7

CIÈNCIES
APLICADES

UPCGRAU

Taller de Matemàtiques

Pràctiques en Matlab/Octave, amb un apèndix en Python

Joaquim Puig

7

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

iniciativa digital politècnica
Publicacions Acadèmiques UPC

UPCGRAU

Taller de Matemàtiques

Pràctiques en Matlab/Octave, amb un apèndix en Python

Joaquim Puig

En col·laboració amb el Servei de Llengües i Terminologia de la UPC

Primera edició: setembre de 2011

Disseny i dibuix de la coberta: Jordi Soldevila
Disseny maqueta interior: Jordi Soldevila
Maquetació: Mercè Aicart

Oficina de Publicacions Acadèmiques Digitals de la UPC
Jordi Girona Salgado 31,
Edifici Torre Girona, D-203, 08034 Barcelona
Tel.: 934 015 885 Fax: 934 054 101
www.upc.edu/idp
E-mail: info.idp@upc.edu

Dipòsit legal: B-29936-2011
ISBN: 978-84-7653-913-2

“Desitjo que, com a bon informàtic, no resisteixis la temptació de manipular el programa (...) La varietat de resultats, totalment imprevisibles, fa molt estimulant aquesta tasca.” Joan Puig i Reixach [15]

Dedicat al meu pare

Índex

Presentació

Aquests apunts que presentem s'han elaborat per a unes pràctiques anomenades "Taller de Matemàtiques", que són comunes a les assignatures *Àlgebra Lineal* i *Càlcul I* de primer curs dels graus de l'Escola Tècnica Superior d'Enginyeria Industrial de Barcelona (ETSEIB). L'objectiu és introduir un conjunt d'eines informàtiques que permetin donar solucions numèriques a problemes que sorgeixen en aquestes assignatures. Entre altres coses, es vol que els alumnes siguin capaços de fer càlculs elementals amb matrius i vectors, representar i definir funcions, estudiar-ne els zeros, calcular integrals...

És evident que fer unes pràctiques en abstracte no tindria cap sentit i, per tant, cal triar un programari que ens permeti dur-les a terme de forma eficient. Hi ha força paquets que reuneixen unes característiques adequades, tant lliures i gratuïts (`Python`, `Octave`, `Scilab`) com propietaris i de pagament (`Matlab`).

En aquestes pràctiques, hem decidit usar el llenguatge que comparteixen els paquets `Matlab` i `Octave`, que tenen un ús molt estès en l'enginyeria. De tota manera, hem volgut centrar-nos en els procediments que seguim per a l'estudi de problemes, més enllà de les eines que cadascuna de les plataformes ens ofereixi. Finalment, hem inclòs un petit apèndix per poder seguir en `Python` el que fem en `Matlab/Octave`.

Referències: Per a l'elaboració d'aquests apunts hem utilitzat material divers que el Departament de Matemàtica Aplicada I de la Universitat Politècnica de Catalunya ha produït al llarg dels anys, com són els apunts de Toni Susín [17, 18] i de Jaume Amorós [1]. També hem usat els apunts de Guillem Borrell [2] i els desenvolupats a la UPM [4]. Quant als llibres, tot i que queden fora de l'abast del curs, hem seguit els de Quarteroni i Saleri [16] i el de Moler [14].

Agraïments: Vull agrair a les persones següents, que han aportat millores a aquests apunts: David Alonso, Jaume Amorós, Inma Baldomá, Marc Camara, Marta Casanellas, Joan Cirera, Amadeu Delshams, Humildad Escorihuela, Josep Ferrer, Tomàs Lázaro, José Vicente Mandé, Maria Rosa Massa, Marta Peña, Rafael Ramírez Ros, Fátima Romero, Germán Sánchez, Toni Susín i Marta València. Vull fer extensiu l'agraïment a tot

el Departament de Matemàtica Aplicada I pel suport que m'han donat en l'elaboració de les pràctiques i d'aquests apunts.

Errata i material complementari: A l'adreça web `www.ma1.upc.edu/~jpuig/taller` trobareu material addicional per aquests apunts, com també els codis en `Matlab`, `Octave` i `Python` que podreu descarregar. Si hi trobeu errors o teniu suggerències de millora per aquests apunts, podeu usar també aquesta web per posar-vos en contacte amb mi.

Matlab/Octave com a calculadora científica

En aquesta primera pràctica, ens familiaritzarem amb el `Matlab` i `Octave`, i el seu ús com a substitut de la calculadora científica. En primer lloc, ens aproximarem als entorns de treball de les dues plataformes, i després en coneixerem l'ús més bàsic.

1.1. Matlab, Octave... Què és això?

En aquestes pràctiques, usem indistintament `Matlab` i `Octave`, que són dos paquets per a càlcul científic i anàlisi de resultats àmpliament usats en enginyeria. Tot i que són diferents, ambdós programes usen un conjunt d'ordres molt similar i que, en aquestes pràctiques, hem intentat que sigui màximament compatible. És per això que ens referirem a `Matlab/Octave` per parlar d'aquest llenguatge comú a `Matlab` i a `Octave` o per referir-nos indistintament als dos paquets.

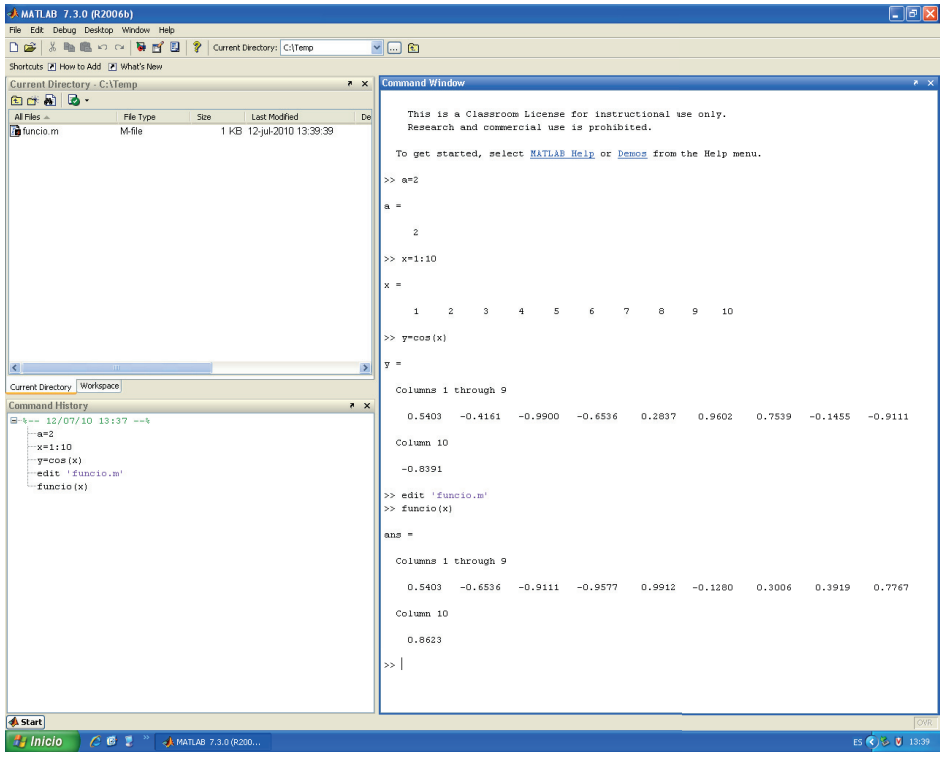
1.1.1 Què és el Matlab i com el podem usar

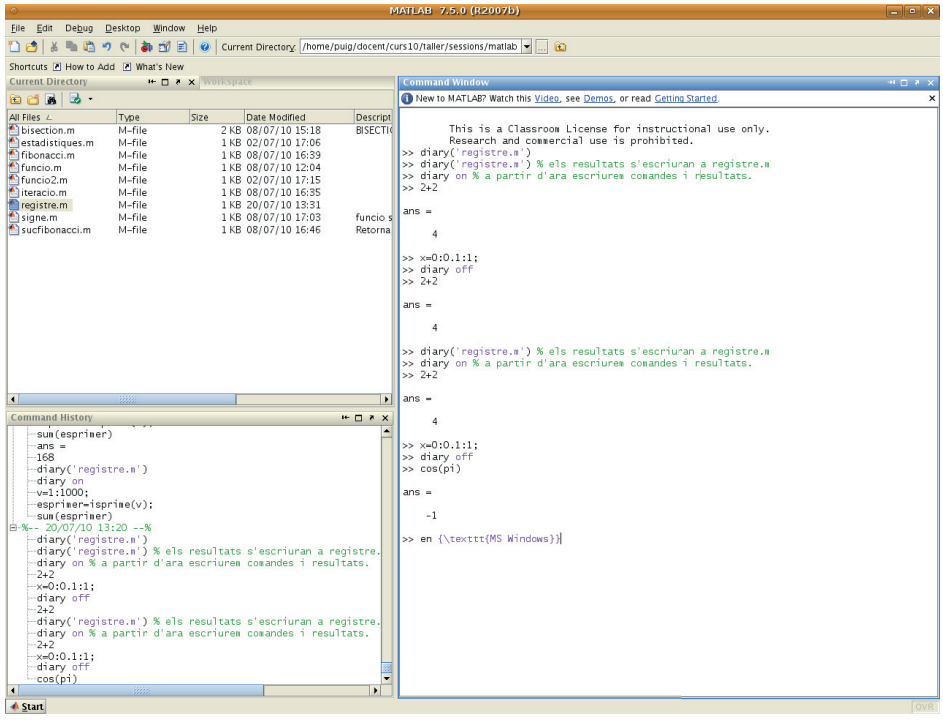
El `Matlab` [13] és un paquet comercial molt usat en enginyeria per a realitzar càlculs científics i visualització de dades que desenvolupa l'empresa MathWorks. Està especialment optimitzat per a les operacions matricials i d'aquí ve el seu nom, que és un acrònim de *Matrix Laboratory*.

Per poder usar el `Matlab`, haurem d'anar a les aules d'informàtica a on estigui instal·lat. Concretament, a les aules informàtiques de l'ETSEIB, en podem usar les versions 6.5 i 7.3 en `MS/Windows` i la 7.6 en `GNU/Linux`. Podem usar-les indistintament per al que farem (per fer aquests apunts, hem usat la versió 7.3 en `GNU/Linux`). Podem executar-los tant en `MS/Windows` com en `GNU/Linux`, a través dels menús d'inici de "Programari Docència" → "Departament de Matemàtica Aplicada I" → "Taller de Matemàtiques".

En iniciar-los, ens apareixerà una pantalla inicial com la que es mostra a la figura 1.1, que conté tres elements importants:

Fig. 1.1
Superior: Finestra d'inici del Matlab 7.3 en MS Windows, amb el directori de treball (*current directory*), el registre d'instruccions (*command history*) i la finestra d'instruccions (*command window*). Inferior: El mateix per al Matlab 7.5 en GNU/Linux.





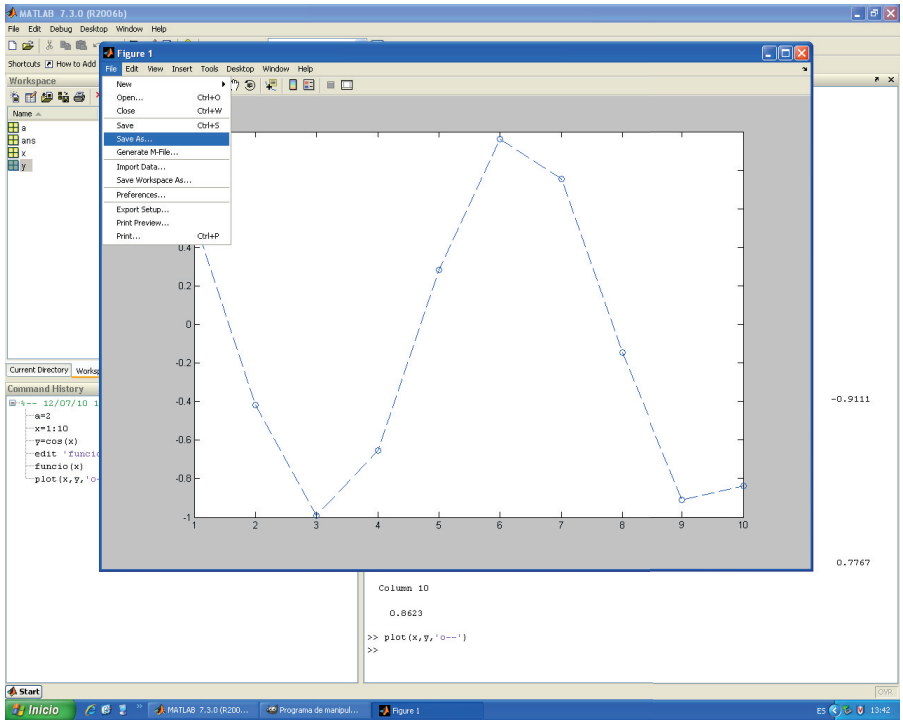
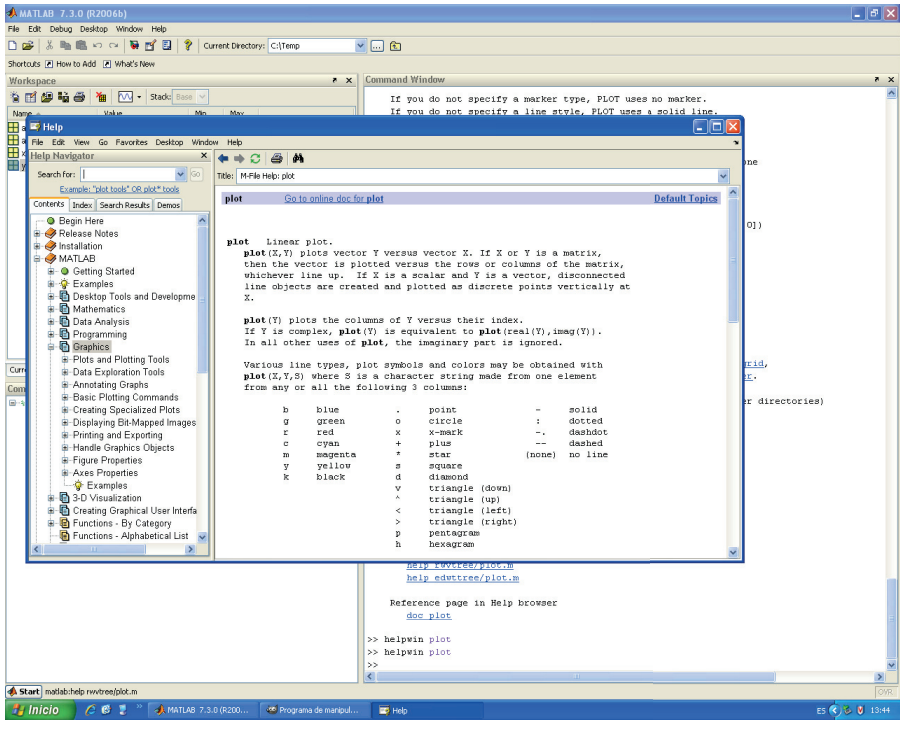


Fig. 1.2
Superior: Finestra de dibuix de Matlab 7.3 en MS Windows.
Inferior: Finestra d'ajuda de Matlab 7.3 en MS Windows. Hi podem accedir a través de l'ordre helpwin.



- La finestra d'instruccions (*command window*), també anomenada consola o línia d'ordres, que és a on anirem escrivint les instruccions.
- La finestra del directori de treball (*current directory*), que indica a quin directori estem treballant. És molt important que, per començar, ens situem en un directori a on tinguem permís d'escriptura. Aquest ha de ser sempre el primer pas que fem en engegar. Podem commutar-la amb l'espai de treball (*workspace*), que ens permetrà editar i visualitzar les variables que representem, a través de la pestanya que apareix a baix.
- El registre d'instruccions (*command history*) on ens apareixeran les ordres que hem anat executant.

En cas de fer dibuixos, com veurem a la pràctica 3, ens apareixerà una nova finestra, com la de la figura 1.2. Per executar la finestra d'ajuda, que és molt completa, podem anar a l'element `help` del menú, prémer la tecla `F1` o bé teclejar `helpwin` a la consola d'ordres.

1.1.2 Matlab i Octave

Com hem dit, `Matlab` és un programa de pagament i, per tant, no el podeu usar lliurement a casa, com tampoc no podeu modificar-lo ni adaptar-lo a les vostres necessitats. De tota manera, tot el que farem en aquestes pràctiques es pot fer igualment amb el paquet `Octave` [19], que és molt compatible amb el programa `Matlab` (en particular, ho és en el que farem). El paquet `Octave` disposa, a més, d'una gran llista de paquets opcionals per a dur a terme molts càlculs i anàlisis de dades que podeu necessitar en el futur. `Octave` és de codi lliure i mantingut per la comunitat de desenvolupadors, sota els auspicis del projecte GNU. En aquests apunts utilitzem la versió d'`Octave` 3.4 o superior, que incorpora força novetats respecte les anteriors. El paquet `Octave` es pot descarregar gratuïtament des de la seva web [19], on també trobareu manuals i més informació. Hi ha versions per a `MS/Windows`, `GNU/Linux` i `OSX` que us podeu baixar d'aquesta web.

Quan engegueu l'`Octave` per primer cop, veureu que, de totes les finestres que apareixen en `Matlab`, aquí només hi ha la línia d'ordres, a través de la qual anirem introduint les ordres (veieu la figura 1.3).

1.2. Ús com a calculadora científica

1.2.1 Consola d'ordres

A la consola d'ordres es poden efectuar tota una sèrie d'operacions. Per exemple, podem escriure la instrucció

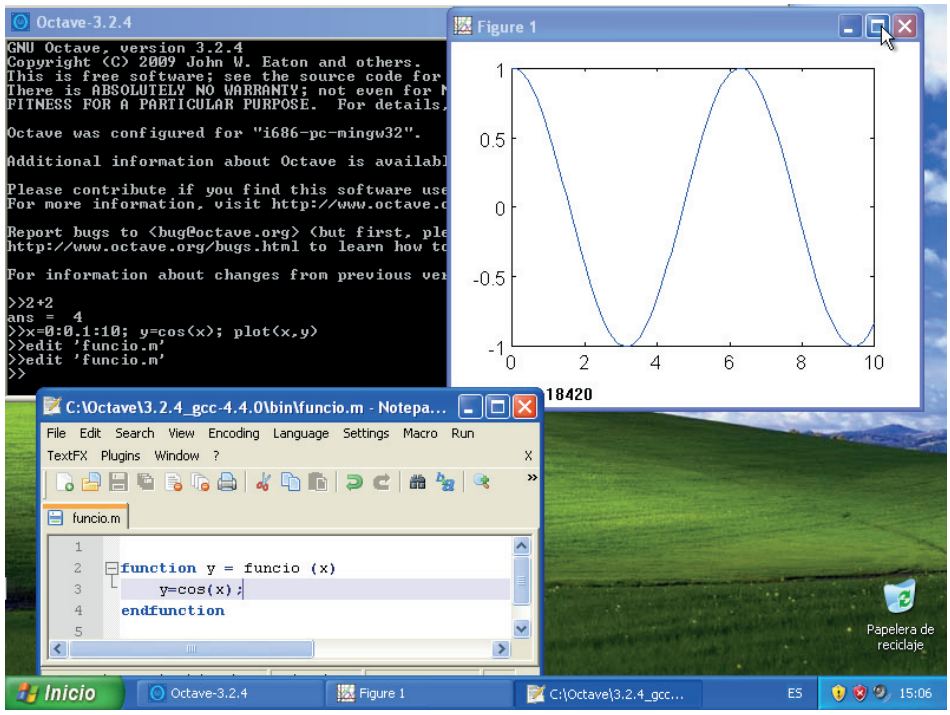
```
>> 2+2
```

i prémer la tecla Retorn perquè ens surti el resultat desitjat (4, en aquest cas)

```
    ans = 4
```

Amb vista a facilitar les operacions, la interfície de `Matlab/Octave` té les dreceres següents:

- Podeu esborrar i moure-us per la línia amb les tecles de desplaçament, retrocés i supressió.



