= \frac{1}{a_{nn}}(b_n - a_{n1}x_1^{(k+1)} - \ldots - a_{n,n-1}x_{n-1}^{(k+1)}) \end{cases}$$

ou, de forma mais compacta,

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) .$$

Exemplo: vamos considerar novamente o sistema

$$\begin{cases} 4x_1 + 2x_2 + x_3 &= 11 \\ -x_1 + 2x_2 &= 3 \\ 2x_1 + x_2 + 4x_3 &= 16 \end{cases} .$$

As relações de recorrência, pelo método de Gauss-Seidel, são

$$\begin{aligned} x_1^{(k+1)} &= \frac{11}{4} - \frac{1}{2} x_2^{(k)} - \frac{1}{4} x_3^{(k)} , \\ x_2^{(k+1)} &= \frac{3}{2} + \frac{1}{2} x_1^{(k+1)} , \\ x_3^{(k+1)} &= 4 - \frac{1}{2} x_1^{(k+1)} - \frac{1}{4} x_2^{(k+1)} . \end{aligned} \tag{5.26}$$

Começando novamente com o vetor $x^{(0)} = [1, 1, 1]$ obtemos sucessivamente

$$x^{(1)} = \begin{pmatrix} 2 \\ 5/2 \\ 19/8 \end{pmatrix} , \; x^{(2)} = \begin{pmatrix} 29/32 \\ 125/64 \\ 783/256 \end{pmatrix} , \; x^{(3)} = \begin{pmatrix} 1033/1024 \\ 4095/2048 \\ 24541/8192 \end{pmatrix} \approx \begin{pmatrix} 1.0087 \\ 1.9995 \\ 2.9957 \end{pmatrix} .$$

Note que, neste exemplo, a taxa de convergência é muito maior.

5.13.4 Convergência do Método de Gauss-Seidel

O critério das linhas também pode ser aplicado ao método de Gauss-Seidel, mas, como no método de Jacobi, trata-se apenas de uma condição suficiente.

Para o método de Gauss-Seidel existe um outro critério, menos restritivo que o critério das linhas, chamado *critério de Sassenfeld*. Seja

$$M = \max_{1 \le i \le n} \beta_i ,$$

onde os β_i são definidos por

$$\begin{aligned} \beta_1 &= \frac{|a_{12}| + |a_{13}| + \cdots + |a_{1n}|}{|a_{11}|} , \\ \beta_i &= \frac{\sum_{j=1}^{i-1} \beta_j |a_{ij}| + \sum_{j=i+1}^{n} |a_{ij}|}{|a_{ii}|} . \end{aligned}$$

A condição $M < 1$ é suficiente para que as aproximações sucessivas pelo método de Gauss-Seidel convirjam.

Muito importante: A convergência (ou não) dos métodos de Jacobi ou Gauss-Seidel independe do vetor inicial escolhido.

Exercício: use o método de Gauss-Seidel para resolver o sistema abaixo. Verifique se a matriz satisfaz o critério das linhas e o critério da Sassenfeld.

$$\begin{bmatrix} 10 & -2 & -2 & 1 \\ -2 & 5 & -1 & -1 \\ 1 & 1/2 & -6 & 1 \\ -1 & -1 & 0 & 20 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ -9 \\ 17 \end{bmatrix}$$

Capítulo 6

interpolação e Extrapolação

6.1 Introdução

Suponhamos um conjunto de $n+1$ pontos com duas coordenadas x e y, conhecidos por um processo qualquer

$$(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$$

onde

$$x_0 < x_1 < x_2 < \ldots < x_n \,.$$

Os valores y_i podem resultar, por exemplo, de um experimento físico, de um conjunto de observações astronômicas, ou mesmo de uma longa série de cálculos numéricos que não podem ser colocados em uma forma funcional simples. O problema de interpolação consiste em achar, para um determinado valor de x, $x \neq x_0, x_1, \ldots, x_n$ e $x_0 < x < x_n$, um valor razoável para y. Já o problema da extrapolação consiste em estimar y para valores de x fora do intervalo $[x_0, x_n]$.

A interpolação é feita determinando-se uma *função interpolante*, $y = f(x)$, a partir das coordenadas conhecidas. De forma geral, quando se procura determinar a função interpolante existem duas situações distintas:

1. Se o conjunto de coordenadas existente tem uma alta precisão, ou seja, se os valores y_i são bem conhecidos, é razoável exigir que a função interpolante satisfaça: $y_i = f(x_i),\ i = 0, 1, \ldots, n$;

2. Caso contrário esta exigência não é justificável, e podemos ter $y_i \neq f(x_i)$, o que poderá inclusive corrigir valores obtidos imprecisamente.

A interpolação é relacionada a (mas distinta de) outro problema muito comum, que é a *aproximação de funções*. Esta tarefa consiste em encontrar uma função que seja facilmente computável para ser usada no lugar de uma função mais complicada. No caso da interpolação, em geral conhece-se um conjunto de valores y_i para valores da abcissa sobre os quais não se tem controle algum (os valores x_i e y_i são simplesmente dados). No caso de aproximações de funções, pode-se computar a função original para quaisquer valores de x com o propósito de se desenvolver a aproximação.

A interpolação sempre assume algum grau de suavidade da função interpolada, o que frequentemente pode não ser o caso. Por exemplo, considere a função

$$f(x) = 3x^2 + \frac{1}{\pi^4} \ln[(\pi - x)^2] + 1 \,, \tag{6.1}$$

que é muito bem comportada exceto para $x = \pi$. Se fornecemos os valores de $f(x)$ calculados em $x = 3.13$, 3.14, 3.15 e 3.16 para qualquer interpolador, certamente teremos uma resposta muito errada se interpolarmos para $x = 3.1416$, embora um gráfico unindo estes 4 pontos pareça muito suave, como ilustrado na figura abaixo.

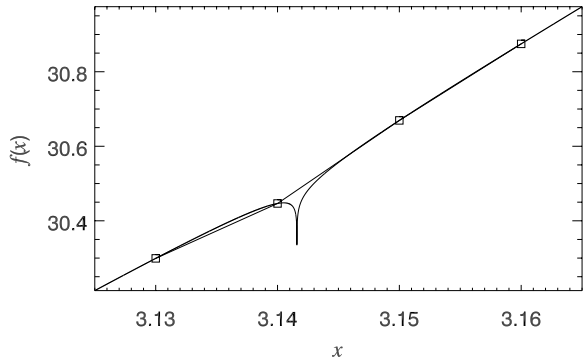


Figura 6.1: Exemplo de função problemática para qualquer interpolador.

Conceitualmente, a interpolação transcorre em dois estágios:

1. ajusta-se (uma vez) uma função interpolante para os pontos fornecidos, e
2. avalia-se (tantas vezes quantas forem necessárias) a função interpolante para os valores de x desejados.

Na prática, este método, conhecido como interpolação global, não é o melhor de se utilizar pois é computaciolmente ineficiente. Em geral inicia-se com o valor tabulado mais próximo ao valor de x desejado e realiza-se uma sequência de correções à medida que outros pontos próximos são considerados (*interpolação local*). O procedimento tipicamente toma $\mathcal{O}(m^2)$ operações, onde $m \ll n$ é o número de pontos efetivamente usados. Se tudo correr bem, a última correção será a menor de todas, e seu valor pode ser usado como uma estimativa informal (mas não rigorosa) do erro. Em esquemas como este, pode-se também dizer que há duas etapas:

1. encontra-se ponto inicial adequado na tabela (digamos, x_i), e
2. faz-se a interpolação usando-se os m pontos vizinhos a i
   (por exemplo, centrados em x_i).

A interpolação local dá valores interpolados que em geral não têm derivadas contínuas. Isso ocorre porque para valores x diferentes usa-se um subconjunto diferente de pontos para a interpolação. Para situações em que a continuidade das derivadas é uma questão importante, deve-se usar a chamada função *spline*, que veremos abaixo. Splines cúbicas são as mais populares, elas garantem que a função interpolada seja contínua até a segunda derivada, e tendem a produzir resultados melhores e mais estáveis que polinômios.

O número m de pontos usado na interpolação menos 1 é chamado de ordem da interpolação. Aumentar a ordem não necessariamente aumenta a acurácia, especialmente no caso de interpolação polinomial (ver figura 6.2).

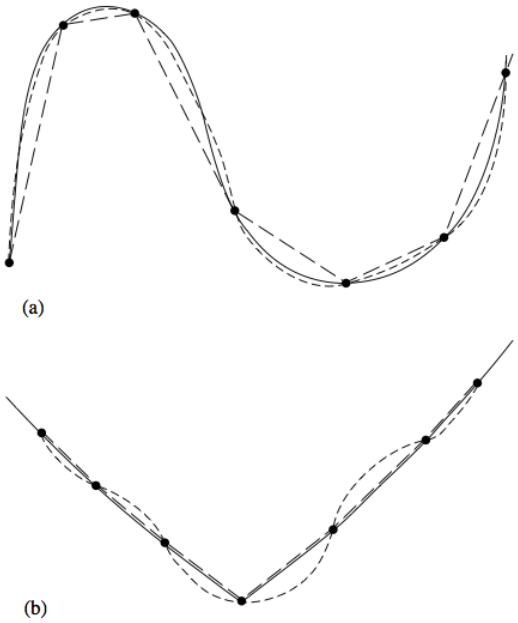


Figura 6.2: (a) Uma função suave (linha sólida) é interpolada de forma mais acurada por um polinômio de ordem mais alta (traços curtos) do que polinômios de ordem baixa (traços longos). (b) Uma função com cantos (derivadas descontínuas) é pior representada por polinômios de alta ordem. [extraído de Numerical Recipes]

Os métodos de interpolação abaixo são também métodos para extrapolação. Neste caso, deve-se tomar muito cuidado em monitorar os erros, caso contrário os resultados podem ser catastróficos. Uma função interpolante, quando usada numa extrapolação, pode dar muitos problemas quando x estiver mais distante dos limites do intervalo $[x_0, x_n]$ do que o espaçamento típico dos valores tabulados.

6.2 Funções interpolantes

São inúmeros os tipos de funções utilizadas para interpolação e devemos ter um certo critério para escolher uma delas, levando em consideração o seu grau de suavidade no intervalo considerado e a sua simplicidade.

De forma geral, uma função interpolante $f(x)$ pode ser escrita como

$$f(x) = a_0 f_0(x) + a_1 f_1(x) + \ldots + a_n f_n(x)$$

onde $f_i(x),\ i = 0, 1, \ldots, n$ representa uma classe de funções. Veremos abaixo exemplos de algumas classes de funções comumente utilizadas para construir funções interpolantes.

1. **Monômios**: $f_i(x) = x^i$

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n = \sum_{i=0}^{n} a_i x^i \,.$$

Claramente, $f(x) = P_n(x)$ é um polinômio de grau n

2. **Funções de Fourier**: $f_i(x) = a_i \cos(ix) + b_i \sin(ix)$

$$\begin{aligned} f(x) &= a_0 + a_1 \cos(x) + a_2 \cos(2x) + \ldots + a_n \cos(nx) + \\ &\quad + b_1 \sin(x) + b_2 \sin(2x) + \ldots + b_n \sin(nx) \\ f(x) &= a_0 + \sum_{i=1}^{n} [a_i \cos(ix) + b_i \sin(ix)] \,. \end{aligned}$$

3. **Exponenciais**: $f_i(x) = e^{b_i x}$

$$f(x) = a_0 e^{b_0 x} + a_1 e^{b_1 x} + \ldots + a_n e^{b_n x} = \sum_{i=0}^{n} a_i e^{b_i x} \,.$$

Existem outras classes de funções, porém são menos usadas. A classe mais usada é a dos monômios (ou polinômios) e citamos as seguintes vantagens:

1. Sua teoria é simples e bem desenvolvida;
2. São fáceis de ser calculados;
3. Somas, produtos e diferenças de polinômios são polinômios;
4. Se $P_n(x)$ é um polinômio, $P_n(x + a)$ e $P_n(ax)$ também o são.
5. Outras classes de funções podem ser aproximadas por polinômios. Para isso utiliza-se o Teorema de Aproximação de Weierstrass:

   "Se $f(x)$ é contínua em $[a, b]$, então para $\forall \epsilon > 0$, existe um $P_n(x)$ de grau n, $n = g(\epsilon)$, tal que $|f(x) - P_n(x)| < \epsilon$, com $a \leq x \leq b$".

6.3 Passo preliminar: buscando em uma tabela ordenada

Vimos acima que uma forma usual de se fazer interpolação é considerar apenas os m pontos vizinhos de x na tabela de dados. Faz-se necessário, portanto, uma rotina que procure o índice i do conjunto de abcissas x_i, $i = 1, \ldots, n$ tal que $x_i < x < x_{i+1}$, para o caso em que o conjunto de abcissas seja monotonicamente crescente ($x_0 < x_1 < x_2 < \ldots < x_n$), ou $x_i > x > x_{i+1}$, para o caso de abcissas monotonicamente decrescentes ($x_n < x_{n-1} < \cdots < x_2 < x_1$).

Dois eficientes métodos para se implementar esta procura são o método da bissecção e o método da caçada. O primeiro consiste em bisseccionar sucessivamente o intervalo $[1, n]$, sempre verificando, em cada passo qual subintervalo contém o valor x procurado (Figura 6.3-a). O outro método, chamado método da caçada, parte do princípio de que na maioria das aplicações faz-se chamadas consecutivas de uma rotina de interpolação para valores da abcissa muito próximos. Dessa forma, pode-se guardar a informação da última posição da tabela e "caçar" o próximo valor, seja para cima ou para baixo, em incrementos de 1, 2, 4 e assim sucessivamente, até que o valor de x deseja esteja confinado em um dado subintervalo, quando então a posição na tabela é encontrada por bisseção (Figura 6.3-b).

Exercício de programação: implemente um subrotina em Fortran, usando o método da bissecção, que encontre a posição de x em uma tabela de abcissas dada. A rotina deve retornar um número entre 1 e $n - 1$ caso $x_1 < x < x_n$, 0 caso $x < x_1$ e n caso $x > x_n$.

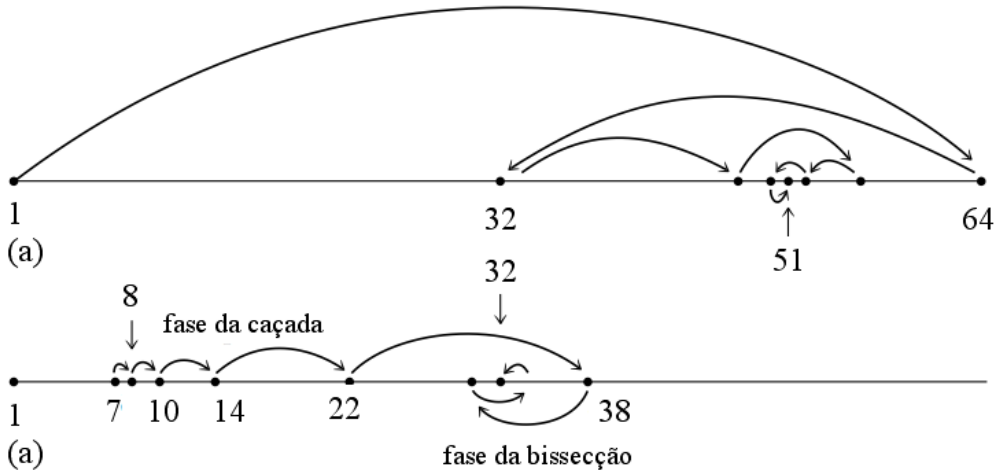


Figura 6.3: (a) Usando a bissecção para encontrar um ponto em uma tabela. São mostrados os passos que convergem para o elemento 51 de uma tabela de 64 pontos. (b) Exemplo do uso do algoritmo da caçada. Mostra-se um caso particularmente desfavorável em que inicia-se no elemento 7 (que fora a último ponto conhecido) para se chegar no ponto 32. [adaptado de Numerical Recipes]

6.4 Interpolação linear

Dado um conjunto de pontos $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, a interpolação linear para x, $x_i < x < x_{i+1}$ é

$$y = (1 - f)y_i + f y_{i+1} \tag{6.2}$$

onde

$$f \equiv \frac{x - x_i}{x_{i+1} - x_i}\,.$$

Conceitualmente, a Eq. (6.2) pode ser interpretada como uma média ponderada de y_i e y_{i+1} com peso f, onde f é a distância relativa de x com respeito a x_i.