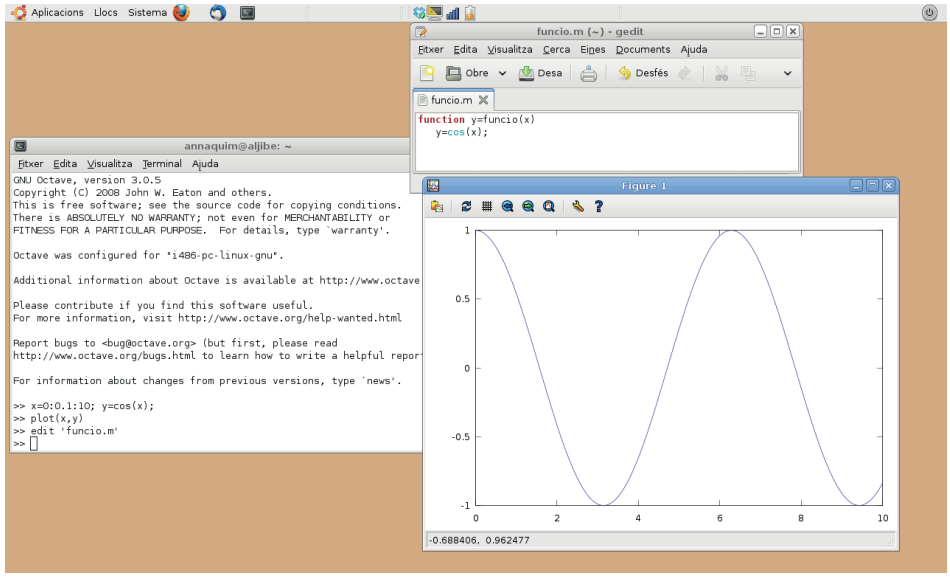


Fig. 1.3
Superior: Sessió d'Octave en MS Windows.
Inferior: Sessió d'Octave en GNU/Linux.

- La fletxa cap amunt recupera les ordres anteriors.
- El resultat s'emmagatzema en la variable `ans`, com veurem més avall.
- Per accedir a l'ajuda en línia, podem fer `help ans` per exemple, a `Octave` se'ns proporcionaria l'ajuda següent (en anglès):

```
>> help cos
-- Mapping Function:  cos (X)
    Compute the cosine of each element of X.

cos is a built-in mapper function

Additional help for built-in functions and operators is
available in the on-line version of the manual.  Use the command
`doc <topic>' to search the manual index.

Help and information about Octave is also available on the WWW
at http://www.octave.org and via the help@octave.org
mailing list.
```

mentre que a `Matlab` se'ns obrirà una finestra amb l'ajuda i exemples.

1.2.2 Expressions numèriques

Com és lògic, el `Matlab/Octave` permet efectuar les operacions que esperaríem per a una calculadora científica:

```
>> (1+2*3+4^5-6)/(7*8) %*=producte, ^=exponent
ans =  18.304
>> format long % mostra amb precisio doble
>> (1+2*3+4^5-6)/(7*8)  % parentesis importants
ans =  18.3035714285714
```

Observem el següent:

- Tot el que posem a partir del símbol `%` és un comentari i no s'executa. Per evitar problemes de compatibilitat, no escriurem accents en el codi ni en els comentaris.
- Podem copiar totes o algunes de les línies del codi i enganxar-les a la línia d'ordres. S'entendrà que cada canvi de línia és el final d'una instrucció (el mateix que fer Retorn a la línia d'ordres).
- Els decimals es mostren amb un punt a baix i no pas amb una coma.
- Quan s'inicia el `Matlab/Octave`, la precisió per defecte és `short`. Per mostrar tots els decimals que usa el sistema, podem passar a la precisió estesa, fent `format long` (si volem tornar a precisió simple, fem `format short`).
- El símbol `^` representa l'exponenciació i `*`, el producte, de manera que la primera línia és $(1+2\cdot 3+4^5-6)/(7\cdot 8)$. A `MS/Windows`, pot ser que hagueu de prémer el símbol `^` dos cops perquè aparegui.

La majoria de funcions matemàtiques bàsiques vénen expressades per les seves notacions naturals (en anglès). Les instruccions següents serviran d'exemple:

```
>> cos(pi/3) %pi es pi, cos es el cosinus
ans =
   0.500000000000000
>> exp(1) % exp() funcio exponencial, exp(1) el num. e
ans =
   2.718281828459046
>> log(exp(1)) % log() es el logaritme neperia
ans =
     1
>> sqrt(4) % sqrt() és l'arrel quadrada
ans =
     2
>> log10(10) % log10() es el logaritme en base 10
ans =
     1
>> power(10,log10(2)) % power(a,b) es a^b
ans =
   2.000000000000000
```

És important notar el següent:

- Encara que amb la notació habitual de matemàtiques no escrivim els punts del producte, en `Matlab/Octave` sí que ho hem de fer a través de l'operador `*`. Podeu veure que, si fem `2 3`, donarà un error i no pas sis.
- Les ordres `sin()`, `cos()`, `log()`, `log10()`, `sqrt()` són *funcions* del `Matlab/Octave` i per cridar-les els heu de passar l'argument entre parèntesis. Per exemple, `cos(0)` és $\cos 0$. Els parèntesis són imprescindibles. Observeu que el símbol per al logaritme natural o neperià és `log()`, per al logaritme en base 2 és `log2()` i per al logaritme en base 10 és `log10()`. El símbol `sqrt()` correspon a l'arrel quadrada (de l'anglès *square root*). Per calcular e^x, l'exponencial, usarem l'ordre `exp()`. A l'esquema del final de la pràctica hi ha una referència entre les funcions més habituals i el seu valor en `Matlab/Octave`.
- En canvi, l'ordre `pi` és una *variable interna* que conté una aproximació del valor de $\pi \approx 3.14159265358979$. El nombre e no és una variable del `Matlab` (proveu de fer `e^2`) però sí que ho és de l'`Octave` (proveu de fer el mateix).
- A `Matlab/Octave`, el resultat de la divisió entre dos nombres enters amb divisió no exacta, per exemple `2/3`, és sempre l'aproximació en tipus doble i no pas el resultat de la divisió sencera, zero en aquest cas.

Com a exercici, podeu calcular l'expressió següent, tot convertint-la a expressió de `Matlab/Octave`:

$$\left(\sin\frac{\pi}{3} + 2\cos\pi - \ln(e^2) + \sqrt{10}\right)\log_2 4.$$

1.2.3 Precisió de la màquina, avisos i errors

La instrucció següent

```
>> log2(2)+log(exp(1))+sin(pi/2)+sqrt(1)+4*atan(1)/pi+log10(10)
ans =  6
```

mostra el resultat de l'operació

$$\log_2 2 + \log e^1 + \sin\frac{\pi}{2} + \sqrt{1} + \frac{4\arctan 1}{\pi} + \log_{10} 10 = 6$$

Observem que, en aquest resultat, ens el mostra de manera exacta com a `6` (no com a `6.000000`), ja que concorda amb la precisió interna de la màquina. Arribats en aquest punt, és interessant poder representar nombres molt grans o molt petits en notació científica, mitjançant la qual un número de la forma $a.b \cdot 10^c$ s'escriu com `a.bec` Així, per expressar 52000000, escriuríem `5.2e7` o la llista següent d'exemples:

```
5.2e7 % resultat= 52000000
1e2 % el mateix que 100
1e10 % un 1 i 10 zeros= 10000000000
1e-2 % el mateix que 0.01
1e-10 % es el mateix que 0.0000000001
1.52e-2 % 0.0152
0.000152e2 % una manera complicada d'escriure el mateix
15.2e-4 % manera complicada bis
```

Internament, es treballa amb una certa precisió (anomenada èpsilon de la màquina ε), que podem conèixer a través de l'ordre `eps`

```
>> eps
ans =  2.22044604925031e-16
```

i que ens pot donar resultats lleugerament diferents, depenent de si usem `Matlab` o `Octave`. Això comporta que el resultat de l'operació següent:

```
>> 1+1e-17
ans =  1
```

no sigui $1 + 10^{-17}$ sinó 1 perquè la diferència entre els dos valors és més gran que l'èpsilon de la màquina. Tampoc no és possible representar valors superiors als que ens indiquin la constant del sistema `realmax`.

```
>> realmax
ans =  1.79769313486232e+308
>> 1e309
ans = Inf
```

Una altra constant interessant és la que ens indica `realmin`, que representa el menor nombre positiu que es pot representar amb tots els decimals amb què es treballa. Vegem-ho

```
>> realmin % menor nombre positiu representable exactament
ans =  2.22507385850720e-308
>> realmin/10 % podem dividir per 10 pero perdem decimals
ans =  2.22507385850720e-309
```

```
>> realmin/100 % idem
ans =  2.22507385850720e-310
>> realmin*1e-16 % per la maquina aixo es zero.
ans = 0
```

Abans hem vist que, en fer `1e309`, ens ha donat com a resultat `Inf` i no pas un valor numèric. Altres maneres de produir resultats no numèrics són dividint per zero, cosa que ens donarà un `warning` o avís,

```
>> 1/0
warning: division by zero
ans = Inf
```

o bé aplicant funcions en valors fora del seu domini

```
>> log(0)
ans = -Inf
```

Un altre tipus d'avís ens apareix quan fem operacions del tipus $0/0$ o similars

```
>> 0/0
warning: division by zero
ans = NaN
>> 1/0 - 1/0
ans = NaN
```

en canvi, observeu la diferència amb

```
>> 1/0 + 1/0
ans = Inf
```

1.2.4 Nombres complexos

Com sabem, hi ha funcions que, tot i no tenir valor real, tenen valor complex, com ara $\sqrt{-4} = 2i$. En aquest cas, `Matlab/Octave` ens donarà el valor complex

```
>> sqrt(-3)
ans =  0.000000000000000 + 1.732050807568877i
>> sqrt(-4)
ans =  0 + 2i
>> log(-exp(4))
ans =  4.00000000000000 + 3.14159265358979i
```

A part del símbol `i`, també és possible usar `j` com a unitat imaginària. Tornarem als nombres complexos més endavant, a la segona pràctica. Podem fer altres operacions i calcular el mòdul d'un nombre complex i el seu argument, com també parts reals, imaginàries, o conjugar-los:

```
>> j^2
ans = -1
>> 2*angle(i)/pi % angle() ens dona l'argument d'un complex
ans =  1
>> abs(sqrt(2)+sqrt(2)*i) % abs ens dona el modul d'un complex
ans =  2
>>  z=1+2*i
z =
   1.0000 + 2.0000i
>> real(z) % part real de z
ans =
     1
>> imag(z) % part imaginaria de z
ans =
     2
>> conj(z) % el conjugat de z, 1+2i
ans =
   1.0000 - 2.0000i
```

1.2.5 Variables

Hem vist ja alguns exemples de variables del sistema, com ara `pi`, `ans`, `Inf`. És molt convenient poder definir variables nosaltres mateixos, per poder emmagatzemar valors que després usarem en altres càlculs. A:

```
>> a=3.12
a =
    3.1200
>> b=4
b =
     4
>> c=b+a
c =
    7.1200
```

la variable `c` conté el valor `7.12`, que és el resultat de la suma. Si no volem que es mostrin els resultats intermedis, podem acabar cada línia amb un punt i coma (`;`):

```
>> a=3.12;
>> b=4;
>> c=b+a
```

o bé fins i tot posar-los tots en una mateixa línia separats per punt i coma:

```
>> a=3.12; b=4; c=b+a
c =
    7.1200
```

Molts cops voldrem sobreescriure el valor d'una variable per un altre valor, potser actualitzant un valor antic de la variable:

```
>> a=0
a =
     0
>> a=cos(a) % ara a val 1=cos(0)
a =
     1
```

amb la qual cosa el valor inicial de `a`, que era `0`, passa a ser `cos(0)=1` i, per tant, esborra el valor inicial. Usant això i la fletxa cap amunt podem fer petites iteracions, com per exemple:

```
>> a=0
a = 0
>> a=cos(a)
a =  1
>> a=cos(a)
a =  0.540302305868140
>> a=cos(a)
a =  0.857553215846393
>> a=cos(a)
a =  0.654289790497779
>> a=cos(a)
a =  0.793480358742566
>> a=cos(a)
a =  0.701368773622757
>> a=cos(a)
a =  0.763959682900654
>> a=cos(a)
a =  0.722102425026708
>> a=cos(a)
a =  0.750417761763761
>> a=cos(a)
a =  0.731404042422510
>> a=cos(a)
a =  0.744237354900557
>> a=cos(a)
a =  0.735604740436347
>> a=cos(a)
a =  0.741425086610109
```

A la pràctica 9, veurem maneres millors de fer aquestes iteracions.

En principi, també podem sobreescriure les variables del sistema, com per exemple fer que π sigui tres. Clarament, això no és molt bona idea, com es veu a continuació:

```
>> pi % aixo ens mostrara el valor de pi del sistema
ans =
   3.141592653589793
>> cos(pi) % el resultat es -1
ans =
    -1
>> pi=3 % redefinim pi
pi =
     3
```

```
>> cos(pi) % ja no es -1, sino -0.989992496600445
ans =
  -0.989992496600445
>> clear pi % tornem pi al seu valor original.
>> pi
ans =
   3.141592653589793
```

La darrera instrucció, `clear`, és molt important ja que *neteja* el valor que li haguem assignat a una variable i la restaura al valor original. En un moment de pànic, sempre podem fer `clear` i Retorn i ens restaurarà el valor inicial de la variable. Per exemple, en `Matlab`,

```
>> a=0
a = 0
>> clear % esborrem totes les variables
>> a
??? Undefined function or variable 'a'.
```

ens retorna un error perquè `a` no correspon a cap valor emmagatzemat a la nostra sessió. Observem que els textos dels avisos d'error poden diferir en `Matlab` o en `Octave`, però que essencialment vénen a dir el mateix. Per exemple, el text d'aquest avís en `Octave` seria

```
>> a=0
a = 0
>> clear
>> a
error: `a' undefined near line 182 column 1
```

ens

1.3. Complements opcionals

1.3.1 Registre de la sessió amb `diary()`

L'ordre `diary` ens permet mantenir un registre de les ordres i els resultats que anem obtenint, tant en `Matlab` com en `Octave`. Per exemple, si volem que ens guardi un conjunt d'instruccions en un fitxer anomenat `registre.txt`, farem el següent:

```
>> diary('registre.txt') % els resultats s'escriuran a registre.txt
>> diary on % a partir d'ara escriurem ordres i resultats.
>> 2+2
ans =
     4
>> log(1);
>> diary off
>> cos(pi)
ans =
    -1
```

i al fitxer `registre.txt` (que podem veure amb qualsevol editor de text) s'hi haurà escrit el següent:

```
diary on % a partir d'ara escriurem ordres i resultats.
2+2
ans =
     4
log(1);
diary off
```

Observem que el fitxer conté tant les instruccions com els resultats (i, per tant, no és només un conjunt d'ordres en `Matlab/Octave`). A més, cal anar amb compte de no usar-la indiscriminadament perquè ens podríem quedar sense espai al disc.

1.4. Exercicis

1.1. Calculeu l'expressió numèrica $\cos\left((\pi-1)^{\frac{5}{2}}\right)$.

Resposta: -0.957659480323385

1.2. La fórmula per calcular les quotes d'un préstec amb amortització simple és la següent

$$A = P\frac{i(1+i)^n}{(1+i)^n-1}$$

on

- A és la quota de cada periode (mensual, anual...)
- P és el capital inicial del préstec.
- i és el tipus d'interès (en tant per u) en cada període (per exemple, si és d'un 2% anual, això significa que mensualment i val $\frac{2}{12\cdot 100}$. Se suposa el tipus d'interès constant.
- n és el nombre total de períodes de cada pagament (per exemple, amb una hipoteca mensual de 30 anys, això significa que hi ha $30\cdot 12 = 360$ mensualitats).

El preu mitjà d'un habitatge a Barcelona és d'uns 4000 euros el metre quadrat. Suposem que volem comprar un pis de 70 metres quadrats i que disposem d'uns estalvis de 50000 euros. Calculeu la quota mensual per finançar la resta de l'import d'un habitatge amb una hipoteca al tipus d'interès constant del 2.5% anual durant 30 anys.

Resposta: 908.778 euros al mes.

1.5. Esquema de la pràctica

- En iniciar `Matlab/Octave` hem de canviar a un directori amb escriptura.
- Canvi de directori: `chdir('cami_al_directori')`.
- Ajuda en línia d'ordres `help tema_ajuda`.
- Símbol de comentari: `%`
- Operacions bàsiques `a+b, a-b, a*b, a/b`.
- Les funcions s'indiquen entre parèntesis:
  - Funcions exponencials i potencials e^x: `exp(x)`, $\ln x$: `log(x)` (logaritme neperià), $\log_{10} x$: `log10(x)` (logaritme decimal),a^b: `a^b`, $\sqrt{x}$: `sqrt(x)`...
  - Funcions trigonomètriques: `cos(x), sin(x), tan(x)` i les inverses `asin(x), acos(x), atan(x)`.
  - Funcions trigonomètriques hiperbòliques: `cosh(x), sinh(x), tanh(x)`.
- Nombres complexos: nombre imaginari `i` o `j` (`z=a+b*i`): `angle(z)` (argument complex), `real(z)` (part real), `imag(z)` (part imaginària), `conj(z)` (conjugat).
- Format dels resultats: `format short` (format curt) i `format long` (format llarg).
- Constants del sistema: `pi`...
- Assignació de variables: `nom_variable= valor_variable`. `clear nom_variable` en neteja el valor i `clear all` neteja totes les variables definides per l'usuari.
- Per sortir: `quit` o `exit`.

→ 2

Vectors i polinomis

En aquesta pràctica, veurem com treballar amb vectors i amb polinomis en Matlab/Octave. Pel que fa als darrers, aprendrem a aproximar les seves arrels i a fer diferents operacions i factoritzacions.