A interpolação acima é chamada *piecewise*, pois apenas os extremos de um certo subintervalo são usados na interpolação. Como vimos acima, este tipo de função interpolante é contínua mas possui arestas, ou seja, não possui uma derivada primeira contínua nos pontos x_i.

6.4.1 Interpolação Bilinear

A formulação acima para a interpolação linear pode ser facilmente estendida para interpolação linear em duas dimensões (dita bilinear) e dimensões maiores. Vamos explicitar abaixo o problema para duas dimensões, notando que a extensão a outras dimensões é trivial.

Suponhamos uma função desconhecida $f = f(x, y)$, e que temos um conjunto de valores discretos dessa função para vários pontos x e y

$$f_{ij} = f_i j(x_i, y_j)$$

onde $i = 0, \ldots, n$ e $j = 0, \ldots, n$.

queremos estimar o valor de f em um ponto (x, y)

6.4.2 Interpolação $\log \times \log$

DIGITAR...

6.5 Interpolação polinomial

A interpolação polinomial consiste em utilizar-se um polinômio de certo grau como função interpolante. Para um polinômio de ordem n, descrito por $P_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$, as incógnitas são os coeficientes $a_0, a_1, a_2, \ldots, a_n$. A interpolação linear, vista acima, é um caso particular da interpolação polinomial para $n = 1$.

A questão que se apresenta é encontrar o melhor polinômio interpolante para cada caso. Como visto na seção 6.1, há duas situações limites, dependendo se a condição $P_n(x_i) = y_i,\ i = 0, 1, \ldots, n$ será imposta ou não. Estas duas situações são descritas abaixo.

6.5.1 Polinômio Interpolante

Um polinômio interpolante pode ser utilizado quando os dados são precisos o suficiente, caso em que podemos impor que $P_n(x_i) = y_i,\ i = 0, 1, \ldots, n$.

Podemos fazer uso de um bem-conhecido teorema da Álgebra:

Teorema: existe um e só um polinômio de grau n ou menor que assume valores específicos para $n + 1$ valores de x.

Este teorema assegura que existe apenas um polinômio interpolante de ordem n ou menor, que passa por um conjunto de $n + 1$ pontos. Entretanto, nada garante que o polinômio interpolante seja uma boa aproximação para $x \neq x_i, i = 0, 1, \ldots, n$.

6.5.2 Polinômio dos Mínimos Quadrados

Quando os valores y_i não são precisos, não se deve exigir que $P_n(x_i) = y_i$. Neste caso, se m é o número de pontos, em geral procura-se um $P_n(x)$ tal que $n \ll m$ (ou seja, a ordem do polinômio interpolante deve ser muito menor que o número de pontos disponíveis).

Para se achar o polinômio interpolante aplica-se o critério dos mínimos quadrados, que estabelece que a soma dos quadrados das diferenças entre y_i e $P_n(x_i)$ deve ser mínima. Isso será visto com maiores detalhes no capítulo 7.

6.5.3 Avaliação de Polinômios

Antes de discutir como determinar o polinômio interpolante, vamos fazer uma breve digressão sobre como avaliar numericamente um polinômio. Seja o polinômio $P_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$, onde são conhecidos $a_0, a_1, \ldots, a_n$. Dado um valor de x, vamos calcular $P_n(x)$. Um programa em Fortran seria:

```
REAL :: P
REAL, ALLOCATABLE :: a(:)
INTEGER :: i,n

...
        ALLOCATE(a(n))
```

```
...
        P=0.
        DO i=0,n
                P=P+a(i)*x**i
        ENDDO
```

Desta forma temos $n(n+1)/2$ multiplicações e n adições de ponto flutuante.

Uma forma mais eficiente de implementar o cálculo é usar a regra de Horner, segundo a qual $P_n(x)$ é transformado em

$$P_n(x) = a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + xa_n)\ldots))$$

Em Fortran:

```
...
        P=a(n)
        DO i=n-1,0,-1
                P=P*x+a(i)
        ENDDO
```

6.6 Polinômio interpolante

Veremos agora como construir um polinômio interpolante de ordem n, que passa por um conjunto de $m = n + 1$ pontos. Veremos inicialmente um algoritmo para calcular os coeficientes a_i do polinômio (seção 6.6.1) e depois um algoritmo mais eficiente usado apenas para avaliar o polinômio em um ponto x, situação mais comumente encontrada (seção 6.6.2).

6.6.1 Sistema de equações

Nesta seção descrevemos uma maneira de se determinar os coeficientes do polinômio interpolante. Um uso válido destes coeficientes poderia ser, por exemplo, avaliar simultaneamente o valor interpolado da função e de algumas das suas derivadas. Um ponto importante a ser considerado é que devido a erros de arredondamento, em geral os coeficientes do polinômio podem ser determinados com precisão muito menor do que o valor do polinômio para uma certa abcissa (seção 6.6.2). Assim, a não ser que os coeficientes sejam realmente necessários, o método abaixo deve ser evitado.

Seja uma tabela de $n+1$ pontos, (x_i, y_i), $i = 0, \ldots, n$. Para determinar os coeficientes do polinômio $P_n(x)$ que passa por estes pontos, montamos o seguinte sistema de equações

$$\begin{cases} a_0 + a_1 x_0 + a_2 x_0^2 + \ldots + a_n x_0^n = y_0 \\ a_0 + a_1 x_1 + a_2 x_1^2 + \ldots + a_n x_1^n = y_1 \\ \vdots \\ a_0 + a_1 x_n + a_2 x_n^2 + \ldots + a_n x_n^n = y_n \end{cases}$$

onde $a_0, a_1, a_2, \ldots, a_n$ são os coeficientes do polinômio, que são as incógnitas deste sistema. Uma maneira de se determinar os coeficientes pode ser resolver o sistema abaixo pelo

método de Gauss

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ \vdots & & & & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} .$$

A matriz acima é conhecida como *matriz de Vandermonde*. A existência de várias potências de x numa mesma linha desta matriz tende a tornar o sistema desbalanceado, ou seja, haverá um grande intervalo dinâmico entre a segunda e a última coluna de cada linha. Por exemplo, se $x_i = 0.1$ e $n = 10 \rightarrow x_i^{10} = 10^{-10}$. Isso pode causar problemas sérios de arredondamento, que comprometerão a acurácia na determinação dos coeficientes. Existem métodos específicos para a resolução deste tipo de sistema, que são descritos na seção 2.8 do Numerical Recipes.

6.6.2 Polinômios de Lagrange

Existem outros métodos de se avaliar o polinômio interpolante de modo mais simples que a resolução do sistema. Nestes métodos, a característica fundamental é que o que se avalia é o polinômio para um dado valor de x, e não os coeficientes propriamente ditos.

Seja uma função tabelada para $n+1$ pontos distintos, $(x_i, y_i),\ i = 0, \dots, n$, e sejam os polinômios de grau n definidos como

$$L_i(x) \equiv \frac{(x-x_0)(x-x_1)\dots(x-x_{i-1})(x-x_{i+1})\dots(x-x_n)}{(x_i-x_0)(x_i-x_1)\dots(x_i-x_{i-1})(x_i-x_{i+1})\dots(x_i-x_n)} .$$

Os polinômios $L_i(x)$ são conhecidos como polinômios de Lagrange. Um uma forma mais compacta, pode-se escrevê-los como

$$L_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j} .$$

É fácil ver que para $i \geq 0$ e $j \leq n$,

$$L_i(x_j) = \begin{cases} 1 & \text{se } i = j \\ 0 & \text{se } i \neq j \end{cases} . \tag{6.3}$$

Desta forma, podemos determinar o polinômio interpolador de y relativamente aos pontos $x_i, i = 1, \dots, n$, utilizando os polinômios de Lagrange, da seguinte maneira

$$P_n(x) = L_0(x)y_0 + L_1(x)y_1 + ... + L_n(x)y_n = \sum_{i=0}^{n} L_i(x) f(x_i) . \tag{6.4}$$

Como os polinômios de Lagrange satisfazem as condições (6.3), é evidente que $P_n(x_i) = y_i, i = 1, \dots, n$. Além disso, o grau de P_n é menor ou igual a n. Segue-se, portanto, que o polinômio P_n é o único polinômio interpolador que passa pelos $n+1$ pontos.

Exemplo: Calcule o polinômio interpolante para os pontos

$$(-1, -11); (1, 3); (2, 7); (3, 17)$$

$$(x_0, y_0); (x_1, y_1); (x_2, y_2); (x_3, y_3)$$

$$
\begin{aligned}
P_3(x) &= \frac{(x-1)(x-2)(x-3)}{(-1-1)(-1-2)(-1-3)}(-11) \\
&+ \frac{(x+1)\quad(x-2)(x-3)}{(1+1)\quad(1-2)(1-3)}(3) \\
&+ \frac{(x+1)(x-1)\quad(x-3)}{(2+1)(2-1)\quad(2-3)}(7) \\
&+ \frac{(x+1)(x-1)(x-2)}{(3+1)(3-1)(3-2)}(17) \\
P_3(x) &= \frac{11}{24}(x^3-6x^2+11x-6) + \frac{18}{24}(x^3-4x^2+x+6) \\
&- \frac{56}{24}(x^3-3x^2-x+3) + \frac{51}{24}(x^3-2x^2-x+2) \\
P_3(x) &= x^3-3x^2+6x-1\,.
\end{aligned}
$$

Algumas observações importantes:

- O polinômio é o mesmo que aquele calculado por um sistema de equações, pois ele é único;
- Em geral as fórmulas de Lagrange não são usadas para determinar os coeficientes do polinômio interpolante, devido à complexidade dos cálculos e do algoritmo necessário (ver seção 6.6.3) ;
- Estas fórmulas são usadas diretamente para interpolar $y = P_n(x)$;
- Com dois "loops", um para o cálculo de $\prod$ e outro para o cálculo de $\sum$, obtém-se o valor de $y = P_n(x)$ interpolado.

6.6.3 Um método alternativo

Digitar. Numerical Recipes, 3a. edição, seção 3.5.

6.6.4 Falhas do polinômio interpolante

O método de interpolação por polinômios não funciona indiscriminadamente para todas as funções, e em todo o intervalo analisado. Ele pode ser particularmente problemático quando a tabela interpolada tem os pontos da abcissa igualmente espaçados.

Um exemplo clássico dos problemas que podem ocorrer com a interpolação polinomial é o fenômeno de Runge (*Runge's Phenomenon*). Consideremos como função geradora dos pontos base a função de Runge:

$$f(x) = \frac{1}{1+25x^2}\,.$$

A Fig. 6.4 mostra o gráfico da função de Runge no intervalo $[-1,1]$ e as curvas de pois polinômios interpolantes ($P_5(x)$ e $P_{20}(x)$), construídos a partir de pontos igualmente espaçados no intervalo

$$x_i = -1 + (i-1)\frac{2}{n}, i = 1,\ldots,n\,.$$

Vemos que a divergência é tão maior quanto maior for o grau do polinômio. Pode-se, inclusive, mostrar que o erro na interpolação tende a infinito quando o grau do polinômio aumenta.

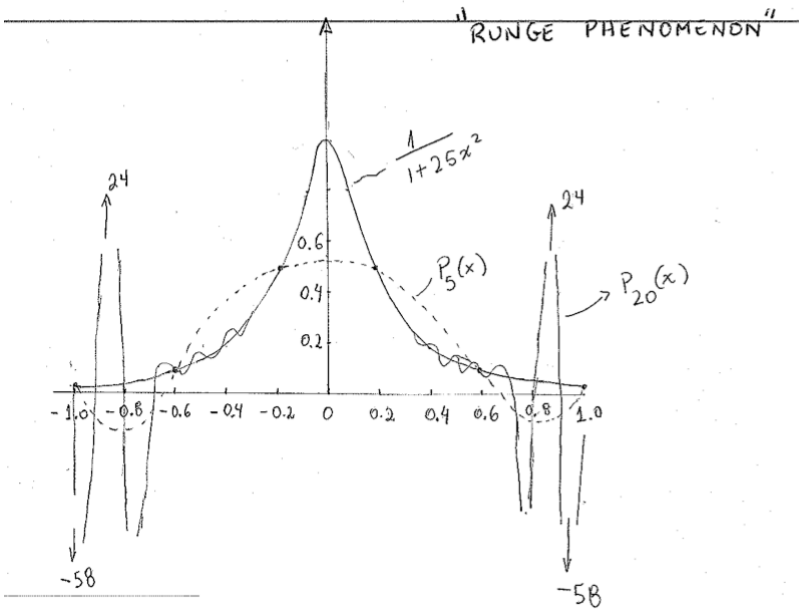


Figura 6.4: O fenômento de Runge.

6.7 Erro do Polinômio Interpolante

DIGITAR

6.8 Interpolação por Spline Cúbica

Em matemática, uma *spline* é uma função polinomial suave que é definida por partes (*piecewise*) e possui uma alta suavidade nos locais onde as diferentes partes se conectam (conhecidas como nós ou pontos-base). O termo spline vem do técnica empregada por desenhistas para traçarem curvas suaves: imagine uma régua flexível e indeformável, que é presa na superfície de desenho em alguns pontos: esta régua assumira a forma de uma spline.

Splines cúbicas são frequentemente empregadas em interpolação com a finalidade de garantir a suavidade da função interpolante. Trata-se de uma solução muito eficiente para o fenômeno de Runge visto anteriormente. Matematicamente, o método consiste na definição de um polinômio de 3º grau para cada par de pontos consecutivos. Ou seja, dados os pontos $y_i = y(x_i)$, $i = 0, \ldots, n$, define-se um polinômio cúbico para o intervalo $[x_0, x_1]$, $p_1(x)$, outro para o intervalo $[x_1, x_2]$, $p_2(x)$, e assim por diante. Para $n+1$ pontos, temos assim n polinômios. Obviamente, os polinômios devem ser tais que a função seja contínua nos nós. Além disso, para que a variação do raio de curvatura seja contínua nos nós, em cada um a 2ª derivada do polinômio anterior deve ser igual à do polinômio posterior. O mesmo se dá com a 1ª derivada. As condições explicitadas acima traduzem-se em $4n - 2$ equações, listadas na Tabela 6.1.

Equações	Condições
n	$p_i(x_i) = y_i$
n	$p_i(x_{i+1}) = y_{i+1}$
$n-1$	$p_i''(x_i) = p_{i-1}''(x_i)$
$n-1$	$p_i'(x_i) = p_{i-1}'(x_i)$
$4n-2$	

Tabela 6.1: condições necessárias para se determinar a função spline que passa por $n+1$ pontos.

6.8.1 Dedução das fórmulas de spline

O procedimento básico consiste em determinar os 4 coeficientes de cada um dos n polinômios de 3º grau, ou seja, $4n$ incógnitas precisam ser determinadas. Como vimos, os valores dos polinômios e suas derivadas primeira e segunda nos $n+1$ pontos fornecem apenas $4n-2$ equações, pois não podemos calcular $p_1'(x_0)$ e $p_1''(x_0)$. Para contornar esse problema, assumiremos que tanto no primeiro quanto no último segmento da função spline a inclinação é constante (ou seja, nos extremos do intervalo a função converte-se em uma reta). Logo, para estes segmentos a 1ª derivada é uma constante e a 2ª derivada é nula. Dessa forma, ao impormos que $p_1''(x_0) = 0$ e $p_n''(x_n) = 0$, ficamos com as $4n$ equações necessárias para tornar o sistema determinado. Uma função spline assim construída chama-se função spline natural.

Para facilitar a notação, vamos definir

$$h_i \equiv x_{i+1} - x_i \,,$$

$$\phi_i \equiv p_{i-1}''(x_i) = p_i''(x_i) \,.$$

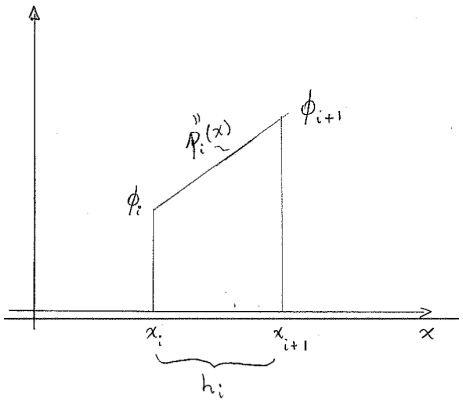


Figura 6.5: Representação gráfica da segunda derivada de p_i.