2.1. Vectors

A Matlab/Octave, els vectors s'expressen com una successió de nombres, separats per comes o espais, i delimitats per parèntesis quadrats. Així, per exemple, el vector de $\mathbb{R}^5$

$$v = (1,2,3,4,5)$$

s'expressaria en Matlab/Octave com

```
>> v=[1,2,3,4,5] % creacio d'un vector
v =
   1   2   3   4   5
```

i podem accedir a cadascun dels seus elements v_i, que es numeren començant per l'1 (i no pas pel zero), posant el nom del vector seguit de l'índex de l'element que volem trobar entre parèntesis rodons:

```
>> v(1) % primer element del vector
ans =
     1
>> v(5) % cinque element del vector
ans =
     5
>> v(end) % end es el darrer index
ans =  5
```

Si demanem un índex superior o inferior, ens donarà un error de fora de rang tant en Matlab

```
>> v(0) % dona un error, els indexs comencen a 1
??? Subscript indices must either be real positive integers or logicals.
>> v(6) % dona un error, nomes hi ha 5 components
??? Index exceeds matrix dimensions.
```

com en Octave

```
>> v(0) % dona un error, els indexs comencen a 1
error: invalid vector index = 0
```

És interessant notar que podem fer les operacions pròpies de l'espai vectorial de $\mathbb{R}^n$, la suma de vectors i el producte per escalars, a través de la mateixa notació de suma i producte que s'usa per als escalars. Per exemple:

```
u=[4,1,6,7,1]
u =
   4   1   6   7   1
>> u+v % sumem dos vectors
ans =
    5    3    9   11    6
>> 2*u % multipliquem u per 2
ans =
    8    2   12   14    2
```

Evidentment, cal que les dimensions (que podem trobar mitjançant l'ordre `length()`) siguin les mateixes. Així per exemple, en Matlab

```
>> u+w % error: vectors de diferent dimensio
??? Error using ==> plus
Matrix dimensions must agree.
```

mentre que en Octave

```
>> w=[1,2]; length(w) % dimensio d'un vector
ans =  2
>> u+w % error: vectors de diferent dimensio
error: operator +: nonconformant arguments (op1 is 1x5, op2 is 1x2)
error: evaluating binary operator `+' near line 29, column 2
```

2.1.1 Algunes operacions amb vectors

Altres operacions útils són el producte escalar de dos vectors de la mateixa dimensió, que es crida a través de la funció `dot(u,v)`

```
>> dot(u,v) % producte escalar de vectors u i v
ans =  57
```

i el producte vectorial per a vectors de $\mathbb{R}^3$ (i, per tant, de tres components)

```
>> u=[1,1,0]; v=[0,1,1];
>> cross(u,v) % producte vectorial de u i v de R^3
ans =
   1  -1   1
```

que dóna com a resultat un vector de $\mathbb{R}^3$. També ens pot ser útil l'ordre `sort()`, que ens ordena un vector en ordre creixent

```
>>  v=[3,2,1];
>> sort(v) % ordena el vector v en ordre creixent
ans =
   1   2   3
```

També tenim l'ordre `sum()`, que dóna la suma d'un vector, i l'ordre `prod` que en dóna el producte:

```
>> v=[1,2,3,4];
>> sum(v) % suma les components d'un vector
ans =
    10
>> prod(v) % multiplica les components d'un vector
ans =
    24
```

Podem crear també vectors que continguin les sumes acumulades d'altres vectors, a través de l'ordre `cumsum()` (de *cumulative sum*):

```
>> cumsum(v) % vector amb les sumes acumulades.
ans =
     1     3     6    10
>> cumprod(v) % vector amb productes acumulats.
ans =
     1     2     6    24
```

2.2. Els polinomis i les seves arrels

Els polinomis en `Matlab/Octave` es representen com a vectors indicant simplement quins són els coeficients de major a menor grau. Així, per exemple, el polinomi

$$p(x) = x^7 + 3x^2 - 1$$

quedaria representat pel vector

```
p= [1,0,0,0,0,3,0,-1]
```

i les operacions de suma i producte per escalars de la secció anterior també valdrien de la mateixa manera.

Per avaluar un polinomi en un valor determinat, podem usar la funció `polyval()`, que pren com a arguments el polinomi i el punt on la volem avaluar, encara que també ho podríem fer directament:

```
>> x=0.5
x =
   0.500000000000000
>> polyval(p,x) % avaluem p en x
ans =
  -0.242187500000000
>> x^7+3*x^2-1 % ha de donar el mateix, pero millor polyval()
ans =
  -0.242187500000000
```

Molt sovint voldrem trobar les arrels dels polinomis, cosa que aconseguirem mitjançant l'ordre `roots()`. Per exemple, les arrels del polinomi

$$q(x) = x^2 - 5x + 6,$$

que són 3 i 2, les calcularíem així

```
>> q=[1,-5, 6]
q =
   1  -5   6
>> roots(q)
ans =
   3
   2
```

Observem que, en aquest cas, el resultat de la funció `roots()` és un vector representat en forma vertical (de fet a la pràctica 5 veurem que és una matriu columna). Per poder manipular-lo i accedir als seus elements, l'haurem d'assignar a una variable. Per exemple, per comprovar-ne el resultat, faríem el següent:

```
>> arrels=roots(q);
>> arrels(1),arrels(2)
ans =  3
ans =  2
>> polyval(q,arrels(1))
ans = 0
>> polyval(q,arrels)
ans =
   1.0e-15 *
   0.888178419700125
   0.888178419700125
```

que és pràcticament zero. Si ho haguéssim executat en `Octave`, hauríem vist que el resultat era zero.

La darrera ordre és molt interessant, ja que li hem passat un vector com a segon argument de la funció `polyval`, i el resultat és un nou vector que té com a element k-èsim el

resultat d'avaluar el polinomi en la component *k*-èsima del vector argument. Aquest tipus d'*aritmètica vectorial* permet agilitar moltes operacions i és un dels punts forts del llenguatge `Matlab/Octave`.

Els polinomis no tenen per què ser de grau 2, sinó que poden ser de graus superiors, per exemple el polinomi següent:

$$r(x) = x^3 - 6x^2 + 11x - 6 = (x-3)\cdot(x-2)\cdot(x-1)$$

té per arrels 1, 2 i 3, i el seu càlcul en `Matlab/Octave` seria el següent:

```
>> r=[1,-6,11,-6]
r =
     1    -6    11    -6
>> roots(r)
ans =

   3.000000000000002
   1.999999999999998
   1.000000000000000
```

i, per tant, veiem que `Matlab/Octave` (com qualsevol paquet de càlcul numèric) pot no donar un resultat exacte sinó una aproximació. Com s'ho fa el `Matlab/Octave` per fer aquest càlcul? Podríem pensar que usa la fórmula que existeix per al càlcul de les arrels d'un polinomi de tercer grau. Tanmateix, per a polinomis de grau més gran que quatre, no existeix cap fórmula i les arrels s'han d'aproximar per mitjà d'algun *mètode numèric*, com veurem a la pràctica dedicada als zeros de funcions. Un exemple interessant és el que passa quan intentem trobar les arrels del polinomi

$$p(x) = (x+1)^7 = x^7 + 7x^6 + 21x^5 + 35x^4 + 35x^3 + 21x^2 + 7x + 1$$

que és, amb multiplicitat 7, -1:

```
>> p=[1,7,21,35,35,21,7,1]
p =
     1     7    21    35    35    21     7     1
>> arrels=roots(p) % calcul arrels
arrels =
  -1.008976439902066
  -1.005627465048996 + 0.007012224774421i
  -1.005627465048996 - 0.007012224774421i
  -0.998023795747401 + 0.008801238285198i
  -0.998023795747401 - 0.008801238285198i
  -0.991860519252570 + 0.003934467943721i
  -0.991860519252570 - 0.003934467943721i
```

el resultat és sorprenentment poc acurat, ja que amb prou feines tenim dos decimals correctes. En canvi, si mirem de calcular quin és el resultat d'avaluar el polinomi en aquestes arrels, tindrem que el resultat és força correcte:

```
>> polyval(p,arrels) % avaluem p a les arrels
ans =
   1.0e-14 *
 -0.532907051820075
 -0.466293670342566 - 0.013964523981613i
 -0.466293670342566 + 0.013964523981613i
 -0.555111512312578 - 0.016826817716975i
 -0.555111512312578 + 0.016826817716975i
 -0.355271367880050 - 0.006852157730108i
 -0.355271367880050 + 0.006852157730108i
```

L'ús d'un paquet numèric com Matlab/Octave no ens dispensa de pensar sinó que ens convida a fer-ho des d'un altre punt de vista!

2.2.1 Operacions amb polinomis

Producte i divisió de polinomis

Recordem que dos polinomis es poden multiplicar a través de la multiplicació de polinomis. Per exemple, si volem fer el producte de

$$p(x) = x^3 + 1 \quad \text{i} \quad q(x) = x^2 + 2x - 2$$

que donarà com a resultat un nou polinomi de grau 5

$$p(x) \cdot q(x) = x^5 + 2x^4 - 2x^3 + x^2 + 2x - 2$$

i intentem fer el següent en Matlab ens donarà un error:

```
>> p*q % aixo donara un error
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

mentre que si ho fem en Octave l'error serà el següent:

```
>> p=[1,0,0,1] % p(x)= x^3+1
p =
   1   0   0   1
>> q=[1,-2,2] % q(x)= x^2-2*x+2
q =
   1  -2   2
>> p*q % aixo donara un error
error: operator *: nonconformant arguments (op1 is 1x4, op2 is 1x3)
```

La raó és que Matlab/Octave interpreta que els vectors dels coeficients de p i q són matrius fila i el producte l'interpreta com a producte de matrius. Com que les dimensions són incompatibles, el resultat és un error. Per trobar correctament el producte de vectors, cal que usem la instrucció `conv(p,q)`, que ens dóna l'anomenat producte de

convolució de dos vectors i que conté els coeficients del polinomi resultat de multiplicar els polinomis representats pels vectors `p` i `q`

```
>> conv(p,q) % producte de polinomis p i q
ans =
   1  -2   2   1  -2   2
```

que és el resultat correcte i coincideix amb `conv(q,p)`. Per exemple, per fer el quadrat d'un polinomi

$$r^2(x) = (x+1)^2 = x^2 + 2x + 1$$

faríem

```
>> r=[1,1] % r(x)= x+1
r =
   1   1
>> conv(r,r) % resultat (x+1)^2
ans =
   1   2   1
```

A part del producte, també podem fer la divisió entre dos polinomis. Recordem que la divisió de dos polinomis dóna com a resultat un quocient i un residu de la divisió. Així, donats els polinomis

$$p_1(x) = x^4 - 1 \quad \text{i} \quad p_2(x) = x^3 - 1$$

aleshores la divisió de p_1 per p_2 és

$$\frac{p_1(x)}{p_2(x)} = q(x) + \frac{r(x)}{p_2(x)}$$

on q i r són polinomis, i el grau de r és menor que el de p_2. A `Matlab/Octave` usaríem l'ordre `deconv(p1,p2)` (que és l'ordre oposada a la convolució)

```
>> p1=[1,0,0,0,-1] % p_1(x)= x^4 -1
p1 =
   1   0   0   0  -1
>> p2=[1,0,0,-1] % p_2(x)= x^3 -1
p2 =
   1   0   0  -1
>> deconv(p1,p2) % divisio polinomial
ans =
   1   0
```

que només ens dóna el valor de q. Si volem que també ens retorni el valor del residu, cridarem aquesta ordre:

```
>> [q,r]=deconv(p1,p2) % p_1(x)= q(x)*p_2(x) + r(x)
q =
   1   0
r =
   0   0   0   1  -1
```

que és la manera de rebre funcions que retornin dos elements, en aquest cas q i r. Ara comprovarem que

$$q(x) = x \quad \text{i} \quad r(x) = x - 1$$

són el resultat de la divisió dels polinomis, és a dir, que es compleix:

$$p_1(x) = q(x) \cdot p_2(x) + r(x)$$

```
>>conv(q,p2)+r % comprovacio, dona p1
ans =
   1   0   0   0  -1
```

Cal anar amb compte amb aquestes ordres quan els primers coeficients d'un polinomi valen zero o són molt petits, ja que això ens pot donar un error o resultats inconnexos:

```
>> g=[0,0,0,1,-2] % ingenuament x-2
g =
   0   0   0   1  -2
>> l=[1,-2] % realment x-2
l =
   1  -2
>> h=[1,3,4,2] % x^3+3x^2+4*x+2
h =
   1   3   4   2
>> [f,r]=deconv(h,l) % divisió correcta
f =
    1    5   14
r =
    0    0    0   30
>> [f,r]=deconv(h,g) % especific de Matlab
??? Error using ==> deconv at 21
First coefficient of A must be non-zero.
```

mentre que en `Octave` no ens farà l'operació:

```
f = 0
r =
   1   3   4   2
```

que és un resultat incorrecte. Per treure els zeros que apareguin al començament d'un vector caldrà que indiquem manualment els elements posteriors als zeros inicials (a la pràctica següent veurem com indicar rangs d'índexs):

```
>> g=[0,0,0,1,-2] % ingenuament x-2
g =
   0   0   0   1  -2
>> g([4,5]) % seleccionem a partir del 4
ans =
   1  -2
```

Derivació i integració de polinomis

Finalment, donat un polinomi de grau n

$$p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$

ens pot interessar trobar-ne la derivada

$$p'(x) = na_n x^{n-1} + (n-1)a_{n-1}x^{n-2} + \cdots + a_1$$

o la primitiva P amb $P' = p$ i $P(0) = 0$

$$P(x) = \frac{a_n}{n+1}x^{n+1} + \frac{a_{n-1}}{n}x^n + \cdots + \frac{a_1}{2}x^2 + a_0 x$$

(recordem que qualsevol altra primitiva l'obtenim sumant una constant a P). Aquestes operacions sobre polinomis es poden fer usant les ordres `polyder()` i `polyint()` de `Matlab/Octave`:

```
>> p=[1,2,3,-1] % p(x)=x^3+2*x^2+3*x-1
p =
   1   2   3  -1
>> pder=polyder(p) % derivem p(x) resp. x
pder =
   3   4   3
>> polyint(pder) % primitiva de p(x) amb p(0)=0
ans =
   1   2   3   0
```

Observem que, com que $p(0) = -1 \neq 0$, el resultat d'aplicar primer `polyder()` i després `polyint()` a p no ha estat p sinó $p+1$. Tant `polyder()` com `polyint()` tenen arguments opcionals que no tractem i que podeu consultar a l'ajuda de `Matlab/Octave`.

Finalment, pot ser útil, donats dos polinomis p_1 i p_2, conèixer la descomposició en fraccions simples del quocient p_1/p_2. Per exemple, això ens permet calcular la descomposició

$$\frac{s^2+s+1}{s^3-5s^2+8s-4} = \frac{-2}{s-2} + \frac{7}{(s-2)^2} + \frac{3}{s-1}$$

a través de l'ordre `residue()`. Aquesta es crida de forma lleugerament diferent en `Matlab` i en `Octave`. De tota manera, podem fer-ho de forma unificada si ens assegurem que el grau del numerador és més petit que el grau del denominador (per reduir el grau, sempre podem usar la funció `deconv()` que acabem de veure). Quan es compleixi aquesta condició, usarem l'ordre `[r, p] = residue(b,a)`, que ens donarà un vector `r` i un vector de pols `p` de manera que

$$\frac{b(s)}{a(s)} = \frac{r_1}{(s-p_1)} + \ldots \frac{r_n}{(s-p_n)}$$

amb la convenció que si els pols del vector `p` apareixen diversos cops les multiplicitats es posen de manera creixent. Veiem-ho en un exemple:

```
>> b = [1,1,1]; % b(s)=s^2+s+1
>> a = [1,-5,8,-4]; % a(s)= s^3-5s^2+8s-4
>> [r,p] = residue(b,a) % fraccions simples b(s)/a(s)
r =

   -2.000000000000011
    7.000000000000008
    3.000000000000008
p =
    2.000000000000000
    2.000000000000000
    1.000000000000001
```

Aquí el vector `r` és el numerador de cadascun dels termes i el vector `p` són els pols que apareixeran als denominadors. Observem que, com que l'arrel 2 apareix repetida, això significa que el coeficient `7.000000000000008` és el que multiplica el factor de $1/(x-2)^2$.

2.3. Exercicis

2.1. Donat el polinomi següent, responeu les preguntes següents:

$$p(x) = 4x^6 - 36x^5 + 56x^4 + 2x^3 - 19x^2 + 37x - 14$$

1. Quant val $p(0.5)$?
2. Quant val $p'(-0.5)$?
3. Quantes arrels reals té?
4. Quina és l'arrel més gran real?
5. Quant val la suma de les arrels reals?
6. Quant val la part imaginària més petita d'una arrel?

Resposta: 2.4375, 17.5, 4, 7, 8.51344794331483 i -0.756969524226409.

2.2. Donat el quocient de polinomis següent,

$$\frac{x+3}{(x-2)(x-1)(x^2-3x-40)}$$

i la seva descomposició en fraccions simples,

$$\frac{x+3}{(x-2)(x-1)(x^2-3x-40)} = \frac{A}{(x-8)} + \frac{B}{(x-2)} + \frac{C}{(x-1)} + \frac{D}{(x+5)}$$

quant val A?

Resposta: 0.0201465201465202

2.3. El polinomi següent té un parell d'arrels complexes conjugades. Indiqueu el valor absolut de la seva part imaginària:

$$x^6 - 7x^5 + \frac{51}{4}x^4 + \frac{17}{4}x^3 - \frac{161}{4}x^2 + \frac{201}{4}x - 21$$

Resposta: 0.8660254037844...

2.4. El polinomi següent p té una única arrel real α:

$$p(x) = x^7 + \frac{1}{49}x^5 + x^3 + \frac{1}{2}x + 1$$

Quant val $p'(\alpha)$?

Resposta: 3.74122044156494...

2.4. Esquema de la pràctica

- Vectors: entre `[ ]` i separats per espais o comes, per exemple, `v=[1,2,3,4]`.
  - Operacions sobre vectors `length(v)` (longitud d'un vector), `sum(v)` (suma de les components), `prod(v)` (producte de les components), `max(v)`, `min(v)`.
  - Accés a una component: `v(index_component)`. Els vectors s'indexen començant per `1` i, per referir-nos al darrer índex, podem usar `end`.
  - Assignació d'una component: `v(index_component)=valor`.
  - Operacions vectorials: `escalar*v` (esclar per vector), `u+v` (suma de vectors).
  - Operacions element a element: `u.*v` (producte element a element), `u./v` (divisió element a element), `u.^v` (potència element a element).
  - Funcions element a element: `abs(v)` (valor absolut de cada component), `cos(v)`.
  - Altres: `dot(u,v)` (producte escalar de dos vectors), `cross(u,v)` (producte vectorial de dos vectors de $\mathbb{R}^3$), `cumsum(v)` (vector de sumes acumulades), `cumprod(v)` (vector de productes acumulats), `sort(v)` (ordena `v` per ordre creixent).
- Polinomis: s'expressen com un vector que conté els coeficients de major a menor grau, p.e. `p=[1,2,3,4]` és $p(x) = x^3 + 2x^2 + 3x + 4$.
  - Avaluació `polyval(p,x)`: avalua el polinomi `p` en `x` (número o vector).
  - Arrels d'un polinomi: `roots(p)`.
  - Producte de polinomis: `conv(p,q)`.
  - Divisió de polinomis: `[q,r]=deconv(p1,p2)` dóna $p_1(x) = q(x) \cdot p_2(x) + r(x)$.
  - Derivació `polyder(p)` i primitivització `q=polyint(p)` (amb $q(0) = 0$).
  - Descomposició en fraccions simples de $b(s)/a(s)$: `[r,p] = residue(b,a)`.

→ 3

Representació gràfica

Una de les eines més útils en l'anàlisi i la presentació de resultats numèrics és la seva representació gràfica. Tant `Matlab` com `Octave` disposen de moltes possibilitats de representació gràfica. En aquesta pràctica, veurem la instrucció bàsica per representar punts en coordenades cartesianes i, posteriorment, ens centrarem en la representació de funcions d'una variable.

3.1. L'ordre `plot()`

La instrucció bàsica per dibuixar és l'ordre `plot()`, que conté nombroses opcions que es poden consultar fent `help plot`. En la seva variant més bàsica, `plot(x,y)`, on `x` i `y` són dos vectors de la mateixa dimensió, ens dibuixa els punts formats per les parelles de coordenades x i y determinades per aquests vectors:

```
>> x=[0, 1, 2, 3, 4]
x =
   0   1   2   3   4
>> y=[1, 2, 4, 1, 1]
y =
   1   2   4   1   1
>> plot(x,y) % segments que uneixen (0,1),(1,2),(2,4)...
>> plot(x) % el mateix que plot([1 2 3 4 5],x)
>> plot(y) % el mateix que plot([1 2 3 4 5],y)
```

En prémer Retorn en cadascuna d'aquestes ordres se'ns mostrarà un dibuix amb aquestes instruccions (veieu-ho a la figura 3.1). Encara que sigui molt bàsic, és important ja fer les observacions següents:

1. El motor gràfic de `Matlab` i el d'`Octave` són diferents; per tant, la finestra pot ser lleugerament diferent.
2. A `Matlab` se'ns mostrarà una finestra amb la possibilitat d'editar les gràfiques, ampliar-les, exportar-les a diferents formats, etc...

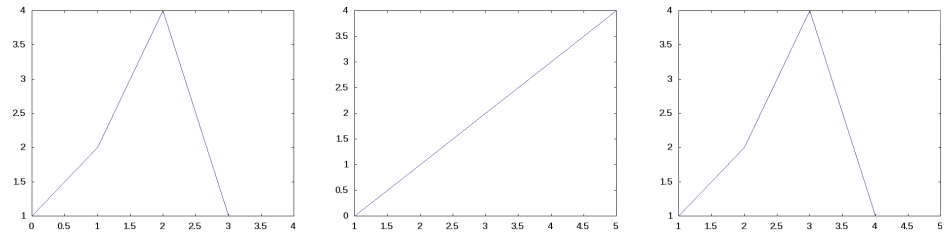


Fig. 3.1
Resultat de dibuixar plot(x,y), plot(x) i plot(y) (observeu com, per la tria del vector x que hem fet, la tercera i la primera gràfiques són iguals, llevat d'un desplaçament d'una unitat en l'eix de les *x*.

3. Per defecte, si només donem un sol vector com a argument, se suposa que aquest és l'ordenada a representar i que les abscisses són els nombres $1, 2, \ldots, n$, on n és la longitud del vector que estem representant.
4. Per tancar la figura, podem fer `close` des de la línia d'ordres o tancar directament la finestra.

És important que puguem modificar l'estil de les línies que hi apareixen. A `Matlab/Octave` això s'aconsegueix mitjançant l'ús de cadenes de text (*strings*) que modifiquen l'aparença del dibuix. Com a mostra d'això representarem ara al pla complex les arrels complexes del polinomi

$$p(x) = x^{12} + 1$$

que són $e^{i\pi/12 + 2\pi ik/12}, k = 0, 1, \ldots, 11$

```
>> p=[1 0 0 0 0 0 0 0 0 0 0 0 1]
p =
   1   0   0   0   0   0   0   0   0   0   0   0   1
>> arrels=roots(p)
arrels =
  -0.965925826289069 + 0.258819045102521i
  ...
   0.965925826289069 - 0.258819045102521i
   0.707106781186548 + 0.707106781186548i
   0.707106781186548 - 0.707106781186548i
>> plot(real(arrels),imag(arrels),'ro')
```

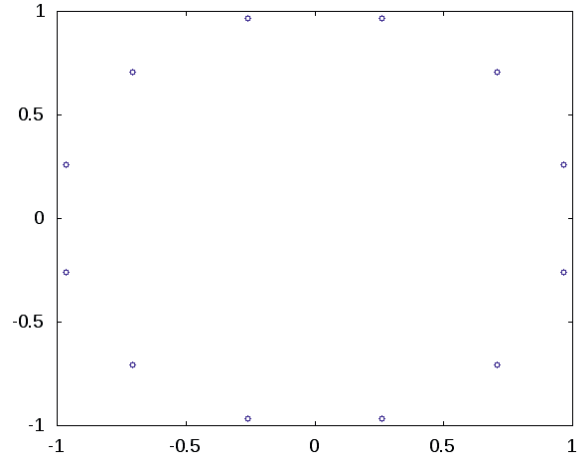


Fig. 3.2
Arrels del polinomi $x^{13}+1$ amb l'ordre plot(real(arrels), imag(arrels),'ro').

En aquest dibuix, que podem veure a la figura 3.2, la cadena de text `'ro'` significa que volem que el color de la línia sigui vermell (`r` de *red*) i que els punts es representin per mitjà de rodones buides, cosa que indiquem per `o`). Altres modificadors els podem conèixer fent `help plot`; els més importants són els següents:

- Tipus de punts: creus `+`, estrelles `*`, cercles `o`, creus en aspa `x`, triangles `^` o punts `.` (molt petits).
- Tipus de línies: línies contínues `-`, discontínues `--`, puntejades `:`.
- Colors: vermell `r`, verd `g`, blau `b`, negre `k`

Així, per exemple, l'ordre `plot(primes(100),'ko--')` mostra els 100 primers nombres primers amb cercles negres units per un traç discontinu.

Tornant a l'exemple de les arrels del polinomi, per veure més clarament que totes aquestes arrels tenen modul 1 podem fer que la relació d'aspecte sigui 1, posarem una ordre anterior que sigui `axis equal` que també ens permet determinar quina finestra volem visualitzar:

```
>> close % tanquem possibles figures obertes
>> axis([-1,1,-1,1],'equal'); hold on
>> plot(real(arrels),imag(arrels),'o'); hold off
```

on indiquem que la finestra del dibuix serà el quadrat $(x,y) \in [-1,1] \times [-1,1]$ i que la finestra s'ajustarà perquè les unitats de les x i les y siguin les mateixes. Les ordres `hold on` i `hold off` modifiquen el comportament d'un dibuix a mesura que hi anem afegint més gràfiques i propietats. L'ordre `hold on` manté la gràfica i hi va afegint les instruccions a continuació. L'ordre `hold off` torna al mode en què cada nova instrucció de dibuix es fa en una nova gràfica.

En cas que dibuixem valors complexos, ens pot ser útil també l'ordre `compass()` que dibuixa nombres en el pla complex com vectors, tal com mostra la figura 10.1.

3.1.1 Rangs de valors

Ja veiem que no és una manera molt eficient de representar funcions haver d'escriure tots els valors de les abscisses que volem representar. Per tant, necessitem tenir alguna forma d'indicar rangs de valors. La primera manera de determinar un rang en `Matlab/Octave` és a través dels operadors `inici:condicio_parada`, que creen un vector que comença a `inici` i conté `inici+1, inici+2, ...` mentre que els elements siguin menors o iguals a `condicio_parada`:

```
>> 1:5 % d'1 a 5 d'un en un
ans =
   1   2   3   4   5

>> 1:5.5 % la condicio de parada es < 5.5
ans =
   1   2   3   4   5
```

Si volem que el pas no sigui 1 sinó una altra quantitat, ho indicarem a través d'un nou argument entre `inici` i `parada`, `inici:pas:condicio_parada`,

```
>> 0:0.25:1.1
ans =
    0.00000    0.25000    0.50000    0.75000    1.00000
>> -2:0.5:1
ans =
   -2.0000   -1.5000   -1.0000   -0.5000    0.0000    0.5000    1.0000
>> 1:2:2*pi
ans =
   1   3   5
```

Observem que el darrer terme no té per què ser l'extrem superior sinó que la condició de parada és que els elements del rang siguin inferiors o iguals a l'extrem superior. Si volem assegurar-nos que els punts inclouen els dos extrems, podem usar l'ordre `linspace(inici,final,num_punts)`

```
>> v=linspace(0,2,6) % igual a 0:0.2:2
v =
   0.00000   0.40000   0.80000   1.20000   1.60000   2.00000
```

i el pas és `(2-0)/(6-1)=0.4`.

Amb això ja és més fàcil representar funcions, com ara polinomis

```
>> x=0.5:0.01:3; % recordem el ; no mostra el resultat, convenient
>> p=[1,-3,2]; % polinomi x^2-3x+2
>> y= polyval(p,x); % avaluem p en les abscisses x
>> plot(x,y),grid
```

i l'ordre `grid` mostra una graella a la gràfica on podem veure més clarament que les arrels són a 1 i a 2. Podríem pensar que, per avaluar el polinomi x^2-3x+2 ho podem fer directament a la línia d'ordres, però observem l'error següent en `Matlab`:

```
>> y=x^2-3*x+2
??? Error using ==> mpower
Matrix must be square.
```

que a `Octave` és essencialment el mateix:

```
>> y=x^2-3*x+2
error: for A^b, A must be square
```

Això passa perquè a `Matlab/Octave` s'interpreta que, com que `x` és un vector (que és un cas particular de matriu), el quadrat és el producte de dues matrius. Com que les dimensions no concorden (hauria de ser una matriu quadrada), obtenim el missatge d'error.

Per especificar que un operador que es pugui aplicar indistintament a matrius i a escalars s'apliqui element a element, caldrà que hi posem un `.` immediatament abans. Així,

```
>> y=x.^2-3*x+2
```

obtindrem el mateix que trobàvem abans amb l'ordre `polyval()`. Observem, per contra, que no ha calgut que posem un `.*` en la multiplicació del vector `x` per `3`, ja que la multiplicació d'un vector per un escalar té un significat únic. Això també ho hem de fer per als operadors `*` i `/` (que és la multiplicació per l'invers). A tall d'exemple, tenim la taula següent:

Expressió matemàtica (x vector)	Expressió en `Matlab/Octave`
$g(x) = e^{-1/x} \sin \frac{1}{x}$:	`y= exp(-1./x).*sin(1./x);`
$g(x) = \log(\cos(x^2))$:	`y= log(cos(x.^2));`
$g(x) = \frac{\sqrt{x}}{\sin x}$:	`y=sqrt(x)./sin(x);`
$g(x) = \frac{1}{x+e^x}$:	`y= 1./(x+exp(x));`
$g(x) = e^{x^2} \sin x^2$:	`y=exp(x.*x).*sin(x.*x);`
$g(x) = e^{-x^2/2} \sin x^2$:	`y=exp(-(x.^2)/2).*sin(x.^2);`

Aplicació: dibuixar funcions definides a trossos

Ara farem servir tot el que hem après per dibuixar la funció següent definida a trossos ([12], exercici 30, pàgina 55) en l'interval $[-1,4]$:

$$f(x) = \begin{cases} \sin(x), & \text{si } x < 0 \\ 1-\cos x, & \text{si } 0 \le x < \pi \\ \cos x, & \text{si } x \ge \pi. \end{cases} \tag{3.1}$$

Com que les ordres `hold on` i `hold off` ens permeten afegir diverses gràfiques a un sol dibuix i anirem definint els diferents rangs dels intervals:

```
x1=-1:0.01:0; % rang a [-1,0]
x2=0:0.01:pi; % rang a [0,pi]
x3=pi:0.01:4; % rang a [pi,4]
plot(x1,sin(x1)) % funcio a [-1,0]
hold on % afegim les grafiques
plot(x2,1-cos(x2)) % funcio a [0,pi]
plot(x3,cos(x3)) % funcio a [pi,4]
hold off
```

i obtindrem la gràfica de la figura 3.3.

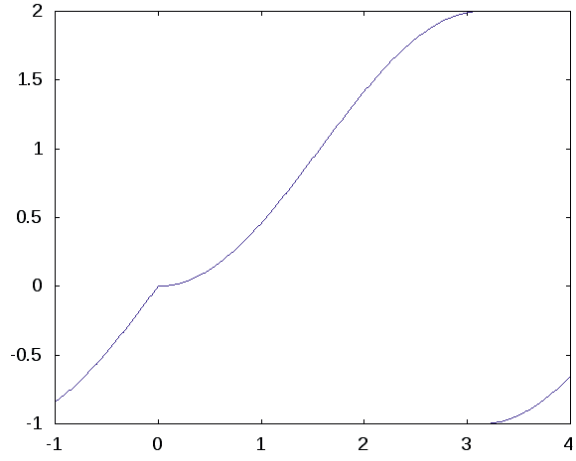


Fig. 3.3
Gràfica de la funció (3.1) a l'interval [–1,4].

Observem que també podríem haver fet la representació sense fer ús de l'ordre `hold on/off` de la manera següent:

```
>> plot(x1,sin(x1), x2,1-cos(x2), x3,cos(x3))
```

ja que l'ordre `plot()` accepta un nombre arbitrari de parelles de vectors abscisses i ordenades.

Complement opcional. Més operacions amb vectors

L'aritmètica vectorial ens permet comptar el nombre d'elements d'un vector que satisfan una condició, amb la convenció que `1` és cert i `0` és fals. Com a exemple molt senzill, imaginem que volem comptar quants enters entre 1 i 8 són menors que 3 estrictament (resposta, 2):

```
>> v=1:8
v =
     1     2     3     4     5     6     7     8
>> v<3 % nou vector amb 1 si v(i)<3 i 0 altrament
ans =
     1     1     0     0     0     0     0     0
>> v<=3 % nou vector amb 1 si v(i)<=3 i 0 altrament
ans =
     1     1     1     0     0     0     0     0
>> v==3 % l'unica component nozero es 3
ans =
     0     0     1     0     0     0     0     0
>> (v<5).*(v>2) % 1 quan v(i)<5 i v(i)>2
ans =
     0     0     1     1     0     0     0     0
>> max((v<5),(v>2)) % 1 quan v(i)<5 o v(i)>2
ans =
     1     1     1     1     1     1     1     1
>> max((v<2),(v>4)) % 1 quan v(i)<2 o v(i)>4
ans =
     1     0     0     0     1     1     1     1
>> isprime(v) % 1 quan v(i) es primer
ans =
     0     1     1     0     1     0     1     0
```

Un cop amb això, podem comptar els elements d'un vector que satisfan una condició usant l'ordre `sum()`. Per exemple, per veure quants primers hi ha entre 1 i 1000, faríem el següent:

```
>> v=1:1000;
>> esprimer=isprime(v);
>> sum(esprimer)
ans =
   168
```

3.2. Fitxers m

És convenient poder definir funcions que haguem d'utilitzar i assignar-los un nom perquè les puguem cridar sempre que ho necessitem. A Matlab/Octave, això es fa a través de les funcions `m`, que són fitxers que inclouen la definició de la funció i que han d'estar en el mateix directori de treball (és important comprovar-ho en el moment d'iniciar el programari). Per editar una funció, que anomenarem `funcio.m` (per exemple) escrivim des de la línia d'ordres

```
>> edit('funcio.m')
```

i se'ns obre un editor per defecte. En el cas del Matlab, l'editor és el que porta incorporat per defecte (si no existeix el fitxer, ens demanarà per crear-lo) i, en el cas de l'Octave, serà el que porti incorporat el sistema per defecte. En aquest editor, haurem de posar la declaració de la funció (i res més):

```
function y=funcio(x)
y=cos(x.^2);
```

i, un cop guardem el fitxer, ja el podem cridar per línia d'ordres. Per exemple

```
>> funcio(sqrt(pi))
ans =
    -1
>> funcio([1,2,3])
ans =
   0.540302305868140   -0.653643620863612   -0.911130261884677
```

Observem el següent:

- Les funcions es declaren de la forma

```
function variable_dependent = nom_funcio(variable_independent)
```

  És molt important que el nom de la funció (en el nostre cas, `'funcio.m'`) sigui el mateix que el nom del fitxer.
- Cal guardar la funció perquè estigui accessible des de la línia d'ordres.
- El ; que apareix és perquè no es mostri el resultat de l'operació.
- Podem fer també operacions intermèdies amb altres variables, però només es retornarà la variable `y` en aquest cas.

També podem fer funcions que retornin diversos valors i que prenguin diversos arguments. Per exemple, a l'ajuda del Matlab/Octave apareix la funció següent per calcular la mitjana i la desviació típica d'un vector:

```
function [mitjana,desviacio] = estadistiques(x)
n = length(x);
mitjana = sum(x)/n;
desviacio = sqrt(sum((x-mitjana).^2/n));
```

que té com a resultat, per a un vector $x = (x_1, \ldots, x_n)$

$$\texttt{mitjana} = \frac{1}{n}\sum_{i=1}^{n} x_n, \quad \texttt{desviacio}^2 = \frac{\sum_{i=1}^{n}(x - \texttt{mitjana})^2}{n}$$

i es crida de la manera següent:

```
>> x=1:10
x =
     1     2     3     4     5     6     7     8     9    10
>> [mitjana,desviacio]=estadistiques(x)
mitjana =
   5.500000000000000
desviacio =
   2.872281323269014
```

3.2.1 Complement opcional. Dibuixant amb l'ordre `fplot()`

Un dels avantatges de definir una funció per mitjà d'un fitxer `.m` és que podem dibuixar-la sense haver d'especificar el rang exacte de valors per al qual volem que es faci el dibuix. Per fer-ho així, usarem l'ordre `fplot()` (de *function plot*) que dibuixa una funció en un interval. Per exemple, la ordre següent dibuixarà la funció $f(x) = \cos(x^2)$, que està definida al fitxer `funcio.m`, a l'interval $(-\sqrt{8\pi}, \sqrt{8\pi})$:

```
>> fplot(@funcio,[-sqrt(8*pi),sqrt(8*pi)]) % dibuixa la 'funcio.m'
```

És important que posem l'arroba a davant del nom de la funció perquè el `Matlab/Octave` sàpiga la funció que ha de dibuixar, com també per indicar l'interval que volem. El modificador `@` el podem situar també sobre les funcions definides pel sistema. Per exemple, per dibuixar la tangent hiperbòlica en $(-2, 2)$

$$f(x) = \tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

faríem

```
>> fplot(@tanh,[-2 2]) % dibuixa la funcio tanh del sistema a [-2,2]
```

En canvi, l'ordre

```
>> fplot(@cos+@sin,[0,pi]) % error no es poden sumar aixi
??? Undefined function or method 'plus'
    for input arguments of type 'function_handle'.
```

ens dóna un error perquè no podem sumar funcions d'aquesta manera. Per fer-ho, hauríem de definir una funció que representés la suma de les dues.

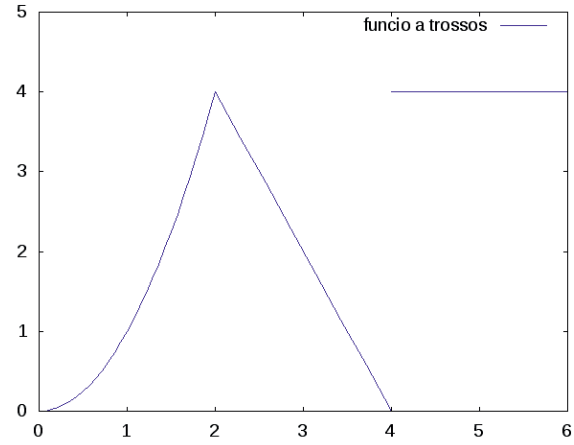


Fig. 3.4
Gràfica de la funció definida a trossos (3.2).

Això ens proporciona una altra manera senzilla de dibuixar funcions definides a trossos. Per exemple, dibuixarem ara la funció següent ([12], problema 29, pàgina 55):

$$f(x) = \begin{cases} x^2, & \text{si } x \leq 2 \\ 8 - 2x, & \text{si } 2 < x \leq 4 \\ 4, & \text{si } x \geq 4 \end{cases} \tag{3.2}$$

Suposant que als fitxers `f1.m`, `f2.m` i `f3.m` hem definit les funcions $x \mapsto x^2$, $x \mapsto 8 - 2x$ i $x \mapsto 4$, respectivament, podem dibuixar aquesta funció definida a trossos usant l'ordre següent:

```
fplot(@f1,[0,2]) % x^2 a [0,2]
hold on % no esborrarem grafiques
fplot(@f2,[2,4]) % 8-2*x a [2,4]
fplot(@f3,[4,6]) % constant 4 a [4,6]
axis([0,6,0,5]) % ajustem la finestra
legend('funcio a trossos')
hold off
```

i obtenim la gràfica de la figura 3.4.

3.3. Modificar i exportar gràfiques en `Matlab/Octave`

Sovint és interessant afegir elements a una gràfica, com ara text, diverses gràfiques, etiquetes, etc. Per fer-ho, hem d'indicar que volem que una gràfica es mantingui i que les noves ordres gràfiques que indiquem volem que se sobreposin a l'existent. Així, si volem dibuixar la gràfica de $1 + x + x^2/2$ i de $exp(x)$, podem fer el següent (hi posem també unes indicacions autoexplicatives):

```
>> x=0:0.01:2; y1=1+x+0.5*x.^2; y2=exp(x); % x i y a representar
>> plot(x,y1,'r'); hold on % mantenim la figura
>> plot(x,y2,'b'); %  dibuixem en blau
>> title('Aproximacio de Taylor de 2n ordre')
```

```
>> xlabel('x') % titol per a l'eix x
>> ylabel('y') % titol per a l'eix y
>> legend('1+x+x^2/2','exp(x)') % llegenda dels plots.
>> hold off % desfem el mantenir la figura.
```

de manera que ens surti un resultat com a la figura 3.5.

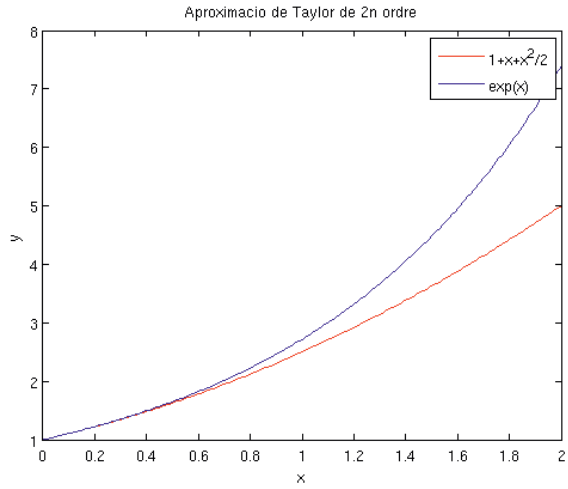


Fig. 3.5 Exemple de dibuix amb Matlab usant títols, llegenda i noms per als eixos. Amb Octave, el dibuix pot ser lleugerament diferent.

Amb vista a l'exportar la figura a algun format com `png`, `jpg` o `pdf` perquè la puguem usar posteriorment, és indubtable que `Matlab` disposa de més recursos que no pas `Octave`. En qualsevol cas, podrem sempre usar l'ordre `print` per exportar-la al format que més ens convingui. Haurem d'indicar, a continuació, `-dformat` on `format` pot ser `png`, `jpg`, `pdf` o molts d'altres que podem conèixer a través de l'ajuda. Per exemple, per obtenir el `png`, fem (amb la finestra de gràfics activa):

```
>> print -dpng 'dibuix.png' % exporta la grafica activa a png.
```

i se'ns crearà el fitxer `dibuix.png` a l'espai de treball. Si ho volem fer en `pdf`, fem

```
>> print -dpdf 'dibuix.pdf' % exporta la grafica activa a pdf
```

3.4. Exercicis

3.1. A partir de l'anàlisi de la gràfica de la funció següent,

$$f(x) = \sin(\sin(\sin(x)+1)+1)$$

quants màxims locals creieu que té la funció en $(-\pi, \pi)$? No compteu els extrems de l'interval.

Resposta: 3

3.2. Representeu gràficament a l'interval $[\frac{-3}{2}, \frac{3}{2}]$ la funció definida a trossos de la manera següent

$$f(x) = \begin{cases} \cosh(4\operatorname{arccosh}(-x)), & x < -1 \\ \cos(4\arccos(x)), & x \in [-1,1] \\ \cosh(4\operatorname{arccosh}(x)), & x > 1 \end{cases}$$

i determineu, a través de la inspecció gràfica, si és contínua en aquest interval. Dibuixeu també a la mateixa figura el polinomi $T_4(x) = 8x^4 - 8x^2 + 1$ i comproveu que són iguals.

3.5. Esquema de la pràctica

- Representació gràfica amb `plot()`:
  - `plot(x,y)`, amb `x` i `y` vectors de la mateixa dimensió, dibuixa els punts amb abscisses `x` i ordenades `y`. També diverses abscisses i ordenades alhora `plot(x1,y1,x2,y2,x3,y3)`.
  - `plot(x,y,'o')` amb cercles, `plot(x,y,'o-')` amb cercles i línies, `plot(x,y,'ro-')` amb cercles i línies vermells ...
  - Control de la finestra de dibuix: `close` tanca la figura, `hold on` afegeix els dibuixos següents a la figura activa, `hold off` canvia aquest comportament.
  - `axis([xmin,xmax,ymin,ymax])` canvia els eixos i `grid on/off` commuta la graella.
  - Anotar les figures: `title('titol de la figura')`, `xlabel('titol de les x')`, `ylabel('titol de les y')`, `legend('llegenda plot1','llegenda plot2')` llegenda dels diferents plots de la figura.
- Exportar la finestra gràfica activa:
  - PNG: `print -dpng 'nom_fitxer.png'`
  - PDF `print -dpdf 'nom_fitxer.pdf'`
- Rangs de valors:
  - `in:fi` és `in, in+1...`, mentre els termes de la successió siguin inferiors o iguals a `fi`. Així, `1.1:4` és el mateix que `[1.1,2.1,3.1]`.
  - `in:pas:fi` és `in, in+pas...` mentre els termes de la successió siguin menor o iguals que `fi`. Així `1:2:5.2` és el mateix que `[1,3,5]`. Si `pas` és negatiu, funciona al revés.
  - `linspace(xmin,xmax,npunts)` retorna `npunts` punts equiespaiats començant en `min` i acabant en `max`.
  - Condicions sobre vectors (opcional): comproven si la condició és certa (`1`) o falsa (`0`) sobre un vector. Per exemple, si `v=1:4 v<3` és `[1,1,0,0]`, `v(v<3)` és `[1,2]` i `sum(1:4<3)` compta els elements que satisfan la condició. Altres condicions `==` (igualtat), `<=` ($\leq$) i `>=` ($\geq$).

- Funcions i fitxers `.m`. Per editar un fitxer `nom_funcio.m`, fem `edit 'nom_funcio.m'` (cal que estigui en un directori amb escriptura):

  - Les funcions amb un argument que retornen un valor són de la forma

```
function y=nom_funcio(x)
y=x.^2; % per exemple
```

    on és molt important que el nom de la funció sigui el mateix que el nom de l'arxiu. Un cop definida així, podem cridar-la des de la línia d'ordres i dibuixar-la amb la funció (opcional) `fplot(@nom_funcio,[a,b])` a l'interval $[a,b]$.

  - Funcions de diversos arguments i que retornen diversos valors

```
function [y1,y2,y3]=nom_funcio(x1,x2)
y1=x1.^2; % per exemple
y2=cos(x2); % per exemple
y3=x1+x2; % per exemple
```

Zeros i extrems de funcions

Un dels primers àmbits on l'ús de mètodes numèrics es fa necessari és en la solució d'equacions no lineals o, equivalentment, per trobar aproximacions als zeros de funcions. En aquesta pràctica, veurem com trobar zeros de funcions d'una variable usant la representació gràfica, el teorema de Bolzano i mètodes numèrics implementats en `Matlab/Octave`. Finalment, aprendrem com aproximar màxims i mínims d'una funció en un interval.

4.1. Localitzar gràficament zeros de funcions

En aquesta pràctica, ens proposem trobar solucions d'una equació no lineal

$$f(x) = 0$$

en un determinat interval $[a,b]$, on $f : [a,b] \mapsto \mathbb{R}$ és una funció com a mínim contínua. Hem vist ja com fer-ho en el cas de polinomis a través de l'ordre `roots()`. Vegem-ho ara per a funcions més generals. Com a exemple, estudiarem les solucions de l'equació no lineal

$$e^x + 3x = 2 + x^2$$

a l'interval $[0,1]$. En primer lloc, ho escrivim com a zero d'una funció:

$$f(x) = e^x + 3x - x^2 - 2$$

que en `Matlab/Octave` ho escriuríem en una funció a través de l'ordre `edit 'funcio.m'` i a l'editor podem posar la definició de la funció

```
function y=funcio(x)
y=exp(x) +3.*x -x.^2 -2;
```

on recordem que hem de posar el punt per indicar que es tracta de producte element a element (així ho podrem aplicar a vectors, cosa que ens serà útil com veurem a continuació). Primerament, representem gràficament la funció a l'interval en qüestió per fer-nos la idea de com és. Recordem que això ho podem fer indicant un rang i representant-ho:

```
>> x=0:0.01:1; y=funcio(x);
>> plot(x,y); hold on
>> grid on; hold off
```

o bé aprofitant que ja hem definit la funció a través de l'ordre `fplot`:

```
>> fplot(@funcio,[0,1]); hold on
>> grid on ; hold off
```

En tots dos casos, a més de la funció, ens dibuixa una graella (*grid*) amb la qual veiem que la funció té un zero a l'interval al voltant de 0.26, tal com es pot veure a la figura 4.1.

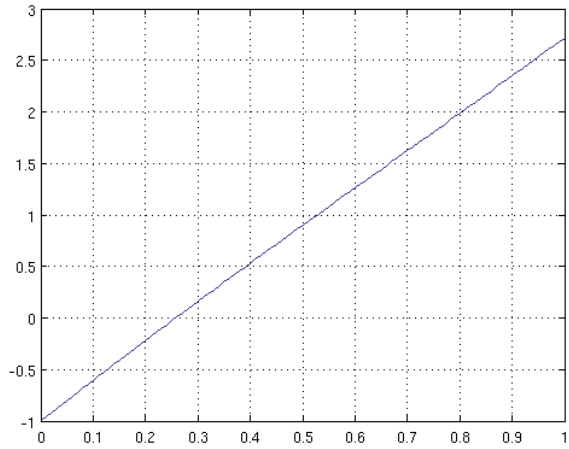


Fig. 4.1
Representació de la funció $f(x)= e^x+3x -x^2-2$ a l'interval [0,1].

Com que la funció és contínua, podem demostrar l'existència d'aquest zero a través del teorema de Bolzano, atès que la funció als extrems de l'interval $[0,1]$ té signes oposats

```
>> funcio(0)*funcio(1)
ans = -2.7183
>> funcio(0.25)*funcio(0.26) % afinem un xic més l'interval.
ans = -2.6567e-04
```

Aquest mètode (a part de la inspecció gràfica) ens permet determinar un interval a on es trobin els zeros de la funció i, per tant, una aproximació inicial per als mètodes iteratius, que expliquem a continuació.

4.1.1 Complement opcional: trobar zeros en un rang

Aquest procediment manual per trobar un interval on la funció prengui valors amb signe contrari als extrems el podem sistematitzar una mica més usant l'ordre `find(condicio)`, que ens dóna el primer valor (i, opcionalment, l'índex) d'un vector que satisfà la `condicio` que indiquem. Així,

```
>> a=0;b=1;pas=0.1;
>> x=a:pas:b; punts= length(x) % punts equiespaiats entre a i b
>> extrems_inferiors=x(1:punts-1) % x(1),...,x(punts-1)
extrems_inferiors =
  0.00000   0.10000   0.20000   0.30000   0.40000   0.50000   0.60000
  0.70000   0.80000   0.90000
>> extrems_superiors=x(2:punts) % x(2),...,x(punts)
extrems_superiors =
  0.10000   0.20000   0.30000   0.40000   0.50000   0.60000   0.70000
  0.80000   0.90000   1.00000
>> canvis_signe=funcio(extrems_superiors).*funcio(extrems_inferiors)
canvis_signe =
  0.604829   0.132214  -0.034945   0.085017   0.477962   1.134293
  2.049369   3.224027   4.665233   6.386883
```

i observem que a l'interval $[0.2, 0.3]$ hi ha l'arrel que busquem, ja que la funció pren valors de signe contrari als extrems de l'interval. Si volem trobar aquest valor sense haver de visualitzar tots els valors del vector `canvis_signe` (cosa que és molt conveninent quan la dimensió de `x` creix), usarem l'ordre `find()`:

```
>> [canvisigne,icanvi] = find(canvis_signe <0)
canvisigne =  1
icanvi =  3
>> [x(icanvi),x(icanvi+1)]
ans =
   0.20000   0.30000
```

que ens diu que a l'interval $[0.2, 0.3]$ hi ha una arrel. El mateix procediment el podríem fer amb `pas=0.001` i obtindríem que el zero és a l'interval $[0.2575, 0.2576]$. La sèrie d'ordres següent ens donaria una aproximació encara millor:

```
>> a=0;b=1;pas=0.0001;
>> x=a:pas:b; punts= length(x) % punts equiespaiats entre a i b
punts =
       10001
>> extrems_inferiors=x(1:punts-1); % x(1)...x(punts-1)
>> extrems_superiors=x(2:punts); % x(2)...x(punts)
>> canvis_signe=funcio(extrems_superiors).*funcio(extrems_inferiors);
>> [canvisigne,icanvi] = find(canvis_signe <0);
>> [x(icanvi),x(icanvi+1)]
ans =
     0.2575    0.2576
```

4.2. L'ordre `fzero()`

Un cop disposem d'un interval on hi hagi un zero de la funció, en el nostre cas `[0.2,0.3]` per exemple, podem usar l'ordre `fzero()` (present a `Matlab` i a `Octave` des de la versió 3.2) que utilitza una sèrie de mètodes interns, per trobar una aproximació d'un zero dins de l'interval que li hem passat

```
>> format long
>> aprox=[0.2,0.3];
>> arrel=fzero(@funcio,aprox) % troba el zero de funcio a  aprox
arrel =  0.257530272027944
>> funcio(arrel) % comprovem que es proper a zero
ans = -5.06792130483547e-08
```

i am Matlab n'obtenim una aproximació lleugerament diferent:

```
>> format long
>> aprox=[0.2,0.3];
>> arrel=fzero(@funcio,aprox) % troba el zero de funcio a l'interval
aprox
arrel =
   0.257530285439861
>> funcio(arrel) % comprovem que es proper a zero
ans =
     4.440892098500626e-16
```

De fet, la funció `fzero()` retorna dos arguments, un amb el valor de l'arrel aproximada (`arrel` en el codi anterior) i l'altre amb el valor la funció avaluada en el zero (`f(arrel)`). Vegem-ho en Matlab:

```
>> [arrel,funcioarrel]=fzero(@funcio,[0.2,0.3])
arrel =
   0.257530285439861
funcioarrel =
     4.440892098500626e-16
```

Per poder assegurar que l'aproximació de l'arrel té un determinat nombre de decimals, per exemple 7, prenem el tall a la setena xifra decimal i comprovarem que la funció pren valors amb signe contrari als números amb aquest tall i a l'aproximació per sobre:

```
>> tall_inferior=0.2575302 % aproximacio per sota
tall_inferior =
   0.257530200000000
>> tall_superior=0.2575303 % aproximacio per sobre
tall_superior =
   0.257530300000000
>> [funcio(tall_inferior),funcio(tall_superior)]
ans =
   1.0e-06 *
  -0.322849076805909   0.055017967515170
```

la qual cosa ens assegura que l'arrel que busquem és 0.2575302..., és a dir, que té els primers set decimals correctes.

Podem dibuixar la funció i el zero per veure-ho més gràficament a través de les ordres següents:

```
>> x=0:0.01:1; plot(x,funcio(x)); hold on;
>> [arrel,funcioarrel]=fzero(@funcio,[0.2,0.3]);
>> plot(arrel,funcioarrel,'ro');
>> grid on; hold off
```

com veiem a la figura 4.2.

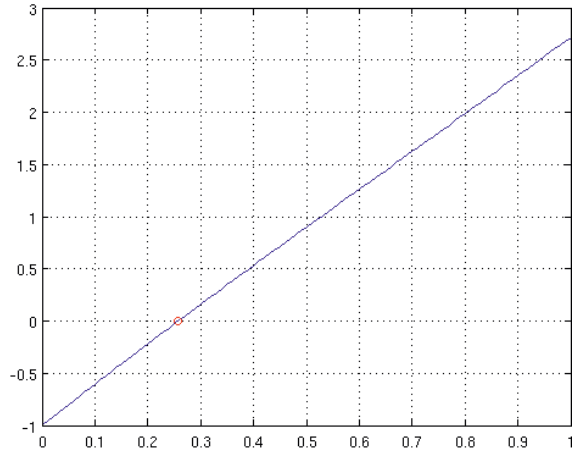


Fig. 4.2
Gràfica de la funció i del zero trobat amb l'ordre fsolve().

Si volem ajustar la crida de la funció `fzero()` a `Matlab` i a `Octave` perquè ens doni el mateix tipus de precisió, podem fer:

```
>> arrel=fzero(@funcio,[0.2,0.3],optimset('TolX',eps))
arrel =  0.257530285439860
```

on l'ordre opcional `optimset('TolX',eps)` fa que la constant `'TolX'` (que mesura l'error en la solució) prengui el valor `eps`. Recordem que `eps` era l'èpsilon de la màquina, i aquest és el valor que pren `Matlab` per defecte quan crida `fzero()`. També podríem posar diferents toleràncies, tot i que toleràncies més petites que `eps` no tenen efecte i convé no posar-les

```
>>  arrel=fzero(@funcio,[0.2,0.3],optimset('TolX',1e-1))
arrel =
   0.300000000000000
>> arrel=fzero(@funcio,[0.2,0.3],optimset('TolX',1e-20))
arrel =
   0.257530285439861
```

És important que l'interval d'aproximació que proporcionem a la funció `fzero()` compleixi que el valor la funció de la qual volem trobar el zero prengui valors amb signe oposat als extrems i, segons el teorema de Bolzano puguem assegurar que aquesta funció té un zero, encara que no sigui únic. Vegem-ho amb els zeros de la funció $f(x) = \sin(x)$ en diferents intervals, segons `Matlab`:

```
>> fzero(@sin,[-pi/2,pi/2])
ans =
     0
>> fzero(@sin,[-pi/2,3*pi/2]) % te dos zeros
```