Como os p_i são polinômios de 3º grau, temos que cada $p_i''(x)$ é um segmento de reta (Fig. 6.5). Dessa forma, podemos escrever, usando a Eq. (6.2)

$$p_i''(x) = \frac{(x_{i+1} - x)}{h_i}\phi_i + \frac{(x - x_i)}{h_i}\phi_{i+1} \,.$$

Integrando $p_i''(x)$ duas vezes, obtemos

$$p_i'(x) = -\frac{1}{2}\frac{(x_{i+1}-x)^2}{h_i}\phi_i + \frac{1}{2}\frac{(x-x_i)^2}{h_i}\phi_{i+1} + C_1,$$

e

$$p_i(x) = \left(-\frac{1}{2}\right)\left(-\frac{1}{3}\right)\frac{(x_{i+1}-x)^3}{h_i}\phi_i + \frac{1}{2}\frac{1}{3}\frac{(x-x_i)^3}{h_i}\phi_{i+1} + C_1 x + C_2\,.$$

Vamos agora fazer uma mudança de variáveis, de forma a reescrever o último termo da equação acima como

$$C_1 x + C_2 = C_1'(x_{i+1}-x) + C_2'(x-x_i)\,.$$

Dessa forma, obtemos

$$p_i(x) = \frac{1}{6}\frac{(x_{i+1}-x)^3}{h_i}\phi_i + \frac{1}{6}\frac{(x-x_i)^3}{h_i}\phi_{i+1} + C_1'(x_{i+1}-x) + C_2'(x-x_i)\,.$$

Para obtermos as constantes de integração C_1' e C_2', impomos as condições $p_i(x_i) = y_i$ e $p_i(x_{i+1}) = y_{i+1}$ (tabela 1)

$$y_i = \frac{1}{6}\frac{(x_{i+1}-x_i)^3}{h_i}\phi_i + C_1'(x_{i+1}-x_i) \to C_1' = \frac{y_i}{h_i} - \frac{h_i\phi_i}{6}$$

$$y_{i+1} = \frac{1}{6}\frac{(x_{i+1}-x_i)^3}{h_i}\phi_{i+1} + C_2'(x_{i+1}-x_i) \to C_2' = \frac{y_{i+1}}{h_i} - \frac{h_i\phi_{i+1}}{6}$$

. Finalmente, chegamos à seguinte expressão para $p_i(x)$, que representa cada uma das partes da função spline

$$p_i(x) = \frac{\phi_i}{6h_i}(x_{i+1}-x)^3 + \frac{\phi_{i+1}}{6h_i}(x-x_i)^3 + \left(\frac{y_i}{h_i} - \frac{h_i\phi_i}{6}\right)(x_{i+1}-x) + \left(\frac{y_{i+1}}{h_i} - \frac{h_i\phi_{i+1}}{6}\right)(x-x_i)\,, \tag{6.5}$$

onde os $\phi_1, \ldots, \phi_n$ são incógnitas.

Para obtermos os ϕ_i, devemos impor as condições da tabela 6.1 que ainda não foram utilizadas, a saber, as condições sobre a primeira derivada de $p_i(x)$. Imporemos a condição que a derivada primeira é contínua nos nós das função spline, ou seja,

$$p_i'(x_i) = p_{i-1}'(x_i)\,. \tag{6.6}$$

Fazendo a derivada da Eq. (6.5) e depois de alguma manipulação algébrica trivial, obtemos

$$p_i'(x_i) = -\frac{1}{2}h_i\phi_i - \left(\frac{y_i}{h_i} - \frac{h_i\phi_i}{6}\right) + \frac{y_{i+1}}{h_i} - \frac{h_i\phi_{i+1}}{6}\,,$$

e

$$p_{i-1}'(x_i) = \frac{1}{2}h_{i-1}\phi_i - \left(\frac{y_{i-1}}{h_{i-1}} - \frac{h_{i-1}\phi_{i-1}}{6}\right) + \left(\frac{y_i}{h_{i-1}} - \frac{h_{i-1}\phi_i}{6}\right)\,.$$

Aplicando a condição (6.6), obtemos, depois de alguma manipulação algébrica

$$h_{i-1}\phi_{i-1} + 2(h_{i-1}+h_i)\phi_i + h_i\phi_{i+1} = 6\left(\frac{y_{i+1}-y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}\right) \tag{6.7}$$

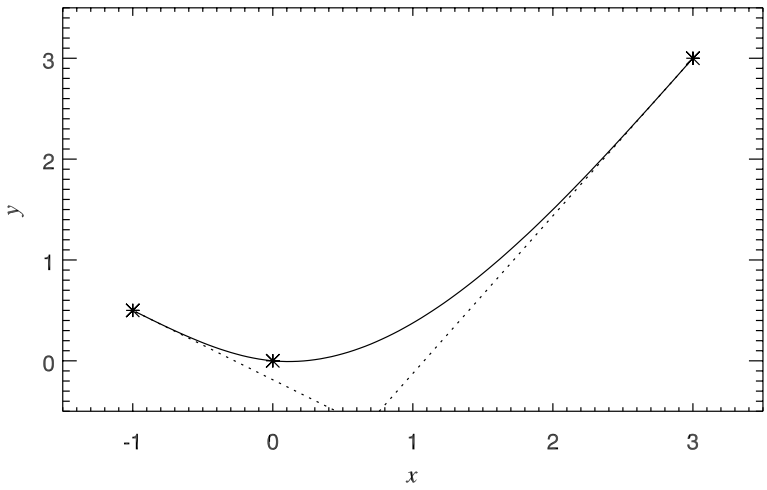


Figura 6.6: Ajuste de spline cúbica que passa pelos pontos $(-1, 1/2)$, $(0, 0)$ e $(3, 3)$. As linhas tracejadas mostram as retas para as quais a função converge nos extremos no intervalo

As Eqs. (6.7), definidas para $i = 1, \ldots, n-1$, representam um sistema de $n-1$ equações. O número de incógnitas é $n+1$, mas duas já estão determinadas por termos imposto que a segunda derivada da primeira e última parte da função é nula (ou seja $\phi_0 = 0$ e $\phi_n = 0$). Antes de mostrar um método geral de obter os coeficientes ϕ_i, vamos ver um exemplo simples.

Exemplo: vamos calcular a função spline cúbica que passa pelos pontos $(-1, 1/2)$, $(0, 0)$ e $(3, 3)$.

Neste caso, temos que determinar $p_1(x)$ e $p_2(x)$. Das definições acima, temos que $h_0 = 1$, $h_1 = 3$, $\phi_0 = 0$ e $\phi_2 = 0$. Falta-nos apenas determinar ϕ_1. Da Eq. (6.7) temos

$$0 + 8\phi_1 + 0 = 6(1 + 0.5)\,,$$

ou seja, $\phi_1 = 9/8$. Da Eq. (6.6) temos que

$$p_1(x) = 0 + \frac{9/8}{6}(x - (-1))^3 + (\frac{1/2}{1} - 0)(0 - x) + (0 - \frac{9/8}{6})(x + 1)\,,$$

ou

$$p_1(x) = \frac{3}{16}(x + 1)^3 + \frac{11}{16}x - \frac{3}{16}\,.$$

e

$$p_2(x) = \frac{9/8}{6 \times 3}(3 - x)^3 + 0 + \left(0 - \frac{3 \times 9/8}{6}\right)(3 - x) + \left(\frac{3}{3} - 0\right)(x - 0)\,,$$

ou

$$p_2(x) = \frac{1}{16}(3 - x)^3 + \frac{25}{16}x - \frac{27}{16}\,.$$

A Fig. 6.6 mostra o função spline determinada sobreposta aos três pontos dados. Note que nos extremos do intervalo a função se transforma em uma linha reta, como esperado.

Exercício: mostrar que as derivadas primeira e segunda da função spline determinada acima é contínua em $(0, 0)$.

As Eqs. (6.7) podem ser escritas em forma matricial da seguinte maneira, formando um sistema $(n-1) \times (n-1)$

$$\begin{bmatrix} 2(h_0+h_1) & h_1 & & & \\ h_1 & 2(h_1+h_2) & h_2 & & \\ & \cdots & \cdots & \cdots & \\ & h_{n-3} & 2(h_{n-3}-h_{n-2}) & h_{n-2} & \\ & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{n-2} \\ \phi_{n-1} \end{bmatrix} = \begin{bmatrix} e_1 - e_0 \\ e_2 - e_1 \\ \vdots \\ e_{n-2} - e_{n-3} \\ e_{n-1} - e_{n-2} \end{bmatrix}$$

onde definimos

$$e_i = 6\left(\frac{y_{i+1} - y_i}{h_i}\right),$$

para simplificar a notação. Este sistema tridiagonal pode ser resolvido através de uma implementação direta usando a decomposição LU, vista no Cap. 5.

Inicialmente, efetuamos a decomposição LU da matriz tridiagonal acima, escrevendo

$$\begin{cases} u_1 = 2(h_0 + h_1) \\ l_j = \frac{h_{j-1}}{u_{j-1}},\ j = 2, \ldots, n-1 \\ u_j = 2(h_{j-1} + h_j) - \frac{h_{j-1}^2}{u_{j-1}},\ j = 2, \ldots, n-1 \end{cases}$$

Uma vez feita a decomposição LU, o sistema é resolvido em dois passos, como mostrado na Seção 5.10.2. Inicialmente calculamos o vetor y por substituição para frente

$$\begin{cases} y_1 = e_1 - e_0 \\ y_j = (e_j - e_{j-1}) - l_j y_{j-1},\ j = 2, \ldots, n-1 \end{cases}$$

e finalmente calcula-se os ϕ's da seguinte forma

$$\begin{cases} \phi_{n-1} = \frac{y_{n-1}}{u_{n-1}} \\ \phi_j = \frac{y_j - h_j \phi_{j+1}}{u_j},\ j = n-2, \ldots, 1 \end{cases} . \tag{6.8}$$

A interpolação por spline pode ser facilmente implementada computacionalmente. Sugere-se que sejam feitas duas subrotinas, a primeira lê os $n+1$ valores dos pontos tabelados e retorna um array com os h's e outro com os ϕ's. Uma outra subrotina usa estes dois arrays e a Eq. (6.5) para avaliar a função no intervalo i desejado.

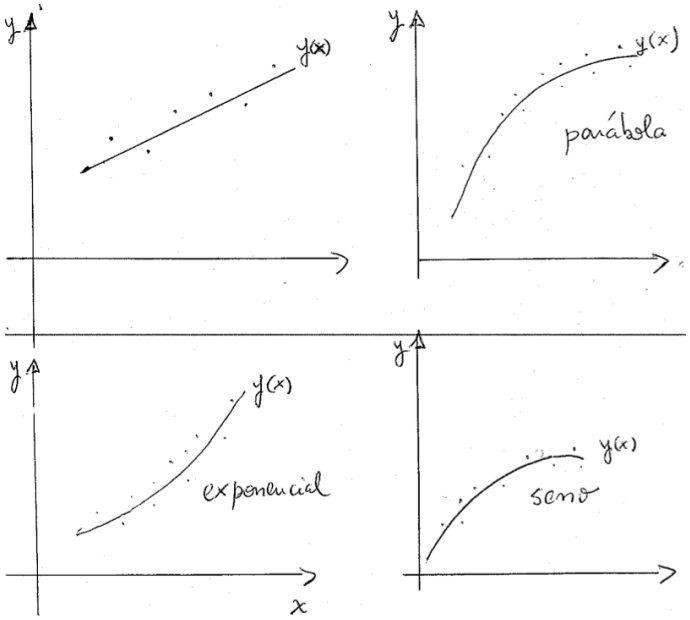


Figura 7.1: O problema do ajuste de funções a um conjunto de dados

Capítulo 7

Aproximação de Funções por Mínimos Quadrados

7.1 Introdução

Dado um conjunto de observações (dados), frequentemente deseja-se condensar os dados ajustando a eles um modelo que depende de parâmetros ajustáveis. Às vezes o modelo é simplesmente uma classe conveniente de funções, tais como polinômios, exponenciais, Gaussianas, etc. e o ajuste visa simplesmente obter os coeficientes apropriados (Fig. 7.1). Em outras situações, os parâmetros do modelo vêm de alguma teoria subjacente que os dados devem necessariamente satisfazer. Por exemplo, os dados podem ser a posição no céu de um asteróide em função do tempo e os parâmetros a serem ajustados os elementos orbitais da órbita do asteróide.

O procedimento de ajuste de dados por uma função envolve definirmos uma *função de mérito*, que mede a concordância entre os dados e o modelo com uma dada escolha

de parâmetros. Em estatística frequentista, a função de mérito é em geral escolhida de forma que pequenos valores da função de mérito representam uma boa concordância entre os dados e a função ajustada. Dessa forma, o ajuste de funções é um problema de minimização em múltiplas dimensões.

Há outras questões que vão além de se encontrar o melhor ajuste. Dados são, em geral, inexatos, ou seja, estão associados a erros (ou ruídos, no contexto de processamento de sinais). Assim, dados típicos nunca ajustam exatamente o modelo adotado, mesmo quando o modelo é correto. Necessitamos, assim, de meios para avaliar se o modelo é ou não apropriado, ou seja, precisamos testar a qualidade do ajuste usando-se algum tipo de teste estatístico.

Outra questão é que, além de obtermos os parâmetros do nosso melhor ajuste, devemos também determinar a acurácia com a qual estes parâmetros foram obtidos. Em termos frequentistas, devemos obter os erros associados a estes parâmetros.

Como aproximar uma função (modelo) a um conjunto de pontos?

Suponhamos um conjunto de pontos (x_i, y_i), $i = 1, \ldots, m$, que desejamos aproximar por uma função $y = f(x)$, escrita como uma combinação linear de uma família de funções, f_i, de forma que

$$y(x) = c_1 f_1(x) + c_2 f_2(x)_+ \ldots + c_n f_n(x)\,,$$

onde os c_i, $i = 1, \ldots, n$ são os parâmetros livres do problema. Note que o número de parâmetros livres, n, deve ser menor ou igual ao número de pontos, m, e que escolha da família de funções depende da natureza do problema. A aproximação consiste em achar os parâmetros livres que minimizem uma função de mérito, que, como vimos, é uma função que mede a qualidade da aproximação.

Vamos definir uma função de mérito (que chamaremos resíduo), da seguinte forma

$$r = \sum_{i=1}^{n} r_i = \sum_{i=1}^{n} [y(x_i) - y_i]\,. \tag{7.1}$$

Assim, uma boa aproximação seria obtida minimizando o resíduo, por exemplo impondo que $r = 0$. Analisemos tal função de mérito com um exemplo simples. Suponhamos que foi realizado um experimento em que se obtiveram os pontos (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), mostrados na Figura 7.4. Pode-se observar que todas as retas que foram traçadas na Figura obedecem ao critério $r = 0$, pois os pontos 1 e 2 são simétricos em relação aos pontos 3 e 4 para todas elas. Isso mostra que minimizar o resíduo não é uma boa escolha para se aproximar uma função. Portanto, a função (7.1) não é uma boa função de mérito.

O problema com esta função reside no fato dos resíduos r_i serem tanto positivos quanto negativos. Assim, se deixarmos de considerar o sinal dos erros evitaremos este problema. Uma forma de fazer isso é considerar o módulo dos erros, mais isso introduziria dificuldades matemáticas indesejadas. Outro critério com esta mesma característica, porém com tratamento matemático mais simples, é exigir que

$$\sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} [y(x_i) - y_i]^2 \tag{7.2}$$

seja mínimo. Este é o famoso critério dos mínimos quadrados.

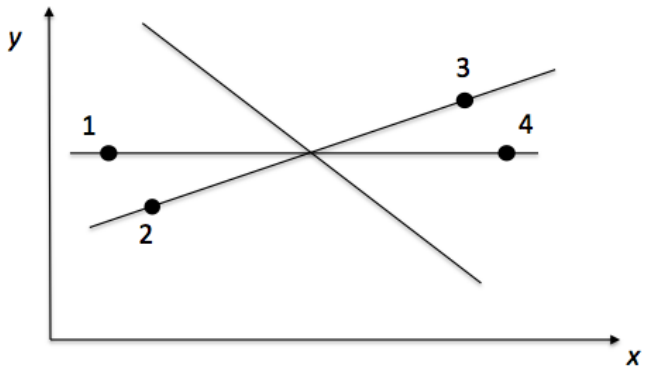


Figura 7.2: Exemplo que ilustra porque o resíduo não é uma boa função de mérito.

7.2 Critério dos Mínimos Quadrados

7.2.1 Sem pesos

Nosso objetivo é aproximar a função geral

$$y(x) = c_1 f_1(x) + c_2 f_2(x)_+ \ldots + c_n f_n(x)\,,$$

com n parâmetros livres, a um conjunto de m pontos usando-se o critério dos mínimos quadrados como função de mérito. Faremos isso minimizando a quantidade

$$\chi^2 \equiv \sum_{i=1}^{m} [y(x_i) - y_i]^2 = \sum_{i=1}^{m} [c_1 f_1(x_i) + c_2 f_2(x_i) + \ldots + c_n f_n(x_i) - y_i]^2\,, \tag{7.3}$$

chamada de "qui-quadrado".