```
??? Error using ==> fzero at 293
The function values at the interval endpoints must differ in sign.
>> fzero(@sin,[-pi/2,5*pi/2]) % te dos zeros
ans =
     1.972152263052530e-31
>> fzero(@sin,[-pi/2,5*pi/2]) % te tres zeros
ans =
     1.972152263052530e-31
```

La segona crida a `fzero()` dóna problemes perquè la funció pren valors amb el mateix signe als extrems de l'interval, encara que tingui un zero a l'interior. En el segon cas, la funció té tres zeros i el mètode només en troba un. És important, doncs, dibuixar la funció abans de posar-nos a buscar zeros.

4.3. El mètode de la bisecció

El mètode de la bisecció es basa en una aplicació reiterada del teorema de Bolzano per trobar una successió d'intervals encaixats, en els quals la funció contínua de la qual volem cercar el zero pren valors amb signe oposat. A cada pas, dividim la longitud de l'interval per la meitat.

Tant `Matlab` com `Octave` no tenen implementat el mètode de la bisecció. De fet, el mètode de la bisecció té una convergència més lenta que els mètodes implementats com a part de l'ordre `fsolve()`. Amb fins exclusivament didàctics, donem el codi de funció següent, extreta de [16]:

```
function [zero,res,niter]=bisection(fun,a,b,tol,nmax,varargin)
%  BISECTION Find function zeros.
%  ZERO=BISECTION(FUN,A,B,TOL,NMAX) tries to find a zero ZERO
%  of the continuous function FUN in the interval [A,B] using
%  the bisection method. FUN accepts real scalar input x and
%  returns a real scalar value. If the search fails an errore
%  message is displayed. FUN can also be an inline object.
%  ZERO=BISECTION(FUN,A,B,TOL,NMAX,P1,P2,...) passes parameters
%  P1,P2,... to function: FUN(X,P1,P2,...).
%   [ZERO,RES,NITER]= BISECTION(FUN,...) returns the value of
%  the residual in ZERO
%   and the iteration number at which ZERO was computed.
%
%   A.Quarteroni, F.Saleri, Introduction to the Scientific
%  Computing, 2003

x = [a, (a+b)*0.5, b];
fx = feval(fun,x,varargin{:});
if fx(1)*fx(3)>0
    error('Signs at the extrema must be different');
elseif fx(1) == 0
    zero = a; res = 0; niter = 0;
    return
elseif fx(3) == 0
    zero = b; res = 0; niter = 0;
    return
end
```

```
niter = 0;
I = (b - a)*0.5;
while I >= tol & niter <= nmax
   niter = niter + 1;
   if sign(fx(1))*sign(fx(2)) < 0
      x(3) = x(2);
      x(2) = x(1)+(x(3)-x(1))*0.5;
      fx = feval(fun,x,varargin{:});
      I = (x(3)-x(1))*0.5;
   elseif sign(fx(2))*sign(fx(3)) < 0
      x(1) = x(2);
      x(2) = x(1)+(x(3)-x(1))*0.5;
      fx = feval(fun,x,varargin{:});
      I = (x(3)-x(1))*0.5;
   else
       x(2) = x(find(fx==0));
       I = 0;
   end
end
if niter > nmax
    fprintf(['Stopping without converging to the tolerance',...
    'because the maximum number of iterations was reached\n']);
end

zero = x(2);
x = x(2); res = feval(fun,x,varargin{:});
return
```

Per usar-la, podem fer com qualsevol altra funció en `Matlab/Octave`. En primer lloc, editem el fitxer `bisection.m`, on copiem i enganxem la funció (recordeu que hem de ser a l'espai de treball perquè després la puguem usar). Un cop la tenim, ja la podem usar indicant l'interval a on cercarem la solució, $[0.5, 2]$ en el nostre cas, l'error en la solució i el nombre màxim de passos iteratius que usarem. Suposem ara que busquem la solució $x = 1$ de l'equació $x^2(x^2 - 1) = 0$ i que a `funcio.m` hi hem definit la funció:

```
>> bisection(@funcio,0.5,2,1e-10,100) % cridem la funcio bisection.m
ans =  0.999999999970896
```

Aquesta rutina també ens pot informar del valor de $f(x)$ per a la solució aproximada que n'hem obtingut (que esperem que sigui propera a zero) i del nombre de passos que ha usat per calcular-la

```
>> bisection(@funcio,0.5,2,1e-10,100)
ans =  0.999999999970896
>> [ZERO,RES,NITER]= bisection(@funcio,0.5,2,1e-10,100)
ZERO =  0.999999999970896
RES = -5.82076609100793e-11
NITER =  33
>> funcio(ZERO) % comprovem que RES=f(ZERO)<1e-10
ans = -5.82076609100793e-11
```

4.4. Màxims i mínims de funcions

Per acabar aquesta pràctica, veiem un mètode per trobar màxims i mínims de funcions en un interval. Per a funcions derivables, podríem pensar que aquest problema es redueix a trobar zeros de la derivada i que, per tant, això està molt relacionat amb el càlcul de zeros que acabem de descriure. A la pràctica, la majoria de paquets de càlcul numèric, `Matlab` i `Octave` inclosos, usen algoritmes més elaborats.

Com a exemple, trobem ara tots els màxims i mínims de la funció

$$f(x) = \sin(\sin(x) + 1)$$

a l'interval $[-\pi, \pi]$. En primer lloc, definim la funció

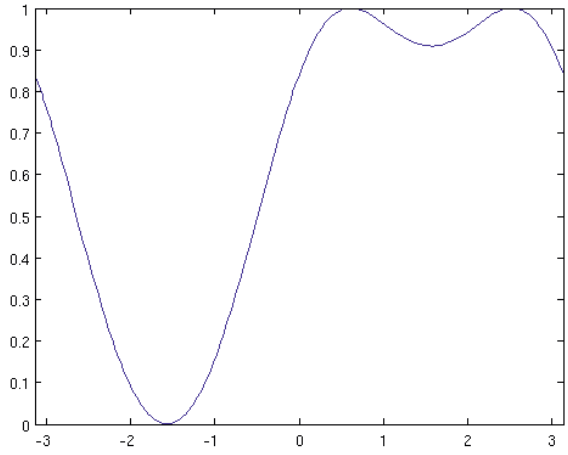
```
function y=funcio(x)
    y=sin(sin(x)+1);
```

i la representem gràficament per veure-la a l'interval $[-\pi, \pi]$

```
>> fplot(@funcio,[-pi,pi])
>> hold on % anirem afegint punts a la grafica
```

i, com observem a la figura 4.3, la funció té un mínim absolut proper a -1.5, un mínim relatiu proper a 1.5 i dos màxims relatius on la funció sembla que pren el valor màxim que és 1 (clarament, la funció f pren només valors entre 0 i 1).

Fig. 4.3
Gràfica de la funció $f(x)=\sin(\sin(x)+1)$ a l'interval $[-\pi,\pi]$.



La funció que usarem per al càlcul de màxims i mínims de funcions d'una variable és l'ordre `fminbnd()`, que "minimitza" una funció en un interval, això és, en cerca el mínim global. La seva sintaxi és

```
xmin= fminbnd(@funcio,a,b)
```

on `funcio` és la funció que volem minimitzar i `a` i `b` són els extrems de l'interval a on volem minimitzar. Aquesta ordre ens retorna un valor `xmin` que conté l'aproximació del mínim absolut de la funció en l'interval. Si volem que, a més, ens retorni el valor mínim de la funció en l'interval (el que seria `funcio(xminim)` podem cridar-la

```
[xmin,fmin]= fminbnd(@funcio,a,b)
```

Així, si l'apliquem a la funció anterior, tindrem, en `Matlab`,

```
>> [xmin,fmin]= fminbnd(@funcio,-pi,pi)
xminim =
  -1.570798671271796
fminim =
     2.748246075157113e-12
>> plot(xmin,fmin,'ro')  % afegim el punt al dibuix
```

que és clarament el mínim global de la funció a l'interval. Si canviem l'interval per poder trobar el mínim relatiu proper a `1.5`, farem

```
>> [xminrel,fminrel]= fminbnd(@funcio,1,2)
xminrel =
   1.570796575954872
fminrel =
   0.909297426825695
>> plot(xminrel,fminrel,'go') % l'afegim al dibuix
```

De nou, les opcions que usen `Matlab` i `Octave` són diferents i això fa que els resultats puguin variar lleugerament. Com passava amb la funció `fzero()`, podem indicar explícitament el paràmetre `'TolX'` per obtenir resultats més semblants. Així, en `Octave` tenim

```
>> [xmin,fmin]= fminbnd(@funcio,-pi,pi) % TolX per defecte es 1e-8
xmin = -1.57079632014132
fmin = 0
>> [xmin,fmin]= fminbnd(@funcio,-pi,pi,optimset('TolX',1e-4))
xmin = -1.57079867127180
fmin =  2.74824607515711e-12
```

Veiem, doncs, que amb l'ordre `fminbnd()` podem anar trobant tots els mínims de la funció. Com ens ho fem per calcular els màxims? Cal que observem que els màxims d'una funció es troben a les abscisses on la funció canviada de signe pren valor mínim. És per això que editem una funció que contingui el negatiu de la funció original, per exemple, `funcionegativa.m`:

```
function y=funcionegativa(x)
    y=-funcio(x);
```

i en calculem els mínims

```
>> edit 'funcionegativa.m'
>> [xmax,fmaxneg]=fminbnd(@funcionegativa,-pi,pi)
xmax =
   0.607470316111433
fmaxneg =
  -0.999999999991398
>> plot(xmax,-fmaxneg,'r*') % canviem el signe al maxim
>> [xmax2,fmaxneg2]=fminbnd(@funcionegativa,2,3)
xmax2 =
   2.534132967939402
fmaxneg2 =
  -0.999999999917098
>> plot(xmax2,-fmaxneg2,'g*')
>> legend('Funcio','Minim global','Minim relatiu',...
 'Maxim absolut 1','Maxim absolut 2','Location','SouthEast')
```

on la darrera ordre és perquè ens mostri la llegenda del gràfic com a la figura 4.4.

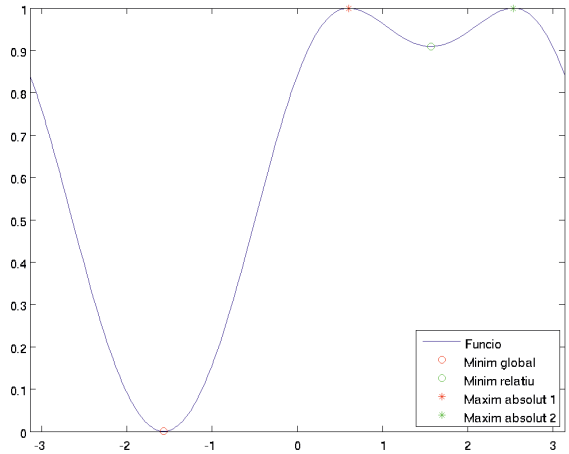


Fig. 4.4 Gràfica de la funció f amb els màxims i mínims relatius i absoluts a l'interval $[-\pi,\pi]$.

Observem que, en aquest cas, la funció és derivable i la seva derivada és

$$f'(x) = \cos(\sin(x)+1)\cos(x)$$

amb la qual cosa els zeros de la derivada que (llevat dels extrems de l'interval) són els candidats a extrems són $\pi/2$, $3\pi/2$ i els valors de $x \in [-\pi,\pi]$ que compleixen

$$\sin(x)+1 = \pi/2,$$

que són $\arcsin(\pi/2-1)$ i $\pi - \arcsin(\pi/2-1)$.

Finalment, observem que l'ordre de minimització `fminbnd()` no necessita que la funció sigui derivable. Així, per exemple, si volem trobar els extrems de la funció $g(x) = |x|$ a l'interval $[-1,1]$, editem la funció `funcio2.m` i hi escrivim

```
function y= funcio2(x)
    y=abs(x);
```

i fent

```
>> fminbnd(@funcio2,-1,1)
ans =
     2.775557561562891e-17
```

observem com s'aproxima molt al valor de zero, que és el mínim absolut de la funció g.

4.5. Exercicis

4.1. Donada la funció

$$f(x) = -x^2 + e^x \cos(x)$$

trobeu totes les arrels a l'interval $(-2,2)$. Comproveu que f en aquestes arrels és petita.

Resposta: -0.646595790718780 i 1.14160179255309.

4.2. Tres amics fan un sopar en què el plat estrella és una truita de patates que han fet i que volen dividir en tres parts d'àrea igual. En comptes de dividir-la de manera radial (en tres sectors de 120°), decideixen fer-ho amb dos talls de ganivet paral·lels a un diàmetre. Suposant que la truita és perfectament circular i de radi 1 (el normalitzem) i, després de pensar-ho una mica, veuen que n'hi ha prou a trobar un valor $x \in (0,1)$ que sigui solució de l'equació

$$\arcsin x + x\sqrt{1-x^2} = \frac{\pi}{4}$$

Quin és aquest valor?

Resposta: 0.403972753299517.

4.3. Trobeu tots els màxims locals de la funció

$$f(x) = \sin(\sin(\sin(x)+1)+1)$$

a l'interval $(-\pi, \pi)$? No compteu els extrems de l'interval.

4.4. Trobeu el zero de la funció següent:

$$f(x) = x^3 + \cos(x) - 2,$$

i comproveu que la resposta x_0 que indiqueu té 6 decimals correctes.

Resposta: $1.172577\ldots$

4.5. Trobeu un zero de la funció següent:

$$f(x) = x - \cos(x^2)$$

i comproveu que la resposta x_0 que indiqueu compleix que $|f(x_0)| < 10^{-8}$.

Resposta: $0.8010707652\ldots$

4.6. Esquema de la pràctica

- Per trobar els zeros d'una funció en `Matlab/Octave`, primer definim la funció en un fitxer `.m`: per exemple, per trobar el zero de la funció $f(x) = e^x + 3x - x^2 - 2$ a l'interval $[0,1]$ escrivim al fitxer `'funcio.m'`

```
function y=funcio(x)
y=exp(x) +3.*x -x.^2 -2;
```

- Per trobar un interval a on la funció tingui un zero, representem la funció a l'interval

```
x=0:0.01:1; y=funcio(x);
plot(x,y); hold on
```

  i comprovem que la funció pren valors amb signe oposat als extrems d'aquest interval.

- Un cop hem identificat un interval a on hi ha l'arrel que busquem (i que la funció prengui valors en signe contrari als extrems de l'interval), per exemple `aproximacio=[0.2,0.3]`, usem l'ordre `fzero(@funcio,aproximacio)`

```
arrel=fzero(@funcio,aprox)
```

  que ens dóna l'aproximació de l'arrel Si, a més, volem conèixer el valor de la funció en aquesta arrel (que ha de ser proper a zero), fen

```
[arrel,farrel]=fzero(@funcio,aprox)
```

- Si volem cridar la funció `fzero()` de manera que doni resultats anàlegs en `Matlab` i en `Octave`, les instruccions són

```
arrel=fzero(@funcio,[0.2,0.3],optimset('TolX',eps))
[arrel,farrel]=fzero(@funcio,[0.2,0.3],optimset('TolX',eps))
```

- Per comprovar que una aproximació, per exemple `0.257530285439861`, té 5 decimals correctes, comprovarem que la funció pren valors amb signes oposats a `0.25753` i a `0.25754`.
- Recordem que, si la funció és un polinomi, és millor usar l'ordre `roots()`. Per exemple, per trobar totes les arrels de $p(x) = x^2(x^2-1) = x^4 - x^2$, faríem

```
roots([1,0,-1,0,0])
```

- Per trobar mínims d'una funció en un interval, usarem l'ordre `fminbnd()`

```
[xmin,fmin]=fminbnd(@funcio,a,b)
```

que troba una aproximació `xmin` del valor de *x* on la funció `funcio` pren el valor mínim a l'interval `[a,b]`, que serà `fmin`. Si volem trobar el màxim de la funció a l'interval farem

```
[xmax,menysfmax]=fminbnd(@funcionegativa,a,b)
```

on ara `funcionegativa.m` conté la funció canviada de signe de `funcio.m`,

```
function y=funcionegativa(x)
    y=-funcio(x);
```

i `xmax` serà el màxim global de la funció a l'interval a on aquesta prendrà el valor `fmax=-menysfmax`.

- Si volem especificar la tolerància de `fminbnd()`, fem

```
fminbnd(@funcio,a,b,optimset('TolX',1e-4)) % TolX 1e-4 (Matlab)
fminbnd(@funcio,a,b,optimset('TolX',1e-8)) % TolX 1e-8 (Octave)
```

Creació i manipulació de matrius

El llenguatge `Matlab/Octave` és especialment indicat per treballar amb matrius i vectors. De fet, el nom del `Matlab` prové de *Matrix Laboratory*. En aquesta pràctica, veurem com definir matrius de forma directa, introduïnt els elements un a un, o bé usant ordres específiques que generaran matrius amb una estructura determinada.

5.1. Matrius en `Matlab/Octave`

A la pràctica 2, hem vist que per definir un vector, havíem d'indicar-lo entre claudàtors `[]` i separant-ne els elements per mitjà de comes o espais. Els vectors indicats d'aquesta manera són els vectors files. A `Matlab/Octave`, podem indicar els vectors columnes separant-los per punt i coma, en comptes d'espai o coma. Observem el següent:

```
>> v = [1, 2, 3] % vector fila
v =
   1   2   3
>> v = [1; 2; 3] % vector columna
v =
   1
   2
   3
```

que ens defineix, respectivament, `v` com un vector fila i després com un vector columna. En general, la manera de definir matrius és "enganxant" vectors columnes l'un al costat de l'altre. Així, per exemple, podem enganxar tres còpies del vector `v`, l'una al costat de l'altra (com si fossin elements d'un vector fila), per crear la següent matriu 3×3

```
A = [v, v, v]
A =
   1   1   1
   2   2   2
   3   3   3
```

Una altra manera d'obtenir el mateix resultat és definir les files prèviament i "enganxar-les" verticalment com si fossin elements d'un vector columna:

```
>> u=[1,1,1];
>> A=[u;2*u;3*u]
A =
   1   1   1
   2   2   2
   3   3   3
```

Aquest procediment d'enganxar també el podem aplicar amb matrius i vectors, cosa que és molt pràctica quan hem d'augmentar matrius. Així, per afegir una nova fila a sota, fem:

```
>> [A;4*u]
ans =
   1   1   1
   2   2   2
   3   3   3
   4   4   4
```

Finalment, també podem definir la matriu directament escrivint tots els elements un a un

```
>> A=[1,1,1;2,2,2;3,3,3]
A =
   1   1   1
   2   2   2
   3   3   3
```

Evidentment, totes les files d'una matriu han de tenir la mateixa dimensió. Altrament, ens trobem amb errors, tant en `Matlab`

```
>> A=[1,1,1;2,2,2;3,3,3] % error de dimensions
??? Error using ==> vertcat
CAT arguments dimensions are not consistent.
```

com en `Octave`

```
>> A=[1,1,1;2,2,2;3,3,3] % error de dimensions
error: number of columns must match (2 != 3)
error: evaluating assignment expression near line 18, column 2
```

Per accedir als elements de matrius, ho fem de manera similar als vectors, és a dir, mitjançant parèntesis rodons, `(,)` indicant la fila i la columna de l'element que volem obtenir. Per exemple,