O χ^2 será mínimo quando as n derivadas parciais

$$\frac{\partial \chi^2}{\partial c_1}, \frac{\partial \chi^2}{\partial c_2}, \ldots, \frac{\partial \chi^2}{\partial c_n} \tag{7.4}$$

se anularem simultaneamente. Vamos escrever de forma explícita esta condição para o j-ésimo coeficiente

$$\frac{\partial \chi^2}{\partial c_j} = 2 \sum_{i=1}^{m} \left[(y(x_i) - y_i) \frac{\partial y(x_i)}{\partial c_j} \right] = 0\,.$$

Temos que

$$\frac{\partial y(x_i)}{\partial c_j} = f_j(x_i)$$

portanto

$$\frac{\partial \chi^2}{\partial c_j} = 2 \sum_{i=1}^{m} [(y(x_i) - y_i) f_j(x_i)] = 0\,,$$

ou ainda

$$\frac{\partial \chi^2}{\partial c_j} = \sum_{i=1}^{m} [(c_1 f_1(x_i) + c_2 f_2(x_i) + \ldots + c_n f_n(x_i) - y_i) f_j(x_i)] = 0\,.$$

Fazendo a multiplicação e separando os somatórios, temos, finalmente, a seguinte expressão para o j-ésimo coeficiente

$$c_1 \sum_{i=1}^{m} f_1(x_i) f_j(x_i) + c_2 \sum_{i=1}^{m} f_2(x_i) f_j(x_i) + \ldots + c_n \sum_{i=1}^{m} f_n(x_i) f_j(x_i) = \sum_{i=1}^{m} f_j(x_i) y_i \,.$$

que, vale lembrar, vem do fato de termos imposto que a derivada parcial do chi-quadrado com relação a c_j deve ser zero.

Desta maneira, a condição (7.4) define um sistema de n equações, que deve ser resolvido para encontrar os parâmetros do ajuste:

$$\begin{bmatrix} \sum_{i=1}^{m} (f_1(x_i))^2 & \sum_{i=1}^{m} f_1(x_i) f_2(x_i) & \ldots & \sum_{i=1}^{m} f_1(x_i) f_n(x_i) \\ \sum_{i=1}^{m} f_2(x_i) f_1(x_i) & \sum_{i=1}^{m} (f_2(x_i))^2 & \ldots & \sum_{i=1}^{m} f_2(x_i) f_n(x_i) \\ \vdots & \vdots & & \vdots \\ \sum_{i=1}^{m} f_n(x_i) f_1(x_i) & \sum_{i=1}^{m} f_n(x_i) f_2(x_i) & \ldots & \sum_{i=1}^{m} (f_n(x_i))^2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} f_1(x_i) y_i \\ \sum_{i=1}^{m} f_2(x_i) y_i \\ \vdots \\ \sum_{i=1}^{m} f_n(x_i) y_i \end{bmatrix} .$$

Este sistema é usualmente chamado de sistema padrão do problema dos mínimos quadrados. Pode-se mostrar que a matriz de tal sistema tem determinante não nulo, mas ele pode ser (e frequentemente o é) mal-condicionado, de forma que a solução pode ser difícil, ou impossível, de se obter.

Exemplo 1: ajuste de uma reta

Se a função a ser ajustada for uma reta, temos que

$$y(x) = c_1 f_1(x) + c_2 f_2(x) \,,$$

onde $f_1(x) = 1$ e $f_2(x) = x$. Os coeficientes serão dados pelo sistema de ordem 2

$$\begin{bmatrix} \sum_{i=1}^{m} 1 & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} y_i \\ \sum_{i=1}^{m} x_i y_i \end{bmatrix} .$$

Como exemplo, vamos ajustar uma reta ao conjunto de pontos

x	10	20	30	40	50	60	70	80
y	2	5	6	7	10	13	14	15

A tabela abaixo nos ajuda a calcular os coeficientes do sistema padrão:

i	x_i	y_i	$x_i y_i$	x_i^2
1	10	2	20	100
2	20	5	100	400
3	30	6	180	900
4	40	7	280	1600
5	50	10	500	2500
6	60	13	780	3600
7	70	14	980	4900
8	80	15	1200	6400
Σ	360	72	4040	20400

Obtemos o seguinte sistema linar

$$\begin{bmatrix} 8 & 360 \\ 360 & 20400 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 72 \\ 4040 \end{bmatrix} ,$$

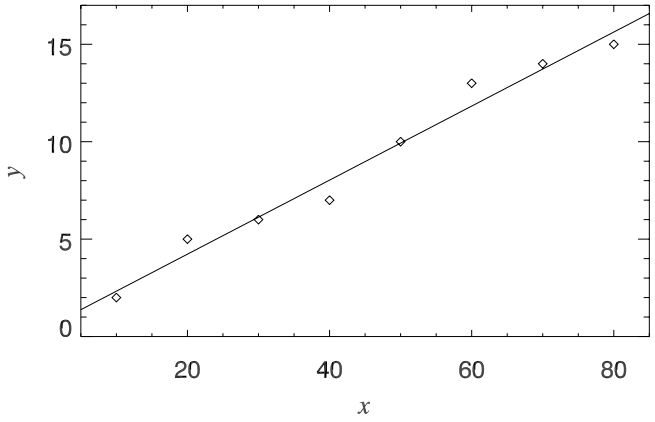


Figura 7.3: Ajuste de uma reta aos pontos listados no exemplo 1.

cuja solução é

$$\begin{aligned} c_1 &= 0.428 \\ c_2 &= 0.190\,. \end{aligned}$$

Desta forma, o melhor ajuste para os pontos dados pelo critério dos mínimos quadrados é a reta $y(x) = 0.428 + 0.190x$.

Exemplo 2: ajuste de uma exponencial

Como ajustar funções não lineares? Algumas funções podem ser rescritas de forma a obter uma outra expressão equivalente e linear. Um exemplo é a função exponencial com dois parâmetros livres

$$y(x) = ae^{-bx}\,,$$

que pode ser reescrita como

$$\ln[y(x)] = \ln(a) - bx \equiv c + dx\,,$$

que é linear nos seus parâmetros c e d.

Importante: preste atenção nos "não-parâmetros", como no caso da função

$$y(x) = ae^{-bx+c}\,.$$

Neste caso, os parâmetros a e c são indistinguíveis. Neste caso, o sistema normal será singular, ou seja, não terá solução (ou terá infinitas soluções).

Exercício:
Ajuste os pontos abaixo a uma exponencial do tipo $y(x) = ae^{-bx}$.

x	0	0.5	1	2
y	2.48	0.901	0.292	3.75×10^{-2}

Resposta: $a = 2.48$ e $b = 2.1$.

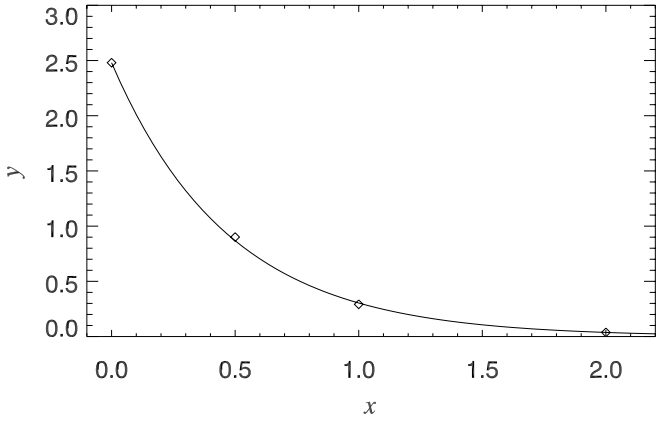


Figura 7.4: Ajuste de uma exponencial aos pontos listados no exemplo 2.

7.2.2 Com pesos

Vamos novamente considerar o caso em que estamos ajustando m pontos observacionais (x_i, y_i), $i = 1, \ldots, m$, com um modelo que tem n parâmetros ajustáveis $(c_j,\, j = 1, \ldots, n)$. Consideremos a questão: "Para um determinado conjunto de parâmetros, qual a probabilidade de que aqueles pontos observacionais em particular tivessem ocorrido?". Se os valores y_i pertencem ao domínio dos reais, estão esta probabilidade é nula, a não ser que tivéssemos adicionado na questão acima a seguinte frase "mais ou menos algum Δy, pequeno e fixo". Vamos, abaixo, sempre considerar tal frase implicitamente. Se a probabilidade de se obter o conjunto de parâmetros for pequena, então podemos concluir que os parâmetros são "improváveis". Ao contrário, intuitivamente sabemos que o conjunto de parâmetros não deveria ser improvável se a escolha tiver sido correta.

Para sermos mais quantitativos, suponhamos que cada ponto y_i tenha um erro observacional (ou de medida) que seja independente das outras observações e que possua uma distribuição Gaussiana (normal) em torno do modelo "verdadeiro" $y(x)$ com um dado desvio padrão σ_i. Desta forma, a probabilidade de, em um determinado experimento, se fazer uma medida de valor y_i com desvio padrão σ_i, para um dado x_i, é

$$P_i(c_1, ..., c_n) = \frac{1}{\sigma_i\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left[\frac{y_i - y(x_i)}{\sigma_i}\right]^2\right)$$

onde o termo $1/\sigma_i\sqrt{2\pi}$ vem do fato de que a integral da probabilidade entre $-\infty$ e ∞ deve ser 1. Recorde que, na expressão acima, $y(x_i)$ representa o valor verdadeiro, ou esperado, para a observação.

A justificativa para assumirmos uma Gaussiana como função de distribuição está no Teorema do Limite Central que afirma (grosso modo) que a densidade de probabilidade de uma variável assume forma Gaussiana se a variável é ela mesma resultante de um grande número de subvariáveis aditivas independentes (ref. Callen).

A probabilidade de obter um conjunto de m pontos observacionais é o produto das probabilidades de cada observação:

$$P(c_1, ..., c_n) = \prod_{i=1}^{m} P_i(c_1, ..., c_n) = \left[\prod_{i=1}^{m}\left(\frac{1}{\sigma_i\sqrt{2\pi}}\right)\right] \exp\left(-\frac{1}{2}\sum_{i=1}^{m}\left[\frac{y_i - y(x_i)}{\sigma_i}\right]^2\right), \quad (7.5)$$

onde o produto de exponenciais foi expresso como a soma dos argumentos. Nesses produtos e somas, quantidades como $1/\sigma_i^2$ atuam como pesos, expressando a contribuição relativa de cada ponto observacional para o resultado final: quando menor for o erro (desvio padrão) maior a importância do dado para o resultado do ajuste.

Vamos assumir que os dados observados são mais prováveis de serem obtidos a partir da distribuição "verdadeira" do que qualquer outra distribuição semelhante com diferentes parâmetros c_j e, portanto, a probabilidade da Eq. (7.5) deve ser máxima. Assim, a estimativa de máxima verossimilhança ("*maximum likehood*") para os c_j são os valores que maximizam a probabilidade da equação (7.5). Pelo fato de o primeiro fator produtório da equação (7.5) ser constante, independente dos valores de c_j, maximizar a probabilidade $P(c_1, ..., c_n)$ é equivalente a minimizar a soma do argumento da exponencial, ou seja, devemos minimizar o χ^2, definido como

$$\chi^2 = \sum_{i=1}^{m} \left[\frac{y_i - y(x_i)}{\sigma_i} \right]^2 .$$

Lembrando que $y(x)$ é escrito de forma geral como $y(x) = c_1 f_1(x) + c_2 f_2(x)_+ \ldots + c_n f_n(x)$, obtemos

$$\chi^2 = \sum_{i=1}^{m} \left[\frac{y_i - c_1 f_1(x) - c_2 f_2(x) - \ldots - c_n f_n(x)}{\sigma_i} \right]^2 . \tag{7.6}$$

Assim, a tarefa de ajuste dos dados será a de encontrar valores c_j que minimizem a soma ponderada dos quadrados dos termos de χ^2.

De forma geral, os valores de χ^2 são afetados por:

- Flutuação nos valores medidos y_i.

- Valores das incertezas σ_i: valores incorretos de σ_i levarão a valores incorretos de χ^2.

- A seleção da função analítica $y(x)$.

Minimização de χ^2: ajuste de uma reta

Sem perda de generalidade, vamos considerar o caso particular em que aproximamos os pontos por uma reta, ou seja, $y(x) = c_1 + c_2 x$. Como feito na seção anterior, para encontrar os parâmetros c_1 e c_2 que fornecem o mínimo valor para χ^2 igualamos a zero as derivadas parciais de χ^2 com respeito a cada um deles:

$$\frac{\partial \chi^2}{\partial c_1} = \frac{\partial}{\partial c_1} \sum_{i=1}^{m} \left[\frac{1}{\sigma_i^2} (y_i - c_1 - c_2 x_i)^2 \right] = -2 \sum_{i=1}^{m} \left[\frac{1}{\sigma_i^2} (y_i - c_1 - c_2 x_i) \right] = 0 \,,$$

$$\frac{\partial \chi^2}{\partial c_2} = \frac{\partial}{\partial c_2} \sum_{i=1}^{m} \left[\frac{1}{\sigma_i^2} (y_i - c_1 - c_2 x_i)^2 \right] = -2 \sum_{i=1}^{m} \left[\frac{x_i}{\sigma_i^2} (y_i - c_1 - c_2 x_i) \right] = 0 \,.$$

Essas equações podem ser rearranjadas como um sistema de equações lineares

$$\begin{aligned} c_1 \sum_{i=1}^{m} \frac{1}{\sigma_i^2} + c_2 \sum_{i=1}^{m} \frac{x_i}{\sigma_i^2} &= \sum_{i=1}^{m} \frac{y_i}{\sigma_i^2} \\ c_1 \sum_{i=1}^{m} \frac{x_i}{\sigma_i^2} + c_2 \sum_{i=1}^{m} \frac{x_i}{\sigma_i^2} &= \sum_{i=1}^{m} \frac{x_i y_i}{\sigma_i^2} \,. \end{aligned}$$

Usando a regra de Cramer

$$c_1 = \frac{1}{\Delta}\begin{vmatrix} \sum_{i=1}^m \frac{y_i}{\sigma_i^2} & \sum_{i=1}^m \frac{x_i}{\sigma_i^2} \\ \sum_{i=1}^m \frac{x_i y_i}{\sigma_i^2} & \sum_{i=1}^m \frac{x_i}{\sigma_i^2} \end{vmatrix} = \frac{1}{\Delta}\left(\sum_{i=1}^m \frac{x_i}{\sigma_i^2}\sum_{i=1}^m \frac{y_i}{\sigma_i^2} - \sum_{i=1}^m \frac{x_i}{\sigma_i^2}\sum_{i=1}^m \frac{x_i y_i}{\sigma_i^2}\right),$$

$$c_2 = \frac{1}{\Delta}\begin{vmatrix} \sum_{i=1}^m \frac{1}{\sigma_i^2} & \sum_{i=1}^m \frac{y_i}{\sigma_i^2} \\ \sum_{i=1}^m \frac{x_i}{\sigma_i^2} & \sum_{i=1}^m \frac{x_i y_i}{\sigma_i^2} \end{vmatrix} = \frac{1}{\Delta}\left(\sum_{i=1}^m \frac{1}{\sigma_i^2}\sum_{i=1}^m \frac{x_i y_i}{\sigma_i^2} - \sum_{i=1}^m \frac{x_i}{\sigma_i^2}\sum_{i=1}^m \frac{y_i}{\sigma_i^2}\right),$$

onde

$$\Delta = \begin{vmatrix} \sum_{i=1}^m \frac{1}{\sigma_i^2} & \sum_{i=1}^m \frac{x_i}{\sigma_i^2} \\ \sum_{i=1}^m \frac{x_i}{\sigma_i^2} & \sum_{i=1}^m \frac{x_i^2}{\sigma_i^2} \end{vmatrix} = \sum_{i=1}^m \frac{1}{\sigma_i^2}\sum_{i=1}^m \frac{x_i^2}{\sigma_i^2} - \left(\sum_{i=1}^m \frac{x_i}{\sigma_i^2}\right)^2 .$$

Para o caso particular onde todas as incertezas são iguais $\sigma_i = \sigma,\ i = 1, \ldots, m$ as fórmulas se reduzem às já estudadas sem pesos.

As incertezas dos parâmetros são dadas por (Bevington & Robinson)

$$\sigma_{c_1}^2 = \frac{1}{\Delta}\sum_{i=1}^N \frac{x_i^2}{\sigma_i^2}$$

$$\sigma_{c_2}^2 = \frac{1}{\Delta}\sum_{i=1}^N \frac{1}{\sigma_i^2}\,.$$

Exemplo 3: ajuste de uma reta

Seja o conjunto de pontos

x	10	20	30	40	50	60	70	80
y	2	12	6	7	10	13	7	15
σ_y	0.5	2.8	0.4	0.6	0.6	0.6	3.5	0.7

Neste conjunto, os pontos (20, 12) e (70, 7) são *outliers*, ou seja, estão fora da tendência geral indicada pelos outros pontos. Por este motivo, têm um erro associado muito maior que os demais. Se fizermos um ajuste destes pontos sem considerar os erros, obtemos o resultado mostrado na Figura 7.5-a. Vemos que este ajuste não descreve bem a tendência geral esperada quando consideramos apenas os pontos com menor erro. O procedimento correto é fazermos o ajuste levando em conta os erros. O resultado é mostrado na Figura 7.5-b. Os valores obtidos para os parâmetros livres são

$$c_1 = -0.10 \pm 0.32$$
$$c_2 = 0.199 + / - 0.004\,.$$

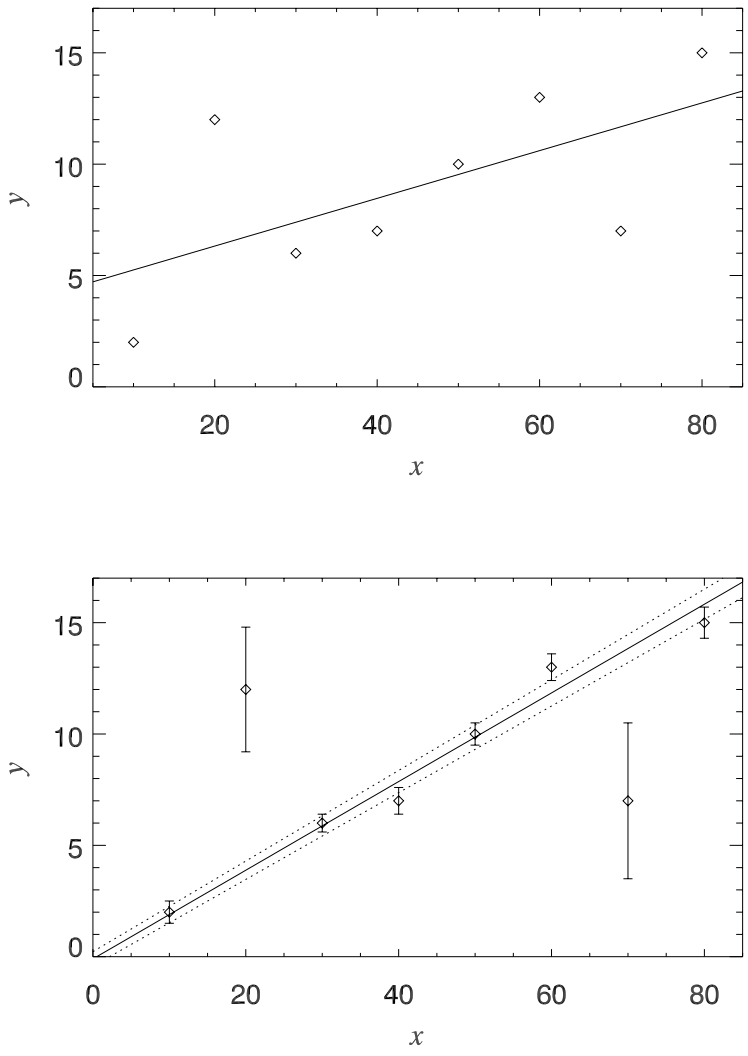


Figura 7.5: a) Acima: ajuste sem considerar os erros dos dados. b) Abaixo: ajuste feito considerando os erros. As linhas pontilhadas correspondem aos limites do erro estimado para o ajuste, e foram traçadas considerando os maiores e menores valores de c_1 e c_2 dentro da incerteza.

Capítulo 8

Integração Numérica

8.1 Introdução [1]

Este capítulo trata de métodos numéricos para o cálculo de integrais definidas. O Cálculo nos ensina que, para se obter

$$\int_a^b f(x)dx,$$

basta achar uma primitiva, isto é, uma função $F(x)$ tal que $F'(x) = f(x)$, de forma que

$$\int_a^b f(x)dx = F(b) - F(a)\,.$$

Uma função f é, em geral, dada por uma "fórmula", que nada mais é do que a combinação finita, *via* somas, multiplicações, divisões e composições de funções elementares. As funções elementares são as usuais: potências de x (negativas e positivas), funções trigonométricas e suas inversas, logaritmos e exponenciais.

Entretanto, no mundo abstrato de todas as funções possíveis, essas funções formam apenas umas minúscula parte. Em outras palavras, a grande maioria das funções não tem uma fórmula que as represente, embora nas aplicações do mundo real os modelos frequentemente conduzam a funções descritas por meio de fórmulas.

Mesmo se nos restringirmos apenas às funções dadas por fórmulas, acabaremos por nos deparar com um fato matemático: *nem todas elas admitem uma primitiva que também seja escrita como combinação (finita) de funções elementares!*

É claro que existe o recurso de se escrever a primitiva F como uma combinação infinita de funções elementares, por exemplo através de uma série de potências

$$F(x) = \sum_{k=0}^{\infty} c_k x^k\,.$$

Isto é possível (em muitos casos de forma até razoavelmente fácil), mas com dois inconvenientes: primeiro, quando formos avaliar $F(a)$ e $F(b)$ através da série (ou da fórmula infinita) pode ser necessária uma quantidade tão grande de termos (ou operações) que inviabilize ou torne muito lento o cálculo. Além disso, nem sempre séries de potência convergem para todos os valores de x, o que exigiria uma análise criteriosa do alcance dessa convergência, em cada caso.

[1] Adaptado de Asano & Coli 2009

De outra parte, é preciso também dispor de instrumentos para estimar integrais a partir de dados experimentais. As aplicações mais óbvias se encontram no cálculo de comprimentos, áreas, volumes, massa, centro de massa, distância percorrida, tempo decorrido, etc. No que segue, discutiremos alguns exemplos onde a integração numérica se faz necessária, ora por se tratar de medida experimental ora porque não há primitiva elementar da função que se quer integrar.

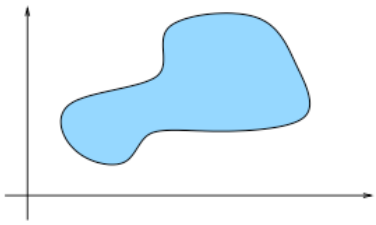
Neste capítulo estudaremos métodos para a integração numérica de funções conhecidas que não possuem primitivas convenientemente representáveis, ou de funções desconhecidas, das quais apenas um conjunto de pontos é conhecido.

Em linhas gerais, os métodos que utilizaremos baseiam-se em somar o valor do integrando para uma sequência de valores da abcissa dentro dos limites da integração. O objetivo é determinar a integral tão acuradamente quanto possível com o menor número de avaliações da função.

No que segue listamos alguns exemplos de problemas que só podem ser resolvidos usando-se um método numérico de de integração

8.1.1 Cálculo de áreas

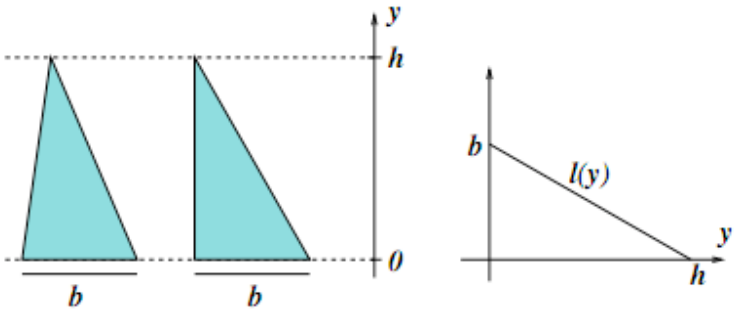
Gostaríamos de um método sistemático para estimar a área de figuras planas como a mostrada ao lado (poderia ser uma ilha, por exemplo). Para isso, vamos nos basear no Princípio de Cavalieri, que diz: *"dados dois conjuntos A e B, se houver uma linha L tal que toda perpendicular a L cruze A e B em intervalos de tamanhos iguais, então A e B têm a mesma área."*



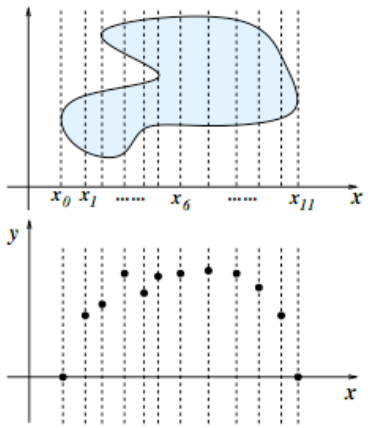
Por exemplo, os triângulos da figura abaixo (à esquerda) têm áreas iguais, pois cada reta R horizontal, à altura y, cruza os triângulos em segmentos de tamanho igual a $l(y)$. Para entender porque $l(y)$ é igual para os dois triângulos, observe que em ambos $l(y)$ varia como uma função afim ("linearmente"), em $y = 0$ tem-se $l(0) = b$ (os triângulos têm bases de igual tamanho) e em $y = h$ tem-se $l(h) = 0$ (os triângulos têm alturas iguais). Portanto a função $l(y)$ tem o aspecto mostrado à direita, na figura.

Isso explica porque todos os triângulos com base e altura iguais têm a mesma área, que pode ser obtida de um deles, por exemplo o da direita. Essa área vale $\frac{1}{2}bh$, e o leitor pode observar que essa também é a área sob o gráfico de $l(y)$ (observação que será importante logo adiante).

O Princípio de Cavalieri tem uma formulação análoga para volumes. Dois sólidos S e T terão mesmo volume se houver uma linha L tal que todo *plano* perpendicular a L cruze S e T em regiões de áreas iguais. Para o leitor que ainda não acreditou nesse princípio, imagine uma pilha de cartas com um arame passando no meio, e então incline e retorça o arame, de forma que a pilha fique desalinhada. As duas pilhas continuam tendo a mesma altura, a área de cada corte é a mesma, e o volume (que é a soma dos volumes "infinitesimais"das cartas) se mantém.



O que podemos fazer com uma figura plana em geral é criar uma segunda figura com a mesma área apoiada no eixo horizontal. Na prática, temos que fazer isso para um número discreto de cortes verticais: medimos o comprimento do corte e transferimos esse valor para a segunda figura. Assim, a segunda figura é um esboço do gráfico "Comprimento do corte vs. Posição do corte", mais precisamente é a região compreendida entre esse gráfico e a linha horizontal. Quando o corte ocorrer em dois intervalos separados a altura do gráfico será igual à soma dos comprimentos das duas intersecções.



Ao final, teremos uma sequência de pontos $x_0, x_1, \ldots, x_n$, que fornecem a posição de cada corte, e valores correspondentes $y_0, y_1, \ldots, y_n$, que são os respectivos comprimentos de cada corte. Esses dadods é que serão usados para se fazer a integração.

O curioso é que o mesmo tipo de "coleta de dados"será feito para a integração de uma função $f(x)$ dada por uma fórmula. Se a integração se der no intervalo $[a, b]$, então deve-se dividir o intervalo por um partição

$$a = x_0 < x_1 < x_2 < \ldots < x_n = b$$

e tomar os valores da função nos extremos dos intervalos da partição:

$$y_0 = f(x_0), y1 = f(x_1), \ldots, y_n = f(x_n)$$

(que podem até ser negativos). A partir desses dados, a maneira de se proceder será a mesma, tanto no caso 'experimental' como no caso 'teórico'. A única diferença é que no caso 'teórico' nós teremos, na maioria dos casos, uma maneira de delimitar o erro cometido na integração.

O *volume de um lago ou de uma montanha* também é passível de ser estimado usando esse tipo de dados. Pode-se fazer isso em duas etapas. Primeiramente, escolhe-se uma

direção (x, por exemplo) onde se posicionarão, perpendicularmente, as retas dos "cortes". Para cada corte do lago, posicionado em x_i, estima-se sua área $A(x_i)$, usando dados (y_i, z_i). Depois estima-se a integral da função "área do corte", usando-se os dados $(x_i, A(x_i))$, que resulta no volume.

8.1.2 Comprimento de curvas e gráficos

Considere o seguinte problema: *"calcular o comprimento do gráfico da função f entre a e b"*. Se a função f for diferenciável, esse problema remete a uma integral.

Para entender melhor, tentemos aproximar a curva por pequenos segmentos de reta e seu comprimento pela soma dos tamanhos desses segmentos. Como sempre dividimos o intervalo $[a, b]$ com uma partição $a = x_0 < x_1 < \ldots < x_n = b$ e em cada intervalo $[x_i, x_{i+1}]$ $(i = 0, \ldots, n-1)$ aproximamos a função pelo segmento de reta que unes os pontos $(x_i, f(x_i))$ e $(x_{i+1}, f(x_{i+1}))$. Pelo Teorema de Pitágoras, esse segmento tem tamanho igual a

$$\sqrt{(x_{i+1} - x_i)^2 + (f(x_{i+1}) - f(x_i))^2}$$

Para simplificar um pouco, podemos supor que todos os intervalos tenham o mesmo tamanho Δx. Além disso, aproximamos a diferença $f(x_{i+1}) - f(x_i)$ por $f'(x_i)\Delta x$, de forma que somando para todos os segmentos obtenhamos, aproximadamente,

$$\sum_{i=0}^{n-1} \Delta x \sqrt{1 + f'(x_i)^2}$$

Fazendo Δx ir a zero estaremos, por um lado, fazendo com que a soma dos comprimentos dos segmentos esteja cada vez mais próxima do comprimento verdadeiro da curva e, por outro lado, fazendo com que a aproximação pela derivada seja cada vez mais fidedigna. No limite, teremos um número que é ao mesmo tempo o comprimento da curva e também a integral

$$\int_a^b \sqrt{1 + f'(x)^2} dx$$

O gráfico de f entre a e b é um caso particular de curva no plano. Cada ponto dessa curva pode ser obtido tomando-se t no intervalo $[a, b]$ e então o ponto $(t, f(t))$. Podemos imaginar esse processo como uma função com domínio $[a, b]$ e contradomínio R^2, que leva t em $(t, f(t))$.

8.2 Fórmulas clássicas para abcissas igualmente espaçadas

Alguns dos métodos que usaremos abaixo originam-se das chamadas *fórmulas clássicas de quadratura* (quadratura é sinônimo de cálculo de uma integral). Essas fórmulas procuravam aproximar integral a partir de alguns poucos pontos do integrando (muito úteis se as contas são feitas todas com lápis e papel!). Hoje boa parte destas fórmulas são peças de museu, e têm pouca utilidade prática. Exceções são a regra do trapézio e a regra de Simpson.

8.2.1 Regra do trapézio

$$\int_{x_0}^{x_1} f(x)dx = h\left[\frac{1}{2}f(x_0) + \frac{1}{2}f(x_1)\right] - \frac{h^3}{12}f''(\xi)\,, \tag{8.1}$$

onde $h = x_1 - x_0$ e ξ é um número no intervalo $[x_0, x_1]$. O termo à direita permite estimar o erro cometido na estimativa da integral. Como não se sabe o ξ para o qual se deve avaliar a segunda derivada, o termo é na verdade desconhecido e por esse motivo é frequentemente escrito como

$$\int_{x_0}^{x_1} f(x)dx = h\left[\frac{1}{2}f(x_0) + \frac{1}{2}f(x_1)\right] + \mathcal{O}(h^3 f'')\,, \tag{8.2}$$

onde o símbolo $\mathcal{O}$ — *ordem* — é comumente empregado em matemática para se referir à ordem de grandeza de um termo em uma equação.

A figura abaixo ilustra a regra do trapézio. Sendo uma fórmula com dois pontos, ela é exata para polinômios de ordem 1. Pelo termo do erro na Eq. (8.1) vemos que se a função tem uma concavidade para cima (o que significa que $f'' > 0$ no intervalo de integração), então o erro tem um valor negativo e a regra do trapézio superestima a integral.