```
>> A=[1,2;3,4]
A =
   1   2
   3   4
>> A(1,2)
ans =  2
```

obtenim l'element a la primera fila i a la segona columna de la matriu `A`. Això ho podem usar per assignar nous valors a elements d'una matriu:

```
>> A(1,2)=2010 % definim l'element (1,2)
A =
       1    2010
       3       4
```

També podem accedir a columnes i files d'una matriu usant rangs de vectors com els que vàrem aprendre a la pràctica 3. La idea general és que, si fem `A(F,C)` on `F` i `C` són rangs de valors (o, més generalment, subconjunts d'índexs) obtindrem la submatriu formada a partir de `A` i que té com a elements els `A(i,j)`, de manera que $(i, j) \in F \times C$. Ho podem veure millor a través d'un exemple:

```
>> A=[1,2,3;4,5,6;7,8,9]
A =

   1   2   3
   4   5   6
   7   8   9
>> A(1,[1,3]) % fila 1 i columnes 1 i 3 de A
ans =
   1   3
>> A([1,2],[1,3]) % files 1 i 2 i columnes 1 i 3
ans =
   1   3
   4   6
```

També podem usar els rangs de valors de la forma `inici:final`, `inici:pas:final` o, fins i tot, el símbol `:` que representa tots els possibles valors d'una fila o una columna:

```
>> A(1:2,1:2:3) % el mateix que A([1,2],[1,3]). Per què?
ans =
   1   3
   4   6
>> A(1:2,1:3) % el mateix que A([1,2],[1,2,3]). Per què?
ans =
   1   2   3
   4   5   6
>> A(1:2,:) % el símbol : vol dir tots els valors de la columna
ans =
   1   2   3
   4   5   6
>> A(:,:) % tots els elements de la matriu. Tambe  A(1:end,1:end)
ans =
   1   2   3
   4   5   6
   7   8   9
```

Aquestes maneres d'accedir a subelements d'una matriu també les podem usar per redefinir elements d'una matriu. Per exemple, si volem canviar la primera fila de la matriu `A` fem

```
>> A(1,:)=[3,2,1] % definim tota la fila 1
A =
   3   2   1
   4   5   6
   7   8   9
```

si volem copiar la primera columna a la tercera (atenció, perdrem el valor de la tercera),

```
>> A(:,3)=A(:,1) % copiem la columna 1 a la 3
A =
   3   2   3
   4   5   4
   7   8   7
```

o, fins i tot, si volem reassignar una part de la matriu a una altra matriu

```
>> A(1:2,1:2)=[0,0;0,0] % fem zero una submatriu de A
A =
   0   0   3
   0   0   4
   7   8   7
```

i afegir-li la primera fila com a nova columna a la dreta:

```
>> [A,A(:,1)] % afegim a la dreta de A la seva columna 1
ans =
   0   0   3   0
   0   0   4   0
   7   8   7   7
```

5.2. Operacions amb matrius i vectors

5.2.1 Suma de matrius i producte per escalars

Abans de veure com es creen matrius amb formes especials, repassem les operacions bàsiques que es poden fer amb vectors i matrius. En primer lloc, podem usar les operacions d'espai vectorial, és a dir, podem sumar i multiplicar matrius per escalars

```
>> A = [1,2,3;4,5,6;7,8,9]
A =
   1   2   3
   4   5   6
   7   8   9
>> 2*A % multipliquem A per 2
ans =
    2    4    6
    8   10   12
   14   16   18
>> A-2*A % operacions matricials
```

```
ans =
  -1  -2  -3
  -4  -5  -6
  -7  -8  -9
```

i en la suma cal que les dues matrius siguin de la mateixa dimensió; altrament, obtindrem un error

```
>> A+A(1:2,1:2) % error de dimensions
??? Error using ==> plus
Matrix dimensions must agree.
```

on se'ns adverteix que les dimensions han de ser iguals.

5.2.2 Funcions aplicades element a element

És possible també aplicar una funció a una matriu, de la mateixa manera (i amb les mateixes precaucions) com podem fer per a vectors:

```
>> cos(pi*A) % element a element
ans =
  -1   1  -1
   1  -1   1
  -1   1  -1
>> A+1 % sumem 1 a tots els elements de A
ans =
    2    3    4
    5    6    7
    8    9   10
```

El resultat de la primera instrucció és clar. Prenem cadascun dels elements de la matriu `A`, els multipliquem per π i en calculem el cosinus. Aquest és un exemple d'una operació element a element. En la segona operació, el resultat de `A+1` és sumar 1 a tots els elements de la matriu `A`.

5.2.3 Operacions matricials

Potències de matrius: `^`, `mpower()` Aquest comportament diferent d'operacions element a element i operacions matricials el podem veure clarament amb l'exemple següent. La matriu

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

compleix que $A^3 = A \cdot A \cdot A$ és la matriu zero, és a dir, és *nilpotent*. Per comprovar aquesta operació en `Matlab/Octave`, fem

```
>> A=[0,1,1;0,0,1;0,0,0]
A =
   0   1   1
   0   0   1
   0   0   0
```

```
>> A^3 % nomes valid per a matrius quadrades
ans =
   0   0   0
   0   0   0
   0   0   0
```

mentre que si el que volíem fer és el producte element a element faríem

```
>> A.^3 % elevem a 3 element a element
ans =

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

que ens donarà el producte element a element. Per tal de distingir millor entre les dues operacions, podem usar l'ordre `mpower(A,3)` (de *matrix power*) que és el mateix que fer `A^3` però sense el perill de confondre'ns amb l'operació element a element.

En el cas de les operacions element a element, a part d'elevar els elements d'una matriu a un escalar, també podem fer `A.^B`, on `A` i `B` tenen les mateixes dimensions, i el que obtindrem serà una nova matriu amb elements $a_{ij}^{b_{ij}}$.

Producte de matrius: `*`, `mtimes()` Amb el producte entre matrius s'esdevé de manera similar,

```
>> B=2*A % multipliquem per 2 la matriu
B =
   0   2   2
   0   0   2
   0   0   0
>> A*B % producte matricial
ans =
   0   0   2
   0   0   0
   0   0   0
>> A.*B % producte element a element
ans =
   0   2   2
   0   0   2
   0   0   0
```

és a dir, quan `A` i `B` són matrius (amb dimensions compatibles per a la multiplicació), `A*B` indica el producte matricial, mentre que `A.*B` indica el producte element a element. Si volem no enredar-nos, podem usar la funció `mtimes(A,B)`, que ens donarà el producte matricial (és a dir `A*B`).

A `Matlab/Octave`, el producte matriu per vector és un cas particular del producte matricial. Això vol dir que per calcular $A \cdot v$, on A és una matriu amb k files i l columnes, v ha de ser un vector columna de dimensió l. Per exemple,

```
>> A=[0,1,1;0,0,1] % matriu
A =
   0   1   1
   0   0   1
>> v=[1;1;1] % vector columna
v =
   1
   1
   1
>> A*v % matriu per vector columna
ans =
   2
   1
```

i, en canvi, si fem el mateix però amb `v` vector fila, ens donarà un error

```
>> v=[1,1,1]; A*v
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

Per passar de vectors fila a vectors columna, podem usar l'operació transposar, indicada per `transpose(v)` o simplement `v'`, que ens crea una nova matriu amb files i columnes intercanviades, de manera que ara sí que podem escriure

```
>> v=[1,1,1]; A*v' % transposem el vector fila i passa a columna
ans =
   2
   1
```

Determinant i inversa d'una matriu: `det()`, `inv()` Donada una matriu quadrada A, la seva inversa és una altra matriu quadrada, que escrivim com A^{-1} i que compleix que $A \cdot A^{-1} = A^{-1}A = I$, on I és la matriu identitat de la mateixa dimensió. Per poder invertir una matriu cal que el seu determinant sigui diferent de zero. A `Matlab/Octave`, el determinant d'una matriu quadrada `A` es calcula com `det(A)` i la inversa, si n'hi ha, a través de `inv(A)`, `inverse(A)` o bé `A^(-1)` (encara que amb aquesta darrera expressió correm el risc de confondre-la amb l'operació element a element, `A.^(-1)`)

```
>> A=[1,-1;1,1]
A =
   1  -1
   1   1
>> det(A) % determinant de la matriu quadrada A
ans =  2
>> B=inv(A) % inversa de la matriu quadrada A
B =
   0.500000000000000   0.500000000000000
  -0.500000000000000   0.500000000000000
>> B*A % producte matricial
ans =
   1   0
   0   1
```

Divisió de matrius: `/,mldiv()` A `Matlab/Octave`, s'entén que la divisió de dues matrius significa multiplicar una per la inversa de l'altra. Com que multiplicar per la dreta o multiplicar per l'esquerra no dóna el mateix en el cas de matrius, cal que expressem, per una banda, la divisió per la dreta i, per l'altra, la divisió per l'esquerra.

- **Divisió per la dreta** $A \cdot B^{-1}$**:** S'indica com `A/B` o bé com `mrdivide(A,B)` (de *matrix right divide*) i cal que B no sigui singular:

```
>> v=[1,1,1]; A=[v;v;v] % A es singular
A =
   1   1   1
   1   1   1
   1   1   1
>> B=[1,1,1;0,1,1;0,0,1] % B no es singular
B =
   1   1   1
   0   1   1
   0   0   1
>> A/B % el mateix que A*inverse(B)
ans =
   1   0   0
   1   0   0
   1   0   0
>> mrdivide(A,B) % el mateix que A/B
ans =
   1   0   0
   1   0   0
   1   0   0
>> A*inverse(B) % el mateix que A/B
ans =
   1   0   0
   1   0   0
   1   0   0
```

- **Divisió per l'esquerra** $B^{-1} \cdot A$**:** S'indica com `B\A` (atenció amb l'ordre) o bé com `mldivide(B,A)` (de *matrix left divide*) i cal que B no sigui singular:

```
>> B\A
ans =
   0   0   0
   0   0   0
   1   1   1
>> mldivide(B,A)
ans =
   0   0   0
   0   0   0
   1   1   1
>> inv(B)*A
ans =
   0   0   0
   0   0   0
   1   1   1
```

Quan les matrius no són invertibles, surt un avís.

5.3. Creació de matrius especials

Sovint, hem d'escriure matrius de dimensions molt grans i no és pràctic haver d'introduir el valor dels elements un a un (ni tan sols a través d'iteradors com els de la pràctica 9). Per facilitar aquesta feina, el llenguatge `Matlab/Octave` posa a la nostra disposició tot un conjunt de rutines per dissenyar matrius de forma especial.

5.3.1 Matrius de zeros: `zeros()`

Aquesta instrucció crea una matriu quadrada de zeros de dimensió `k`, quan es crida amb un sol argument `zeros(k)`, i una matriu de `k` files i `l` columnes, quan es crida amb dos arguments, `zeros(k,l)`:

```
>> zeros(2) % matriu quadrada de zeros 2x2
ans =
   0   0
   0   0
>> zeros(2,3) % matriu de zeros 2x3
ans =
   0   0   0
   0   0   0
>> zeros(1,3) % vector fila de zeros de dim. 3
ans =
   0   0   0
```

5.3.2 Matrius d'uns: `ones()`

Fa el mateix que la funció `zeros()` però omplint la matriu amb 1 en comptes de 0.

```
>> zeros(2,3)+1 % el mateix que ones(2,3)
ans =
   1   1   1
   1   1   1
>> ones(2,3) % matriu d'uns de 2x3
ans =
   1   1   1
   1   1   1
```

5.3.3 Matriu identitat: `eye()`

Funciona anàlogament a la funció `zeros()` però omple els elements de la diagonal principal amb el valor `1` i la resta els deixa a 0. Per tant, quan `eye(k)` rep un únic argument, retorna la matriu quadrada identitat:

```
>> eye(2) % identitat 2x2
ans =
   1   0
   0   1
```

i, quan rep dos arguments, `eye(n,m)` funciona com `zero(n,m)` però posant uns a la diagonal principal:

```
>> A=eye(2,4) % A(1:2,1:2)=identitat. La resta zeros
A =
     1     0     0     0
     0     1     0     0
```

5.3.4 Matrius diagonals: `diag()`

Aquesta funció es pot cridar amb un o dos arguments. Si es crida amb un únic argument, `diag(v)`, on `v` és un vector (fila o columna) de dimensió `k`, obtindrem una matriu quadrada, de dimensió `k`, que té tots els elements igual a 0, llevat de la diagonal, on té els valors donats pel vector `v`

```
>> diag([1,1,1]) % el mateix que eye(3)
ans =
   1   0   0
   0   1   0
   0   0   1
```

Quan cridem aquesta funció amb dos arguments, `diag(v,l)`, on `v` és un vector (fila o columna) de dimensió `k` i `l` és un nombre enter, crea una matriu quadrada amb `k+l` files i `k+l` columnes que conté `v` a la diagonal `l`-èsima (comptant que la diagonal principal és quan `l=0`), i la resta d'elements igual a 0. Ho podem veure millor amb un exemple. Per generar una matriu 4×4 que tingui la diagonal per sobre de la diagonal principal igual a $(1,2,3)$, fem

```
>> diag([1,2,3],1) % [1,2,3] a la diagonal +1
ans =
   0   1   0   0
   0   0   2   0
   0   0   0   3
   0   0   0   0
>> diag([1,2,3],-2) % [1,2,3] a la diagonal -2
ans =
   0   0   0   0   0
   0   0   0   0   0
   1   0   0   0   0
   0   2   0   0   0
   0   0   3   0   0
```

Com a aplicació, trobem quant val el determinant de la matriu quadrada següent de dimensió 10:

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Per no haver d'escriure els cent elements un a un, observem que, llevat dels elements $A_{1,10}$ i $A_{10,1}$, que valen 1, i de les 3 diagonals centrals, que també valen 1, tota la resta són 0. Anem-ho a fer pas a pas:

```
>> N=10
N =  10
>> A=diag(ones(N,1)) % diagonal principal
A =
    1    0    0    0    0    0    0    0    0    0
    0    1    0    0    0    0    0    0    0    0
    0    0    1    0    0    0    0    0    0    0
    0    0    0    1    0    0    0    0    0    0
    0    0    0    0    1    0    0    0    0    0
    0    0    0    0    0    1    0    0    0    0
    0    0    0    0    0    0    1    0    0    0
    0    0    0    0    0    0    0    1    0    0
    0    0    0    0    0    0    0    0    1    0
    0    0    0    0    0    0    0    0    0    1
>> A=A+diag(ones(N-1,1),1) % diagonal per sobre
A =
    1    1    0    0    0    0    0    0    0    0
    0    1    1    0    0    0    0    0    0    0
    0    0    1    1    0    0    0    0    0    0
    0    0    0    1    1    0    0    0    0    0
    0    0    0    0    1    1    0    0    0    0
    0    0    0    0    0    1    1    0    0    0
    0    0    0    0    0    0    1    1    0    0
    0    0    0    0    0    0    0    1    1    0
    0    0    0    0    0    0    0    0    1    1
    0    0    0    0    0    0    0    0    0    1
>> A=A+diag(ones(N-1,1),-1) % diagonal per sota
A =
    1    1    0    0    0    0    0    0    0    0
    1    1    1    0    0    0    0    0    0    0
    0    1    1    1    0    0    0    0    0    0
    0    0    1    1    1    0    0    0    0    0
    0    0    0    1    1    1    0    0    0    0
    0    0    0    0    1    1    1    0    0    0
    0    0    0    0    0    1    1    1    0    0
    0    0    0    0    0    0    1    1    1    0
    0    0    0    0    0    0    0    1    1    1
    0    0    0    0    0    0    0    0    1    1
>> A(N,1)=-1; A(1,N)=-1 % el que manca
A =
    1    1    0    0    0    0    0    0    0   -1
    1    1    1    0    0    0    0    0    0    0
    0    1    1    1    0    0    0    0    0    0
    0    0    1    1    1    0    0    0    0    0
    0    0    0    1    1    1    0    0    0    0
    0    0    0    0    1    1    1    0    0    0
    0    0    0    0    0    1    1    1    0    0
    0    0    0    0    0    0    1    1    1    0
    0    0    0    0    0    0    0    1    1    1
   -1    0    0    0    0    0    0    0    1    1
>> det(A)
ans =  1
```

5.3.5 Matrius aleatòries: `rand()`

El seu funcionament és anàleg a la funció `zeros()` però, en comptes d'omplir la matriu amb zeros, l'omple amb nombres aleatoris uniformement distribuïts a l'interval $[0, 1]$. Cada cop que cridem aquesta instrucció es crea una matriu diferent:

```
>> rand(2) % matriu aleatoria 2x2
ans =
   0.728189496120894   0.360911792585134
   0.785783767171848   0.381327499524045
>> rand(2) % matriu aleatoria 2x2
ans =
   0.298292933495621   0.784069794760585
   0.158944222497769   0.634316703362842
```

5.3.6 Canvi de dimensions de matrius: `reshape()`

Aquesta funció és útil per canviar la forma de matrius. Donada una matriu `A` de `k` files i `l` columnes, la funció `reshape(A,m,n)`, on `m` i `l` són dos nombres naturals que compleixen que `k*l`, és igual a `m*n`, retorna una nova matriu de `m` files i `l` columnes formada pels elements de `A` en el mateix ordre i llegits primer per files i després per columnes. Amb un exemple ho veurem millor:

```
>> a=1:12
a =
   1   2   3   4   5   6   7   8   9  10  11  12
>> B=reshape(a,3,4) % a te 12 elements i la passem a 3x4
B =
    1    4    7   10
    2    5    8   11
    3    6    9   12
>> reshape(B,4,3) % B te 12 elements i la passem a 4x3
ans =
    1    5    9
    2    6   10
    3    7   11
    4    8   12
```

mentre que, si no es compleix la condició sobre el nombre de files i columnes, retorna un error.

5.3.7 Altres matrius especials

`Matlab/Octave` té altres rutines que permeten dissenyar formes especials de matrius. A continuació, en posem alguns exemples:

`tril()` i `triu()` Aquestes funcions serveixen per obtenir les matrius triangular superior (`triu()`) i triangular inferior (`tril()`) a partir d'una altra matriu:

```
>> A=reshape(1:9,3,3)
A =
   1   4   7
   2   5   8
   3   6   9
>> tril(A) % posa zeros per sobre la diagonal
ans =
   1   0   0
   2   5   0
   3   6   9
>> triu(A) % posa zeros per sota la diagonal
ans =
   1   4   7
   0   5   8
   0   0   9
```

També es poden cridar amb un enter `triu(A,k)` que indiqui la diagonal a partir de la qual omplim amb zeros, per exemple:

```
>> triu(A,-1) % posa zeros per sota la diagonal-1
ans =
   1   4   7
   2   5   8
   0   6   9
```

`vander()` La matriu de Vandermonde d'un vector $c = (c_1, \ldots, c_n)$ és la matriu quadrada formada per les potències de les components de c

$$V = \begin{pmatrix} c_1^{n-1} & \ldots & c_1^2 & c_1 & 1 \\ c_2^{n-1} & \ldots & c_2^2 & c_2 & 1 \\ c_3^{n-1} & \ldots & c_3^2 & c_3 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_n^{n-1} & \ldots & c_n^2 & c_n & 1 \end{pmatrix}$$