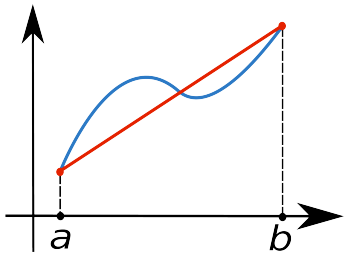


Figura 8.1: Regra do trapézio (fonte [a]).

Estimativa do erro da regra do trapézio

Avaliaremos agora o cálculo do erro da regra do trapézio. Pelo Teorema Fundamental do Cálculo, existe uma função F tal que $F'(x) = f(x)$. Seja I o valor correto da integral de $f(x)$ em $[x_0, x_1]$ e T o valor da estimativa pela regra do trapézio. O erro na Regra do Trapézio será dado por

$$I - T = \int_{x_0}^{x_1} F'(x)dx - \frac{h}{2}[f(x_0) + f(x_1)] \tag{8.3}$$

$$= F(x_1) - F(x_0) - \frac{h}{2}[f(x_0) + f(x_1)] \tag{8.4}$$

$$= F(x_0 + h) - F(x_0) - \frac{h}{2}[f(x_0) + f(x_0 + h)]\,. \tag{8.5}$$

Vamos usar a expansão em séries de Taylor

$$f(x_0 + h) = f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) + \dots\,.$$

para expandir o primeiro e o último termo da Eq. (8.5). Obtemos

$$F(x_0 + h) = F(x_0) + hf(x_0) + \frac{h^2}{2}f'(x_0) + \frac{h^3}{6}f''(x_0) + \dots$$

e

$$f(x_0 + h) = f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) + \frac{h^3}{6}f'''(x_0) + \dots\,.$$

Substituindo as duas expressões acima em Eq. (8.5) obtemos

$$I - T = -\frac{h^3}{12}f''(x_0) + \mathcal{O}(h^4)\,. \tag{8.6}$$

Obtemos assim o termo da Eq. (8.1).

8.2.2 Regra de Simpson

A regra de Simpson usa três pontos do intervalo $[x_0, x_2]$ separados por uma distância h:

$$\int_{x_0}^{x_2} f(x)dx = h\left[\frac{1}{3}f(x_0) + \frac{4}{3}f(x_1) + \frac{1}{3}f(x_2)\right] + \mathcal{O}(h^5 f^{(4)}), \tag{8.7}$$

onde $f^{(4)}$ é a quarta derivada da função f calculada em um ponto desconhecido do intervalo. Apesar de ser uma fórmula com 3 pontos, que deveria em princípio ser exata para polinômios de ordem 2, a regra de Simpson é exata para qualquer polinômio de ordem 3 (ou menos). Isso é fácil de ser verificado pelo termo do erro, que depende da quarta derivada da função: a quarta derivada de um polinômio de ordem 3 ou menos é zero.

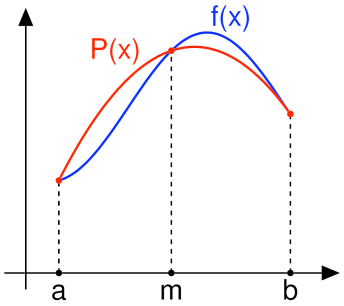


Figura 8.2: Regra de Simpson (fonte [a]).

A Regra de Simpson é fácil de ser derivada, apesar de ser um pouco trabalhosa algebricamente. Matematicamente, ela se origina do *ajuste* da função $f(x)$ por um polinômio de ordem 2 ($P(x)$ na Figura 8.2).

Para se obter a fórmula de Simpson, pode-se usar a fórmula de Lagrange para interpolação polinomial (ver Capítulo 6).

Derivação do erro da regra de Simpson

É possível mostrar, usando-se o mesmo procedimento usado acima para a regra do trapézio, que o erro de uma integral I estimada pelo método de Simpson, S, será dado por.

$$I - S = -\frac{h^5}{90}f^{(4)}(x_0) + \dots . \tag{8.8}$$

Vamos demonstrar a expressão acima. Sabendo que $F'(x) = f(x)$:

$$I - S = \int_a^b F'(x)dx - \frac{1}{3}h[f(x_0) + 4f(x_1) + f(x_2)]$$

$$= F(x_2) - F(x_0) - \frac{1}{3}h[f(x_0) + 4f(x_0 + h) + f(x_0 + 2h)]$$

$$= F(x_0) + 2hf(x_0) + \frac{(2h)^2}{2}f'(x_0) + \frac{(2h)^3}{3!}f''(x_0) + \frac{(2h)^4}{4!}f'''(x_0) + \frac{(2h)^5}{5!}f^{(4)}(x_0) + \dots$$

$$-F(x_0) - \frac{h}{3}f(x_0) - \frac{4}{3}h[f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) + \frac{h^3}{3!}f'''(x_0) + \frac{h^4}{4!}f^{(4)}(x_0) + \dots]$$

$$-\frac{1}{3}h[f(x_0) + 2hf'(x_0) + \frac{(2h)^2}{2}f''(x_0) + \frac{(2h)^3}{3!}f'''(x_0) + \frac{(2h)^4}{4!}f^{(4)}(x_0) + \dots]$$

Na expressão acima, os termos com $f(x_0)$, $f'(x_0)$, $f''(x_0)$, e $f'''(x_0)$ se anulam, restando apenas termos que dependem de $f^{(4)}$

$$I - S = \left[\frac{(2h)^5}{5!} - \left(\frac{4h^5}{3\cdot 4!} + \frac{16h^5}{3\cdot 4!}\right)\right] f^{(4)}(x_0) + \ldots = -\frac{1}{90}h^5 f^{(4)} f(x_0) + \ldots$$

Portanto, concluímos que o erro cometido pelo método de Simpson é $\mathcal{O}(h^5)$.

8.3 Métodos Numéricos de Integração

Queremos resolver o seguinte problema: *dada uma função f:[a,b]→ R, achar a integral de f nesse intervalo, denotada por*

$$\int_a^b f(x)dx\,.$$

8.3.1 Integração por retângulos

A forma mais simples de estimarmos a integral de uma função $f(x)$ no intervalo $[x_0, x_n]$ é aproximarmos a integral por uma soma de retângulos. Vamos dividir o intervalo $[x_0, x_n]$ em n intervalos iguais. Seja $h = (x_n - x_0)/n$ o tamanho dos subintervalos. Temos

$$x_i = x_0 + ih, \quad i = 0, 1, \ldots, n\,.$$

A integral será dada por

$$I = \int_{x_0}^{x_n} f(x)dx = h\sum_{i=0}^{n-1} f(x_i)\,. \tag{8.9}$$

Essa expressão pode facilmente ser generalizada para um conjunto de n pontos não igualmente espaçados. Neste caso.

$$I = \sum_{i=0}^{n-1} h_i f(x_i)\,, \tag{8.10}$$

onde $h_i = x_i - x_{i-1}$.

Note que a fórmula das Eqs. (8.9) e (8.10) é uma fórmula *semi-aberta*, ou seja, para computar a integral no intervalo $[x_0, x_n]$ *não foi usado o valor de* $f(x_n)$. A maioria dos demais métodos que veremos abaixo usam fórmulas fechadas, em que são usados os valores da função nos extremos do intervalo. Existem também métodos que usam *fórmulas abertas* em que não se usam nenhum dos extremos de integração.

Fórmulas abertas ou semi-abertas têm uma propriedade importante: elas permitem calcular a integral no caso em que a função é mal-comportada em um ou ambos os intervalos de integração (por exemplo, no caso em que f vai para algum tipo de singularidade em um dos extremos).

Vamos estimar o erro que cometemos ao aproximar a integral por apenas um retângulo. Como antes, seja I_i o valor da integral no intervalo $[x_i, x_{i+1}]$ e R_i o valor da aproximação por um retângulo. Vamos expandir a função em série de Taylor em torno do ponto x_i,

$$f(x) = f(x_i) + (x - x_i)f'(x_i) + \ldots\,,$$

e efetuar a integral explicitamente

$$I_i = \int_{x_i}^{x_{i+1}} f(x)dx \tag{8.11}$$

$$= \int_{x_i}^{x_{i+1}} f(x_i)dx + \int_{x_i}^{x_{i+1}} (x - x_i)f'(x_i)dx + \dots \tag{8.12}$$

$$= R_i + \frac{1}{2}f'(x_i)(x - x_i)^2\Big|_{x_i}^{x_{i+1}} + \dots \tag{8.13}$$

$$= R_i + \frac{1}{2}h^2 f'(x_i) + \dots . \tag{8.14}$$

Assim, o erro na integral por um retângulo é

$$I_i - R_i \approx \mathcal{O}(h^2) .$$

Vamos agora estimar o erro para uma integral efetuada pela soma de n retângulos. Nesse caso o erro total será aproximadamente a soma dos erros cometidos em cada subintervalo, ou seja

$$I - R \approx \sum_{i=0}^{n-1} \mathcal{O}(h^2) = \mathcal{O}(nh^2) .$$

Como $h = (x_n - x_0)/n$, temos que o erro total será

$$I - R \approx \sum_{i=0}^{n-1} \mathcal{O}\left[\frac{(x_n - x_0)^2}{n}\right] .$$

Ou seja, o erro cai de forma linear à medida que aumentamos o número n de intervalos, e tende a 0 quando $n \to \infty$.

Um método de integração é dito de ordem m se o erro cai com o aumento do número de subintervalos como

$$\mathcal{O}\left(\frac{1}{n^m}\right) .$$

O método dos retângulos é o de mais baixa ordem. A origem da ineficiência do método está no uso de retângulos para aproximar a função. Nos métodos que veremos adiante usaremos aproximações cada vez melhores para $f(x)$, o que resultará em métodos de ordem maior.

A vantagem do método dos retângulos está na grande facilidade em implementá-lo. Por exemplo, no código abaixo, `h` é o tamanho dos subintervalos e `func` é um array que armazena os valores da função $f(x)$ para os $n-1$ valores da abcissa. Neste caso, a integral é calculada com apenas uma linha de código:

```
REAL, DIMENSION(N-1) :: func
REAL :: h, integral
...
integral = h*SUM(func)
...
```

No caso de abcissas não igualmente espaçadas, armazenamos os valores dos subintervalos em um array (digamos harray). Neste caso, a integral seria

```
REAL, DIMENSION(N-1) :: func, harray
REAL :: integral
...
integral = SUM(harray*func)
...
```

8.3.2 Regra do ponto médio

Este método corresponde a uma melhoria simples e eficiente do método dos retângulos. Consideremos um subintervalo $[x_i, x_{i+1}]$ e seu ponto médio

$$\bar{x}_i = \frac{x_i + x_{i+1}}{2}\,.$$

Neste caso, a integral da função $f(x)$ neste subintervalo pode ser aproximada por

$$M_i = \int_{x_i}^{x_{i+1}} f(x)dx = h_i f(\bar{x}_i)\,,$$

e a integral no intervalo $[x_0, x_n]$ será simplesmente

$$I \approx M = \sum_{i=1}^{n} M_i = \sum_{i=1}^{n} h_i f(\bar{x}_i)\,. \tag{8.15}$$

Para avaliarmos o erro da regra do ponto médio vamos novamente considerar o erro cometido quando aproximamos a integral entre $[x_i, x_{i+1}]$ com apenas um retângulo. Vamos expandir $f(x)$ em série de Taylor em torno do ponto médio $\bar{x}_i$

$$f(x) = f(\bar{x}_i) + (x - \bar{x}_i)f'(\bar{x}_i) + \frac{1}{2}(x - \bar{x}_i)^2 f''(\bar{x}_i) + \frac{1}{3!}(x - \bar{x}_i)^3 + \dots\,.$$

Integrando a expressão acima entre $[x_i, x_{i+1}]$ obtemos

$$\begin{aligned} I_i &= \int_{x_i}^{x_{i+1}} f(x)dx \\ &= \int_{x_i}^{x_{i+1}} f(\bar{x}_i)dx + \int_{x_i}^{x_{i+1}} (x - \bar{x}_i)f'(\bar{x}_i)dx + \frac{1}{2}\int_{x_i}^{x_{i+1}} (x - \bar{x}_i)^2 f''(\bar{x}_i)dx + \\ &\quad \frac{1}{3!}\int_{x_i}^{x_{i+1}} (x - \bar{x}_i)^3 f'''(\bar{x}_i)dx + \dots\,. \end{aligned}$$

Note que

$$\int_{x_i}^{x_{i+1}} (x - \bar{x}_i)^p dx = \begin{cases} h_i & \text{se } p = 0 \\ 0 & \text{se } p = 1 \\ h_i^3/12 & \text{se } p = 2 \\ 0 & \text{se } p = 3 \end{cases}$$

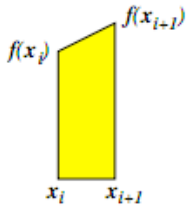
de forma que I_i reduz-se à

$$I_i = M_i + \frac{1}{24}h_i^3 f''(\bar{x}_i) + \dots\,.$$

Vemos, assim, que o erro de I_i é $\mathcal{O}(h^3)$. Considerando-se a integral feita em n intervalos de tamanho h, concluímos que o método do ponto médio é de ordem 2.

Note que a regra do ponto médio é uma fórmula aberta, pois não se usam os valores da função nos extremos do intervalo.

8.3.3 O método dos trapézios



Vamos estender a regra dos trapézios (Eq. 8.1) para avaliarmos a integral de uma função calculada em $n+1$ pontos dentro do intervalo $[x_0, x_n]$. Inicialmente aplicamos a regra do trapézio ao subintervalo i

$$T_i = \int_{x_i}^{x_{i+1}} f(x)dx = \frac{h}{2}\left[f(x_i) + f(x_{i+1})\right] - \frac{h^3}{12}f''(\xi)\,,$$

e em seguida somamos para todos os subintervalos

$$I \approx T = \sum_{i=1}^{n} T_i = h\left[\frac{1}{2}f(x_0) + f(x_1) + \cdots + f(n-1) + \frac{1}{2}f(n)\right] + \mathcal{O}\left(\frac{(x_n - x_0)^3}{12n^2}f''(\xi)\right)\,,$$

ou, em uma forma mais compacta

$$I = h\left[\frac{1}{2}f(x_0) + \sum_{i=1}^{n-1} f(x_i) + \frac{1}{2}f(n)\right] + \mathcal{O}\left(\frac{1}{n^2}\right)\,.$$

Vemos que a regra do trapézio estendida tem a mesma ordem que o método do ponto médio.

Exemplo: Calcular a integral

$$\int_0^1 e^{-x^2}dx$$

usando o Método do Trapézio tomando $n = 2, 4, 8, 16$. o valor da integral é 0.746824133, com precisão até 9 decimais.

Para $n = 2, h = 0.5$:

$$\int_0^1 f(x)dx = \frac{0.5}{2}[f(0) + 2f(0.5) + f(1)] = 0.7313702$$

Para $n = 4, h = 0.25$:

$$\int_0^1 f(x)dx = \frac{0.25}{2}[f(0) + 2f(0.25) + 2f(0.5) + 2f(0.75) + f(1)] = 0.7429840$$

Para $n = 8, h = 0.125$:

$$\int_0^1 f(x)dx = \frac{0.125}{2}[f(0) + 2f(0.125) + 2\ldots + f(1)] = 0.7458655$$

Para $n = 16, h = 0.0625$:

$$\int_0^1 f(x)dx = \frac{0.0625}{2}[f(0) + 2f(0.0625) + 2\ldots + f(1)] = 0.7465846$$

8.3.4 O método de Simpson

Como no caso dos trapézios, é muito simples estender a regra de Simpson para o caso de um intervalo subdividido em n subintervalos. Aplicamos a regra de Simpson ao par de subintervalos i e $i+1$ (note que a regra de Simpson usa três pontos)

$$S_i = \int_{x_i}^{x_{i+2}} f(x)dx = \frac{h}{3}\left[f(x_i) + 4f(x_{i+1} + f(x_{i+2})\right] + \mathcal{O}(h^5 f^{(4)}),$$

e em seguida somamos para todos os subintervalos, obtendo

$$\begin{aligned} S =& \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2) + f(x_2) + 4f(x_3) + f(x_4) + \ldots\right. \\ & \left. + f(n-2) + 4f(n-1) + f(n)\right] + \mathcal{O}\left[\frac{(x_n - x_0)^5}{n^4}\right] \\ =& \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \ldots\right. \\ & \left. + 2f(n-2) + 4f(n-1) + f(n)\right] + \mathcal{O}\left[\frac{(x_n - x_0)^5}{n^4}\right]. \end{aligned}$$

Note que esta regra requer que n seja par. Em uma notação mais compacta, a regra de Simpson estendida fica

$$S = \frac{h}{3}\left[f(x_1) + f(x_n) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + 2\sum_{j=1}^{n/2-1} f(x_{2j})\right] + \mathcal{O}\left(\frac{1}{n^4}\right). \tag{8.16}$$

A Eq. (8.16) mostra uma importantíssima propriedade da regra de Simpson: sendo um método de ordem 4, a precisão de integral aumenta com a *quarta potência de n*!

Exemplo: calcular a integral

$$\int_0^1 e^{-x^2} dx$$

usando o método de Simpson, para $n = 2$ e $n = 4$.

Para $n = 2, h = 0.5$:

$$\int_0^1 f(x)dx = \frac{h}{3}[f_0 + 4f_1 + f_2] = \frac{h}{3}[f(0) + 4f(0.5) + f(1)] = 0.747180$$

$N = 4, h = 0.5$:

$$\int_0^1 f(x)dx \approx \frac{h}{3}[f_0 + 4f_1 + 2f_2 + 4f_3 + f_4]$$

$$\approx \frac{h}{3}[f(0) + 4f(0.25) + 2f(0.5) + 4f(0.75) + f(1)] = 0.746855$$

8.3.5 Fórmulas de Newton-Cotes

Os métodos do trapézio e de Simpson são dois casos de uma série de fórmulas de integração chamados de Métodos de **Newton-Cotes**. Usando o método do trapézio aproximamos segmentos de curva $f(x)$ por linhas retas que então definem trapezóides; no Método de Simpson aproximamos a função $f(x)$ por parábolas. Para melhores aproximações podemos usar curvas cúbicas, quárticas e assim por diante e todos esses são Newton-Cotes.

- Trapézio → dois pontos, 1 intervalo, reta
- Simpson → três pontos, 2 intervalos, parábola, número ímpar de pontos.
- Aprox. cúbica → quatro pontos, 3 intervalos, cúbica (Simpson 3/8), número de pontos $3n + 1$: 4,7,10...

$$S_{3/8} = \frac{3}{8}h[f_1 + 3f_2 + 3f_3 + f_4]$$

Até aqui só apresentamos as fórmulas fechadas de Newton-Cotes *("closed Newton-Cotes formulas")* adequadas aos casos onde a função seja bem definida nas extremidades de $[a, b]$. Para os casos em que a função $f(x)$ não é bem definida em x_0 e/ou x_n (e.g. divergência $\pm\infty$) existem outras fórmulas chamadas abertas *("open Newton-Cotes formulas")*. Mais detalhes e as expressões destas fórmulas veja em *Numerical Recipes*.

8.3.6 Método de Romberg

Este método também é conhecido por *"Romberg Integration"* ou *"Romberg Extrapolation"*. Até aqui, vimos que o erro total para trapézios é da ordem de $\mathcal{O}(h^2)$ (lembre-se que a ordem do erro cai em uma unidade quando consideramos o erro advindo de se aproximar a integral por uma soma de trapézios). Um exame mais detalhado, considerando os termos de ordem superior da série de Taylor, fornece

$$I - T(h) = Ah^2 + Bh^4 + \dots . \tag{8.17}$$

Vamos comparar esse resultado com o obtido na iteração anterior

$$I - T(2h) = A(2h)^2 + B(2h)^4 + \dots \tag{8.18}$$

Multiplicando a Eq. (8.17) por 4 e subtraindo a Eq. (8.18), obtemos

$$\begin{aligned} 4I - 4T(h) &= 4Ah^2 + 4Bh^4 + \dots \\ -I + T(2h) &= -A(2h)^2 - 16Bh^4 + \dots \end{aligned}$$

$$\begin{aligned} 3I - [4T(h) - T(2h)] &= -12Bh^4 + \dots \\ I - \underbrace{[4T(h) - T(2h)]/3}_{\equiv S(h)} &= -12Bh^4 + \dots \end{aligned}$$

Esta aproximação

$$S(h) = \frac{4T(h) - T(2h)}{3} = T(h) + \frac{T(h) - T(2h)}{3}\,, \tag{8.19}$$

difere do valor exato da integral por um termo $\mathcal{O}(h^4)$, enquanto $T(h)$ e $T(2h)$ diferem por um termo $\mathcal{O}(h^2)$. Desta forma, a combinação de estimativas consecutivas da integral pelo método dos trapézios vai *cancelar os termos de ordem inferior do erro*! A ordem do termo remanescente $[\mathcal{O}(h^4)]$ é a mesma que para a regra de Simpson estendida (Eq. 8.16). Na verdade, pode-se mostrar que Eq. (8.19) é *exatamente* a regra de Simpson estendida. Usar a Eq. (8.19) é a forma mais conveniente de se avaliar a regra de Simpson, usando-se, para isso, uma subrotina capaz de calcular, de forma geral, as sucessivas aproximações de uma integral pela regra do trapézio (ver exercício de programação 2).

Claramente, o processo empregado para obter a Eq. (8.19) pode ser repetido. Partimos de duas estimativas sequenciais para a integral

$$I - S(h) = -4Bh^4 + Ch^6 \tag{8.20}$$

e

$$I - S(2h) = -4B(2h)^4 + C(2h)^6 \,. \tag{8.21}$$

Fazendo 16 vezes a Eq. (8.20) menos a Eq. (8.21):

$$\begin{aligned} 16I - 16S(h) &= -4B16h^4 + 16Ch^6 + \ldots \\ -I - S(2h) &= 4B(2h)^4 - C(2h)^6 + \ldots \end{aligned}$$

$$\begin{aligned} 15I - [16S(h) - S(2h)] &= -48Ch^6 + \ldots \\ I - \underbrace{[16S(h) - S(2h)]/15}_{\equiv R_2(h)} &= -(48/15)Ch^6 + \ldots \,. \end{aligned}$$

Obtemos, assim, uma estimativa da função que difere do valor exato por um termo $\mathcal{O}(h^6)$

$$R_2(h) = \frac{16S(h) - S(2h)}{15} = S(h) + \frac{S(h) - S(2h)}{15} \,. \tag{8.22}$$

Ou seja, este procedimento remove os erros de ordem até $\mathcal{O}(h^4)$.

É possível estender-se o método de Romberg para ordens superiores. De forma geral R_n é dado pela relação de recorrência

$$R_k(h) = R_{k-1}(h) + \frac{R_{k-1}(h) - R_{k-1}(2h)}{2^{2k} - 1} \,, \tag{8.23}$$

que é a fórmula de Romberg com erro proporcional a $h^{2(k+1)+1}$ ou $1/n^{2(k+1)}$. O caso anterior [Eq. (8.19)] é justamente a fórmula de Romberg para $k = 1$.

Exemplo: calcular

$$\int_0^1 e^{-x^2} dx$$

usando o método dos trapézios e o método de Romberg.

h	$T(h)$	$\lvert\epsilon\rvert$	$R_1(h)$	$\lvert\epsilon\rvert$	$R_2(h)$	$\lvert\epsilon\rvert$
1	0.68393972	0.06				
0,5	0.73137025	0.01	0.74718043	0.0004		
0,25	0.74298410	0.004	0.74685538	3×10^{-5}	0.74683371	1×10^{-5}
0,125	0.74586561	0.001	0.74682612	2×10^{-6}	0.74682417	4×10^{-8}

Tabela 8.1: Estimativa da integral $\int_0^1 e^{-x^2} dx$ usando o método do trapézio, T, e as relações de recorrência Eq. (8.19), R_1, e Eq. (8.22), R_2. Note que o $|\epsilon|$ mostrado corresponde ao erro real cometido, ou seja, à diferença entre a estimativa e o valor exato da integral.

Vemos que o segundo valor de R_2, no qual foram utilizados 9 pontos do integrando, é preciso já para a 8ª casa decimal! Compare esta precisão com o valor obtido pelo método

do trapézio, que é preciso apenas até a 3ª casa decimal, apesar de usar as mesmas 9 avaliações da função.

Esse método de eliminar o termo dominante do erro é devido à Richardson (1927) e é frequentemente usado para melhorar a precisão ou taxa que o método converge. Depende de se conhecer a forma do erro, mas, mais importante, de que exista uma expansão do erro em termos de h. A expressão $I - T(h) = Ah^2 + Bh^4 + \ldots$ nem sempre é válida. Por exemplo na integral

$$\int_0^1 \sqrt{x} \log x dx \,,$$

o integrando não tem uma derivada finita em $x = 0$ e é possível mostrar que

$$I - T(h) = Ah^{3/2} \log h + Bh^{3/2} + Ch^2 + Dh^4 + \ldots \,.$$

Exercício: usando valores de $h = 1, 1/2, 1/4, 1/8$, calcule um valor acurado para I eliminando os termos principais na expressão acima.

8.4 Quadratura de Gauss

As fórmulas de integração desenvolvidas anteriormente (e.g. Newton-Cotes) são do tipo

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} w_i f(x_i),$$

com $n + 1$ valores de w_i (peso ou "*weight*") e $n + 1$ valores $f(x_i)$.

Assim, se x_i forem igualmente espaçados, temos $n+1$ parâmetros a serem determinados (w_i). Se x_i não são determinados a priori, temos $2n + 2$ parâmetros indeterminados. Isso quer dizer que temos duas vezes o número de graus de liberdade à nossa disposição para encontrar métodos de ordem mais elevada.

A Quadratura de Gauss também tem a forma descrita acima (ou seja, somatória de um conjunto de valores da função no intervalo, multiplicados por um peso) mas os pontos não são equidistantes, mas escolhidos *de forma que a soma ponderada forneça exatamente a integral se $f(x)$ é um polinômio de grau $2n + 1$ ou menos.* Antes de iniciar o desenvolvimento de Gauss, vamos estudar os polinômios ortogonais de Legendre.

$$\begin{aligned}
P_0(x) &= 1 \\
P_1(x) &= x \\
P_2(x) &= \frac{1}{2}(3x^2 - 1) \\
P_3(x) &= \frac{1}{2}(5x^3 - 3x) \\
P_4(x) &= \frac{1}{8}(35x^4 - 30x^2 + 3)
\end{aligned}$$

Tal que

$$P_n(x) = \frac{2n-1}{n} x P_{n-1}(x) - \frac{n-1}{n} P_{n-2}(x) \,,$$

que tem propriedades de ortogonalidade

$$\int_{-1}^{+1} P_n(x)P_m(x)dx = 0,\ n \neq m\,,$$
$$\int_{-1}^{+1} [P_n(x)]^2 dx = C(n) \neq 0\,,$$

onde C é uma constante que depende de n.

Um polinômio de ordem arbitrária

$$p_n(x) = \sum_{i=0}^{n} a_i x^i\,,$$

pode ser representado como combinação linear dos polinômios ortogonais.

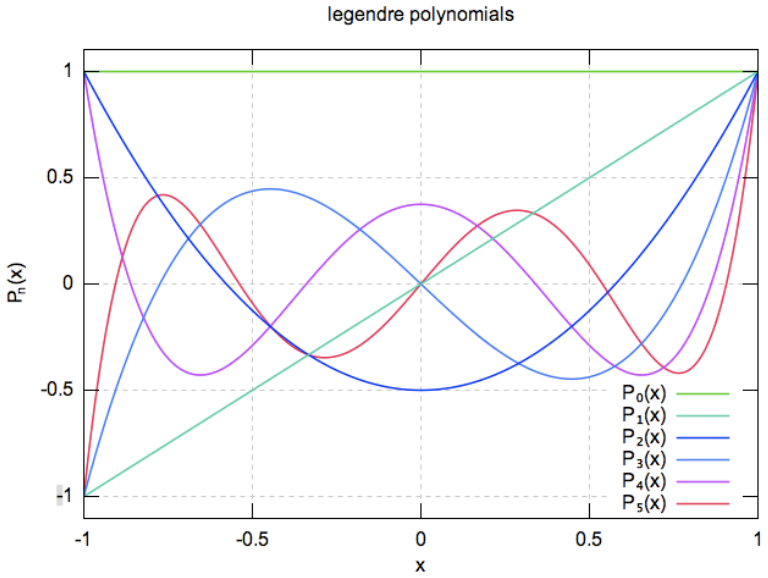


Figura 8.3: Polinômios de Legendre (fonte [a]).

Exemplo: vamos expandir um polinômio de quarta ordem $p_4(x)$ em termos de um polinômio de Legendre.

$$\begin{aligned} p_4(x) =& a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 \\ =& b_0 P_0(x) + b_1 P_1(x) + b_2 P_2(x) + b_3 P_3(x) + b_4 P_4(x) \\ =& b_0 1 + b_1 x + b_2 \left(\frac{3}{2}x^2 - \frac{1}{2}\right) + b_3 \left(\frac{5}{2}x^3 - \frac{3}{2}x\right) + b_4 \left(\frac{35}{8}x^4 - \frac{15}{4}x^2 + \frac{3}{8}\right) \end{aligned}$$

Igualando os termos de mesma potência

$$\begin{aligned}
b_4 =& \frac{8}{35}a_4 \\
b_3 =& \frac{2}{5}a_3 \\
b_2 =& \frac{2}{3}\left(a_2 + \frac{15}{4}b_4\right) = \frac{2}{3}\left(a_2 + \frac{6}{7}a_4\right) \\
b_1 =& a_1 + \frac{3}{2}b_3 = a_1 + \frac{3}{5}a_3 \\
b_0 =& a_0 + \frac{1}{2}b_2 - \frac{3}{8}b_4 = a_0 + \frac{1}{3}a_3 - \frac{1}{5}a_4\,.
\end{aligned}$$

Exercício: mostre que $p_4(x) = x^4 + 3x^3 - 2x^2 + 2x - 1$ é equivalente a

$$p4(x) = -\frac{22}{15}P_0(x) + \frac{19}{5}P_1(x) - \frac{16}{21}P_2(x) + \frac{6}{5}P_3(x) + \frac{8}{35}P_4(x),\,.$$

8.4.1 Quadratura de Gauss-Legendre

A integral

$$\int_a^b f(x)dx\,,$$

pode ser colocada na forma

$$\int_{-1}^1 f(z)dz\,,$$

através de uma mudança de variável

$$z = \frac{2x - (a+b)}{b-a},\ -1 \le z \le 1\,.$$

Sem perda de generalidade, vamos estudar as integrais do tipo

$$\int_{-1}^1 f(x)dx\,.$$

Como anteriormente (Capítulo 6), vamos aproximar $f(x)$ por um polinômio interpolante e integrar como se segue

$$\int_{-1}^1 f(x)dx = \int_{-1}^1 P_n(x)dx + \int_{-1}^1 R_n(x)dx$$

onde $R_n(x)$ é o erro do polinômio interpolante.

Como os pontos x_i ainda não foram determinados podemos escrever o polinômio interpolante na forma de Lagrange

$$\begin{aligned}
f(x) =& P_n(x) + R_n(x) \\
=& \sum_{i=0}^{n} L_i(x)f(x_i) + \underbrace{\left[\prod_{i=0}^{n}(x - x_i)\right]\frac{f^{(n+1)}(x)}{(n+1)!}}_{\text{erro do polinômio interpolante}}
\end{aligned}$$

com $x \neq x_i$, onde

$$L_i(x) = \prod_{i=0}^{n} \left(\frac{x - x_j}{x_i - x_j} \right) .$$

Se $f(x)$ for assumido um polinômio de grau $2n+1$ então o termo $\frac{f^{(n+1)}(x)}{(n+1)!}$ dever ser um polinômio de grau n. Definimos

$$q_n(x) = \frac{f^{(n+1}(x)}{(n+1)!}, x \leq x_i \, ,$$

de forma que

$$f(x) = \sum_{i=0}^{n} L_i(x) f(x_i) + \left[\prod_{i=0}^{n} (x - x_i) \right] q_n(x) \, .$$

Integrando temos

$$\begin{aligned}
\int_{-1}^{1} f(x)dx &= \int_{-1}^{1} \sum_{i=0}^{n} L_i(x) f(x_i) dx + \int_{-1}^{1} \left[\prod_{i=0}^{n} (x - x_i) \right] q_n(x) dx \\
&= \sum_{i=0}^{n} \underbrace{\left[\int_{-1}^{1} L_i(x) dx \right]}_{\equiv w_i} f(x_i) + \int_{-1}^{1} \left[\prod_{i=0}^{n} (x - x_i) \right] q_n(x) dx \\
&= \sum_{i=0}^{n} w_i f(x_i) + \underbrace{\int_{-1}^{1} \left[\prod_{i=0}^{n} (x - x_i) \right] q_n(x) dx}_{\text{erro}} \, ,
\end{aligned}$$

onde

$$w_i = \int_{-1}^{1} L_i(x) dx = \int_{-1}^{1} \prod_{j=0}^{n} \frac{x - x_j}{x_i - x_j} dx, i \neq j \, . \tag{8.24}$$

O objetivo agora é selecionar x_i de forma que o termo do erro desapareça!

Vamos expandir $q_n(x)$ e $\prod_{i=0}^{n}(x - x_i)$ em termos dos polinômios de Legendre.

$$\begin{aligned}
\prod_{i=0}^{n} (x - x_i) &= b_0 P_0(x) + b_1 P_1(x) + \ldots + b_{n+1} P_{n+1}(x) = \sum_{i=0}^{n+1} b_i P_i(x) \\
q_n(x) &= c_0 P_0(x) + c_1 P_1(x) + \ldots + c_n P_n(x) = \sum_{i=0}^{n} c_i P_i(x) \, .
\end{aligned}$$

Efetuando o produto e integrando

$$\int_{-1}^{1} q_n(x) \prod_{i=0}^{n} (x - x_i) dx = \int_{-1}^{1} \left[\sum_{i=0}^{n} \sum_{j=0}^{n} b_i c_j P_i(x) P_j(x) + b_{n+1} \sum_{i=0}^{n} c_i P_i(x) P_{n+1}(x) \right] dx \, .$$

No primeiro termo do integrando, todos os termos $i \neq j$ deverão desaparecer devido à ortogonalidade. Já a integral do segundo termo é zero pois $i \neq n+1$. Dessa forma

$$\int_{-1}^{1} q_n(x) \prod_{i=0}^{n} (x - x_i) dx = \int_{-1}^{1} \sum_{i=0}^{n} b_i c_i [P_i(x)]^2 dx$$

Uma maneira de fazer esta expressão ser zero é que todos os b_i, $i = 0, 1, \ldots, n$ sejam zero. Mas para tanto, olhando para a expressão $\prod_{i=0}^{n}(x - x_i)$ em polinômios de Legendre devermos ter

$$\prod_{i=0}^{n}(x - x_i) = b_{n+1}P_{n+1}(x)\,. \tag{8.25}$$

Mas, para que isto ocorra, *x_i devem ser as raízes do polinômio $P_{n+1}(x)$*.

Chegamos, assim ao que buscávamos. Para uma dada ordem n da quadratura de Gauss-Legendre, temos que

1. calcular a função em $n + 1$ pontos dados pela raiz de $P_{n+1}(x)$ (Eq. 8.25);
2. calcular os pesos de cada n + 1 termo usando Eq. (8.24).

Raízes de $P_{n+1}(x)$ e pesos w_i

ordem	Raízes	Pesos
$n = 0$ (1 ponto)	0	2
$n = 1$ (2 pontos)	$\pm\sqrt{\frac{1}{3}}$	1
$n = 2$ (3 pontos)	0 $\pm\sqrt{\frac{3}{5}}$	$8/9$ $5/9$
$n = 3$ (4 pontos)	$\pm\sqrt{(3 - 2\sqrt{6/5})/7}$ $\pm\sqrt{(3 + 2\sqrt{6/5})/7}$	$\frac{18+\sqrt{30}}{36}$ $\frac{18-\sqrt{30}}{36}$
$\vdots$	$\vdots$	$\vdots$

Existe uma rotina no *Numerical Recipes* (`gauleg.f`) que calcula as raízes x_i e os pesos w_i para um dado n.

Exemplo: calcule

$$\int_0^1 \frac{dx}{1 + x^2}$$

usando quadratura de Gauss-Legendre com três pontos. Inicialmente devemos fazer uma mudança de variáveis, de forma que a integral esteja entre -1 e 1.

$$\int_0^1 \frac{dx}{1 + x^2} = \int_0^2 \frac{\frac{1}{2}dx}{1 + \left(\frac{x}{2}\right)^2} = \int_{-1}^1 \frac{\frac{1}{2}dx}{1 + \left(\frac{x+1}{2}\right)^2} = \int_{-1}^1 \frac{2dx}{4 + (x+1)^2}\,.$$

Agora aplicamos a quadratura de Gauss-Legendre usando os pesos e raízes para $n = 2$ na tabela acima

$$\int_0^1 \frac{dx}{1 + x^2} \approx \sum_{i=0}^{n} w_i f(x_i)\,,$$

onde $f(x) = 2/[4 + (x+1)^2]$. Temos

$$\begin{aligned}\int_0^1 \frac{dx}{1+x^2} \approx & \frac{5}{9}\frac{2}{4+\left(-\sqrt{\frac{3}{5}}+1\right)^2} + \frac{8}{9}\frac{2}{4+(0+1)^2} + \frac{5}{9}\frac{2}{4+\left(\sqrt{\frac{3}{5}}+1\right)^2} \\ \approx & 0.274293789 + 0.355555555 + 0.155417689 \\ \approx & 0.785267033\end{aligned}$$

O valor correto da integral é $\pi/4 = 0.785398163$. Vemos que com apenas 3 avaliações da integral chegamos a um erro de 0.00013. Para efeitos de comparação, se usarmos o método de Simpson para os mesmos do intervalo obteremos o resultado 0.78333336, cujo erro é de 0.002.

Exercício: calcule novamente o valor da integral para $n = 4$ (5 pontos) e compare com o resultado para o método de Simpson estendido para 5 pontos.

8.4.2 Generalização para outros tipos de Integral

De forma geral os polinômios ortogonais satisfazem as relações

$$\int_a^b W(x) g_n(x) g_m(x) dx = 0, n \neq m$$

$$\int_a^b W(x) [g_n(x)]^2 dx = C(n) \neq 0$$

onde $W(x)$ é uma função peso. Se

$W(x) = 1$	temos polinômios de **Legendre**	$P_n(x), a = -1, b = 1$
$W(x) = \frac{1}{\sqrt{1-x^2}}$	temos polinômios de **Chebishev**	$T_n(x), a = -1, b = 1$
$W(x) = e^{-x}$	temos polinômios de **Laguerre**	$\mathcal{L}_n(x), a = 0, b = \infty$
$W(x) = e^{-x^2}$	temos polinômios de **Hermite**	$H_n(x), a = -\infty, b = \infty$

Integral tipo	x_i (raízes do polinômio)	w_i
$\int_{-1}^{1} f(x)dx$	Legendre, $P_n(x)$	$\int_{-1}^{1} L_i(x)dx$
$\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} f(x)dx$	Chebishev, $T_n(x)$	$\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} L_i(x)dx$
$\int_0^{\infty} e^{-x} f(x)dx$	Laguerre, $\mathcal{L}_n(x)$	$\int_0^{\infty} e^{-x} L_i(x)dx$
$\int_{-\infty}^{\infty} e^{-x^2} f(x)dx$	Hermite, $H_n(x)$	$\int_{-\infty}^{\infty} e^{-x^2} L_i(x)dx$

Exercício: Calcule $\int_0^{\infty} e^{-x} \sin x dx$ usando 3 pontos com Quadratura de Gauss-Legendre (valor exato da integral $= \frac{1}{2}$).

$n = 2$	raízes (w_i)	w_i
3 pontos	0.4157745567	0.7110930099
Polinômio	2.2942803602	0.2785177335
de Laguerre	6.2899450829	0.0103892565

Solução: $I = \sum_{i=0}^{n} w_i f(x_i)$, onde $f(x) = \sin x$.

$$\begin{aligned} I \approx \quad & 0.7110930099.\sin(0.4157745567) && = 0.29565438 \\ & +0.2785177335.\sin(2.2942803602) && = 0.208750114 \\ & +0.0103892565.\sin(6.2899450829) && = \underline{0.000070228} \\ & && = 0.504474722 \end{aligned}$$

que está em bom acordo com o valor exato $\frac{1}{2}$ usando apenas 3 pontos.

8.4.3 Erros na Quadratura de Gauss

Legendre	$E_n = \frac{2^{2n+3}[(n+1)!]^4}{(2n+3)[(2n+2)!]^3} f^{(2n+2)}(\xi)$	ξ em $(-1,1)$
Chebishev	$E_n = \frac{2\pi}{2^{2n+2}(2n+2)!} f^{(2n+2)}(\xi)$	ξ em $(-1,1)$
Laguerre	$E_n = \frac{[(n+1)!]^2}{(2n+2)!} f^{(2n+2)}(\xi)$	ξ em $(0,\infty)$
Hermite	$E_n = \frac{(n+1)!\sqrt{\pi}}{2^{2n+1}(2n+2)!} f^{(2n+2)}(\xi)$	ξ em $(-\infty,\infty)$

8.5 Integração pelo Método de Monte Carlo

REVISAR

Métodos de Monte Carlo (MC) são aqueles que se utilizam de números aleatórios (ou pseudo-aleatórios) na sua composição. Eles tem larga aplicação em modelos físicos, como por exemplo na mecânica estatística e simulações de sistemas de muitos corpos (*"many body"*).

Uma lista de aplicações:

- Modelo de Ising- algoritmo de Metrópolis
- *"Salesman problem"*
- *"Random walk"*
- *DMC - "Diffusion M.C. Method"*, cálculo do estado fundamental de sistemas de muitos corpos
- *QDMC - "Quantum Determinant M.C."*, modelo de Hubbard
- *PIMC - "Path Integral M.C."*
- Física de reações (colisões)
- Integrais Multidimensionais, $\int dxdydz\ dp_x dp_y dp_z\ f(x,y,z,p_x,p_y,p_z)$

Nesse curso introdutório vamos apenas considerar a aplicação de números aleatórios para o cálculo da integral de uma função.

8.5.1 Geração de números aleatórios

A geração de números aleatórios (*"random number generators"* - RNG) é uma arte por si só. Atualmente é possível adquirir geradores de números realmente aleatórios (*"true random number generators"*) que digitalizam o ruído gerado em uma junção semi-condutora ao ser atravessado por uma corrente. Entretanto, a taxa de geração é limitada. Em geral, usam-se operações aritméticas para geração de números aleatórios (ou pseudo-aleatórios).

Um dos algoritmos aritméticos mais conhecidos é o devido à Lehmer (1951) e chamado de (*"Linear Congruential Generator "* - LCG). Ele é expresso na forma

$$Z_{i+1} = (aZ_i + b)\text{mod } m = \text{resto de } \frac{aZ_i + b}{m}$$

$$\begin{aligned} m &\rightarrow \text{ módulo} \\ 0 \leq a < m &\rightarrow \text{ multiplicador} \\ 0 < b < m &\rightarrow \text{ incremento} \\ 0 \leq Z_0 < m &\rightarrow \text{ semente } \textit{"seed"} \text{ ou valor inicial } \textit{"start value"} \end{aligned}$$

Se $a,\ b,\ m$ são propriamente escolhidos, então o período será máximo. Nesse caso, todos os inteiros entre 0 e $m-1$ ocorrerão em algum ponto de forma que a semente inicial Z_0 é tão boa como qualquer outra.

Exemplo: Seja $a = 4,\ b = 1,\ m = 9,\ Z_0 = 3$.

$$Z_{i+1} = (4Z_i + 1)\text{mod } 9$$

$$\begin{aligned} Z_0 &= 3 \\ Z_1 &= (4 \times 3 + 1)\text{mod } 9 = 13\text{mod } 9 = \text{ resto } \frac{13}{9} = 4 \\ Z_2 &= (4 \times 4 + 1)\text{mod } 9 = 17\text{mod } 9 = \text{ resto } \frac{17}{9} = 8 \\ Z_3 &= (4 \times 8 + 1)\text{mod } 9 = 33\text{mod } 9 = \text{ resto } \frac{33}{9} = 6 \\ &\vdots \\ &7,\ 2,\ 0,\ 1,\ 5,\ 3 \ \leftarrow \text{ ciclo de tamanho } m \end{aligned}$$

Para termos uma longa sequência, m deve ser um número grande, mas não tão grande que $a \times m$ cause *overflow*.

Se obtivermos m números aleatórios entre 0 e $m-1$ (Z_{i+1}), podemos obter uma distribuição uniforme entre [0,1) (U_{i+1}) pela divisão:

$$U_{i+1} = \frac{Z_{i+1}}{m}$$

8.5.2 Park & Miller (LCG b=0)

$$\begin{aligned} a &= 7^5 = 16807 \\ b &= 0 \\ m &= 2^{31} - 1 = 2147483647 \end{aligned}$$

Ciclo de $2^{31} - 2$ números aleatórios, com $Z_0 \neq 0$.

8.5.3 Integração por Monte-Carlo simples

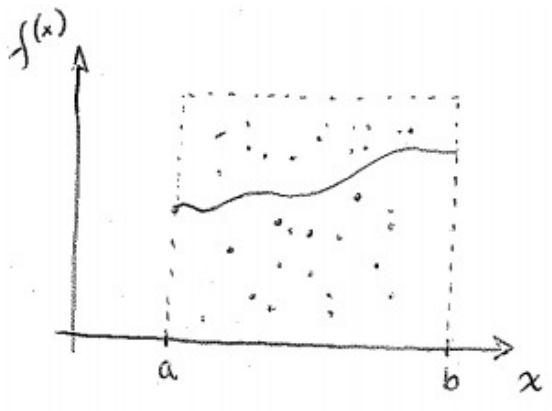


Figura 8.4: Distribuição de pontos sob a área avaliada, e a curva da função correspondente.

Suponha que desejemos saber a área sob a curva que liga a e b. Jogamos então aleatóriamente pontos no retângulo que contém a área de integração. Dividindo o número de pontos sob a curva pelo total de pontos jogados, e multiplicando pela área do retângulo, teremos uma aproximação da área (ou $\int_a^b f(x)dx$) sob a curva.

$$\int_a^b f(x)dx = \text{ Área sob a curva } \approx A\frac{\text{no. de pontos sob a curva}}{\text{no. total de pontos}}$$

onde A é a área do retângulo.

Exemplo: Cálculo da área de um círculo.

$$\text{Área do círculo } \approx A\frac{\text{no. de pontos dentro do círculo}}{\text{no. total de pontos jogados}}$$

onde A é a área do quadrado circunscrito.

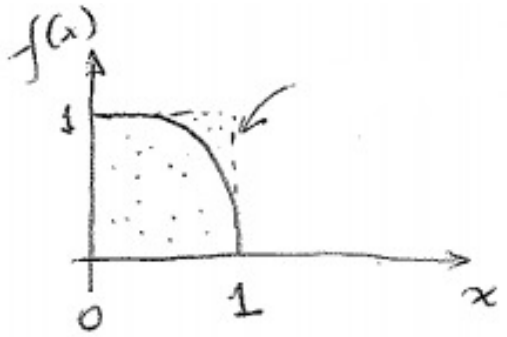


Figura 8.5: Curva $x^2 + y^2 = 1$, ou $y = \sqrt{1 - x^2}$ considerada para o cálculo da área do círculo.

Considerando apenas o 1° quadrante:

$$\int_0^1 f(x)dx = \int_0^1 \sqrt{1-x^2}dx = \frac{1}{4}\text{área do círculo} = \frac{\pi}{4}$$
$$\approx \frac{\text{no. de pontos sob a curva (dentro)}}{\text{no. de pontos total}}$$

Usando um *linear congruential generator* na forma $Z_{i+1} = aZ_i \text{mod } m$, $a = 16807$, $m = 2^{31} - 1$ obtivemos a seguinte tabela para 10 pontos ($Z_0 = 1$):

x	y	$x^2 + y^2$
7.8263693×10^{-6}	0.1315378	1.7302191×10^{-2}
0.75560533	0.4586501	0.7812994
0.5327672	0.2189592	0.3317840
4.7044616×10^{-2}	0.6788647	0.4630705
0.6792964	0.9346929	1.335094(fora)
0.3835021	0.5194164	0.4168672
0.8309653	3.457211×10^{-2}	0.6916987
5.3461634×10^{-2}	0.5297002	0.2834404
0.6711494	7.6981862×10^{-3}	0.4505008
0.3834156	6.6842236×10^{-3}	0.1514754

$$\pi = 4\frac{\text{no. pontos dentro}}{\text{no. total de pontos}} = 4\frac{9}{10} \simeq 3.6$$

Este é um resultado grosseiro, pois poucos pontos foram utilizados.

Pontos	π
10	$\frac{4\times9}{10} \approx 3.6$
20	$\frac{4\times17}{20} \approx 3.2$
$\vdots$	$\vdots$
100	$\frac{4\times71}{010} \approx 2.8$
$\vdots$	$\vdots$
1000	$\frac{4\times792}{1000} \approx 3.16$