```
>> A=vander(1:4) % matriu de Vandermonde de 1,2,3,4.
A =
    1    1    1    1
    8    4    2    1
   27    9    3    1
   64   16    4    1
>> det(A)
ans =  12
>> (4-1)*(3-1)*(2-1)*(4-2)*(3-2)*(4-3)
ans =  12
```

on a la darrera instrucció hem comprovat la fórmula que diu que el determinant de la matriu de Vandermonde associat al vector c ve donat pel producte

$$\det(V) = \prod_{1 \le i < j \le n} (c_i - c_j).$$

Sovint el que apareix com a matriu de Vandermonde és la que té les columnes permutades, és a dir:

$$\begin{pmatrix} 1 & c_1 & c_1^2 & \dots & c_1^{n-1} \\ 1 & c_2 & c_2^2 & \dots & c_2^{n-1} \\ 1 & c_3 & c_3^2 & \dots & c_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & c_n & c_n^2 & \dots & c_n^{n-1} \end{pmatrix}$$

Per obtenir-la en `Matlab/Octave`, usarem que `4:-1:1` ens dóna el rang `[4,3,2,1]`:

```
>> A=vander(1:4)
>> A(:,4:-1:1)  % posa les columnes al reves.
ans =
    1    1    1    1
    1    2    4    8
    1    3    9   27
    1    4   16   64
```

Per canviar les files d'ordre, faríem `A(4:-1:1,:)`.

`blkdiag()` Aquesta rutina ens permet crear matrius diagonals per blocs, és a dir, donades n matrius quadrades

$$A = \begin{pmatrix} A_1 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_n \end{pmatrix}$$

i, per tant, és una matriu quadrada amb dimensió la suma de dimensions de cadascuna de les matrius. Comprovem ara que el determinant és el producte de determinants:

```
>> format short % perque es vegi be el resultat
>> A1=rand(1) % donara diferent
A1 =   0.8147
>> A2=rand(2) % donara diferent
A2 =
    0.9058    0.9134
    0.1270    0.6324
>> A3=rand(3) % donara diferent
A3 =
    0.0975    0.9575    0.9706
    0.2785    0.9649    0.9572
    0.5469    0.1576    0.4854
>> A=blkdiag(A1,A2,A3) % matriu de blocs A1,A2 i A3
A =
  0.8147         0         0         0         0         0
       0    0.9058    0.9134         0         0         0
       0    0.1270    0.6324         0         0         0
       0         0         0    0.0975    0.9575    0.9706
       0         0         0    0.2785    0.9649    0.9572
       0         0         0    0.5469    0.1576    0.4854
>> det(A)-det(A1)*det(A2)*det(A3)
ans =
     0
```

`compan()` Donat un vector `v` de longitud n amb `v(1)` diferent de zero, `A=compan(v)` retorna una matriu quadrada de dimensió n que conté `1` a la subdiagonal principal i que, a la primera fila és `-v(2:n)/v(1)`. Per exemple,

```
>> A= compan([2,4,6,8]) % matriu companion de [2,4,6,8]
A  =
  -2  -3  -4
   1   0   0
   0   1   0
```

A la pràctica 10, veurem que aquesta matriu està molt relacionada amb el polinomi que defineix el vector `v`; en aquest cas, $2x^3+4x^2+6x+8$. Ara per ara, ens limitem a observar que si apliquem el polinomi a la matriu (entenent que les potències són operacions matricials i que sumar un número és sumar la identitat multiplicada per aquest número), obtindrem la matriu zero:

```
>> 2*A^3+4*A^2+6*A+8*eye(3)
ans =
   0   0   0
   0   0   0
   0   0   0
```

5.4. Exercicis

5.1. Donada la matriu següent:

$$A=\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10\\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 3 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0\\ 5 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0\\ 6 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0\\ 7 & 0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0\\ 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 & 0\\ 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 0\\ 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 \end{pmatrix}$$

Calculeu el determinant i comproveu que

$$\det(A)=-\left(\sum_{n=1}^{10} n\right)\left(\prod_{n=1}^{10} n\right).$$

5.2. Donada la matriu següent:

$$A=\begin{pmatrix} 17 & 24 & 1 & 8 & 15\\ 23 & 5 & 7 & 14 & 16\\ 4 & 6 & 13 & 20 & 22\\ 10 & 12 & 19 & 21 & 3\\ 11 & 18 & 25 & 2 & 9 \end{pmatrix}$$

que podem trobar fent `magic(5)` en `Matlab/Octave`, quant val el menor corresponent a l'element $(4,3)$?

Resposta: -15600.

5.3. Trobeu el determinant de la matriu 10×10 següent:

$$\begin{pmatrix}
\sqrt{2} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
1 & \sqrt{2} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & \sqrt{2} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & \sqrt{2} & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & \sqrt{2} & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & \sqrt{2} & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & \sqrt{2} & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & \sqrt{2} & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \sqrt{2} & 1 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \sqrt{2}
\end{pmatrix}$$

Resposta: 2.

5.4. Donada la matriu següent:

$$A = \begin{pmatrix}
2 & 1 & 1 & 1 \\
1 & 2 & 1 & 1 \\
1 & 1 & 2 & 1 \\
1 & 1 & 1 & 2
\end{pmatrix}$$

digueu quant val el determinant de $I + A^3$, on I és la identitat.

Resposta: 1008.

5.5. Esquema de la pràctica

- Creació de vectors i matrius
  - `v = [1,2,3]` defineix un vector fila.
  - `v = [1;2;3]` defineix un vector columna.
  - `A=[1,2,3;4,5,6]` defineix la matriu
$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$
- Modificació, ampliació i reducció de matrius:
  - Accedir a elements d'una matriu `A(fila,columna)`. Redefinir l'element `A(fila,columna)=nou_valor`.

- Seleccionar submatrius: `A(vector_de_files,vector_de_columnes)`. Per exemple, `A(1:2,[1,3])`.
- Seleccionar tota una fila o columna amb el símbol `:`. Per exemple, `A(:,columnes)` selecciona totes les files i les columnes seleccionades al vector `columnes`, com ara `A(:,1:2)`, en l'ordre dels vectors.
- Assignació de submatrius `A(vector_de_files,vector_de_columnes) = matriu_de_nous_valors`. Per exemple, $A(:,2) = [1,1,1]$.
- Ampliació de matrius: `[A,B]` a mplia `A` p osant `B` a la dreta (mateix nombre de files!) i `[A;B]` amplia `A` posant `B` a sota (mateix nombre de columnes!).
- Canviar files per columnes (transposar), `transpose(A)` o bé `A'`.

- Operacions sobre matrius:
  - Dimensions d'una matriu `size(A)`.
  - Suma i producte per escalars: `A+B` (dimensions!) i `2*A`.
  - Operacions element a element: `A.*B`, `A./B`, `A.^B`, `cos(A)`, `A+1`,...
  - Operacions matricials:
    - ⋆ Producte matricial ($A \cdot B$): `A*B` o bé `mtimes(A,B)`
    - ⋆ Potències de matrius (A^k) `A^k`, `mpower(A,k)`.
    - ⋆ Determinant i inversa d'una matriu quadrada: `det(A)` i `inv(A)`.
    - ⋆ Divisió matricial per l'esquerra, ($A^{-1} \cdot B$) `A\B= mldivide(A,B)`, o per la dreta, ($B \cdot A^{-1}$) `B/A= mrdivide(B,A)`.
  - Creació de matrius especials:
    - ⋆ Matrius de zeros `zeros(n)`, `zeros(n,m)` i d'uns `ones(n)`, `ones(n,m)`.
    - ⋆ Matriu identitat `eye(n)`.
    - ⋆ Matrius diagonals `diag(v)` amb `v` vector i `diag(v,posicio)` amb `posicio` enter.
    - ⋆ Matrius aleatòries `rand(n)` i `rand(n,m)`.
    - ⋆ Canviar les dimensions: `reshape([1,2,3,4],2,2)=[1,2;3,4]`.
    - ⋆ Fer matrius triangulars inferiors `tril(A)` i superiors `triu(A)`. També `tril(A,posicio)` i `triu(A,posicio)`.
    - ⋆ `vander(v)` matriu de Vandermonde associada a `v`.
    - ⋆ `blkdiag(A1,A2,A3)` matriu de blocs `A1`, `A2` i `A3`.
    - ⋆ Matriu *companion* d'un vector `v`: `compan(v)`.

Sistemes d'equacions lineals

En aquesta pràctica, estudiem com resoldre sistemes d'equacions lineals del tipus

$$Ax = b$$

quan A és una matriu de m files i n columnes, b és un vector de dimensió m, i busquem les solucions x com un vector de dimensió n. Com bé sabem, aquesta solució pot no existir o bé, si existeix, pot no ser única, de manera que hem de veure també com discutir aquestes diferents possibilitats. Estudiem, en particular per a $b = 0$, el conjunt de solucions (l'anomenat nucli de la matriu A), així com la imatge de la matriu A.

6.1. Resolució de sistemes quadrats invertibles

El cas més senzill és quan aquesta matriu és quadrada i invertible (el seu determinant és diferent de zero), la qual cosa implica que el sistema té una única solució. La solució d'aquest sistema ve donada per

$$x = A^{-1}b$$

i, per tant, el podem calcular amb l'ordre \ que ens permet efectuar la divisió matricial. Fem-ho ara amb la matriu de Vandermonde:

```
A=vander(1:5)
A =
     1     1     1     1     1
    16     8     4     2     1
    81    27     9     3     1
   256    64    16     4     1
   625   125    25     5     1
```

i b el vector següent:

```
>> v= cos(1:5) % [cos(1),cos(2),...,cos(5)]
v =
   0.54030  -0.41615  -0.98999  -0.65364   0.28366
```

que hem de passar a vector columna perquè puguem usar l'operador `\` o, equivalentment, la funció `mldivide()`

```
>> b=transpose(v); % igual que b=v'
>> x=A\b % igual que mldivide(A,b)
x =
  -0.034868
   0.436610
  -1.556664
   1.180289
   0.514935
>> A*x-b % comprovem el resultat
ans =
   0.0000e+00
   5.5511e-17
   4.4409e-16
   0.0000e+00
   7.2164e-16
```

El fet d'haver triat la matriu de Vandermonde associada a `1:5` té com a conseqüència que, si fem

```
>> polyval(x,1:5) % avaluem el pol. x en 1:5
ans =
   0.54030   -0.41615   -0.98999   -0.65364    0.28366
```

obtenim el mateix que el vector `v`. Interpretat com a polinomi, doncs, aquest vector representa l'anomenat *polinomi interpolador* que en `1:5` pren els valors prescrits.

Es podria pensar que aquesta expressió és equivalent a la següent:

```
>> B=inv(A); x=B*v' % no es optim
x =
  -0.034868
   0.436610
  -1.556664
   1.180289
   0.514935
```

però això no és gens òptim, ja que primer calcula la inversa de `A`, que pot ser molt costós en temps i poc acurat quan la dimensió de `A` és gran i després ho multiplica per `v'`. Per tant, no hem d'utilitzar la funció `inv()` llevat que necessitem la inversa realment. Si voleu veure un exemple de com es pot alentir un codi amb aquesta manera, podeu mirar d'executar el següent:

```
>> N=2000; A=rand(N,N);v=cos(1:N); % matriu aleatoria de dimensio N
>> t= cputime; x=A\v'; cputime - t % ens dona el temps de calcul
ans =  2.1595
>> t= cputime; B=inv(A); x=B*v'; cputime - t
ans =  6.1599
```

6.1.1 Cas no invertible, primeres impressions

En cas que la matriu sigui quadrada però no invertible, la rutina ens donarà un error, però al mateix temps intentarà donar una possible solució del sistema d'equacions corresponent. Provem-ho amb la matriu següent:

$$A = \begin{pmatrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{pmatrix}$$

que correspon a un quadrat màgic (la suma de totes les files i columnes val el mateix), en aquest cas singular, i que podem obtenir amb `Matlab/Octave`

```
>> A=magic(4) % un quadrat magic 4x4
A =
   16    2    3   13
    5   11   10    8
    9    7    6   12
    4   14   15    1
>> sum(A(1,:)) % suma de la primera fila
ans =  34
>> sum(A(:,3)) % suma de la tercera columna
ans =  34
```

Intentem resoldre ara el sistema d'equacions següent:

$$\begin{pmatrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Vegem què passa si l'intentem resoldre amb `Octave`:

```
>> A=magic(4); b=ones(4,1); x=A\b % error,rang(A)<4
warning: matrix singular to machine precision, rcond = 4.89645e-18
warning: attempting to find minimum norm solution
warning: dgelsd: rank deficient 4x4 matrix, rank = 3
x =
   0.029412
   0.029412
   0.029412
   0.029412
>> A*x % comprovem el resultat
ans =
   1.0000
   1.0000
   1.0000
   1.0000
```

ens surt un avís i un possible resultat i en `Matlab` també:

```
>> A=magic(4); b=ones(4,1); x=A\b % error,rang(A)<4
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.306145e-17.
x =
    0.0588
    0.1176
   -0.0588
         0
>> A*x % comprovem el resultat
ans =
   1
   1
   1
   1
```

Tant en un cas com en l'altre, el sistema ens avisa que el rang de la matriu (el nombre de files o columnes linealment independents) és $3 < 4$ i que, per tant, intentarà trobar una solució particular. Observem que la solució que ha trobat `Octave` és multiplicar b per $1/34$, que és la suma de files i columnes indistintament. Com que la suma de files sempre val 34, el resultat serà el correcte. En el cas de `Matlab` ens dóna un altre valor, en què la tercera component val 0. En tots dos casos podem comprovar que `A*x` dóna efectivament un vector columna d'uns.

Vegem ara com trobar aquest rang i estudiar les solucions de sistemes lineals no necessàriament quadrats. Per fer-ho, introduirem l'anomenada *forma esglaonada reduïda d'una matriu*, que permet conèixer el conjunt de solucions d'un sistema lineal.

6.2. Forma esglaonada reduïda d'una matriu

En `Matlab/Octave`, la tècnica que s'usa per a la discussió de sistemes lineals és una variant del procés d'eliminació de Gauss per tal d'aconseguir, a través de transformacions elementals, passar la matriu a l'anomenada *forma esglaonada reduïda*. Això es fa mitjançant l'ordre `rref()`.

Vegem-ho amb un exemple. La funció `rref()` opcionalment ens dóna un vector que ens informa de les variables que hem usat com a pivot,

```
>> A=magic(4);
>> [R,jb]=rref(A) % R=rref(A) amb els pivots a jb
R =

   1.00000   0.00000   0.00000   1.00000
   0.00000   1.00000   0.00000   3.00000
   0.00000   0.00000   1.00000  -3.00000
   0.00000   0.00000   0.00000   0.00000
jb =
   1   2   3
```

de manera que

- `r = length(jb)` dóna el rang de `A`.

- `jb` conté els índexs de les variables que usem com a pivot en resoldre $Ax = b$. Les columnes corresponents són les columnes pivot.
- `R(1:r,jb)` és la identitat.

Observem que, com que en el nostre cas `jb=[1,2,3]`, això significa que no hem pivotat en cap moment. Per exemple, si fem

```
>> A=magic(4); A(:,1)=zeros(4,1)
A =

     0     2     3    13
     0    11    10     8
     0     7     6    12
     0    14    15     1
>> [R,jb]=rref(A)
R =
    0    1    0    0
    0    0    1    0
    0    0    0    1
    0    0    0    0
jb =
    2    3    4
```

significa que hem usat les variables x_2, x_3 i x_4 com a pivots (de fet, la variable x_1 no apareixia a les equacions) i, per tant,

```
>> r = length(jb) % rang 3
r =  3
>> R(1:r,jb) % la identitat 3x3
ans =
    1    0    0
    0    1    0
    0    0    1
```

Per entendre millor el procediment de `rref()` calculem la forma esglaonada reduïda d'una matriu A sense usar aquesta ordre i després ho compararem amb el resultat que ens ofereixen `Matlab/Octave`:

```
>> A=[1,0,1,0,3;1,0,3,6,7;2,0,4,6,10]
A = 1  0 1  0  3
     1  0 3  6  7
     2  0 4  6  10
     >> A(2,:)=A(2,:)-A(1,:) % restem la fila 1 a la 2
     A =
     1     0     1     0     3
0     0     2     6     4
2     0     4     6    10
>> A(3,:)=A(3,:)-2*A(1,:) % restem 2*fila1 a fila3
A =
1    0    1    0    3
0    0    2    6    4
0    0    2    6    4
```

ara ja hem obtingut zeros a la primera columna llevat del primer element. Com que la segona columna és tota plena de zeros, hem d'usar com a pivot la tercera columna, a on hem d'aconseguir un zero a la posició `A(2,3)`:

```
>> A(3,:)=A(3,:)-A(2,:) % restem fila2 a fila3
A =
1    0    1    0    3
0    0    2    6    4
0    0    0    0    0
>> A(2,:)=A(2,:)/2 % dividim la fila2 per 2
A =
1    0    1    0    3
0    0    1    3    2
0    0    0    0    0
>> A(1,:)=A(1,:)-A(2,:) % restem fila2 a fila1
A =
1    0    0   -3    1
0    0    1    3    2
0    0    0    0    0
```

que és, finalment, la forma esglaonada reduïda de la matriu original `A`. El vector de pivots, que són les columnes respecte de les quals hem pivotat seran, doncs, `[1,3]`. Vegem ara com l'ordre `rref()` ens dóna aquest mateix resultat:

```
>> A=[1,0,1,0,3;1,0,3,6,7;2,0,4,6,10]; % cal introduir-la de nou
>> [R,jb]=rref(A)
R =
1    0    0   -3    1
0    0    1    3    2
0    0    0    0    0
jb =
1    3
```

6.3. Imatge i nucli d'una matriu

6.3.1 Càlcul de la imatge

Acabem de veure que, per al càlcul del rang d'una matriu, n'hi ha prou a comptar el nombre de columnes pivot en calcular la forma esglaonada reduïda d'una matriu. Si ho volem fer directament, podem usar la funció `rank(A)`

```
>> A=magic(4)
A =
   16     2     3    13
    5    11    10     8
    9     7     6    12
    4    14    15     1
>> rank(A) % rang de la matriu A
ans =  3
>> [R,jb]=rref(A); length(jb) % donara tambe el rang
ans =  3
```

i, per tant, hi ha tres files (o tres columnes) linealment independents. Recordem que `jb` ens dóna els índexs de columnes independents i, per tant,

```
>> rank(A(:,jb))
ans =  3
```

Igualment, com que la imatge de A és el subespai generat per les columnes, les columnes pivot en formen una base i, per tant,

- `A(:,jb)` és una base de la imatge de `A`.

```
>> A(:,jb) % base de la imatge de A
ans =
    2    3   13
   11   10    8
    7    6   12
   14   15    1
```

6.3.2 Càlcul del nucli

El càlcul d'una expressió parametritzada del nucli d'una matriu també es pot fer a partir de la forma esglaonada reduïda, usant que la matriu `A` és similar a la matriu `R` i que `R(1:r,jb)` és la identitat. En efecte, recordem que el nucli d'una matriu és l'espai lineal de vectors $x \in \mathbb{R}^n$, de manera que $Ax = 0$. Si ens fixem en la forma esglaonada reduïda, que havíem calculat abans,

```
>> A=[1,0,1,0,3;1,0,3,6,7;2,0,4,6,10];
>> [R,jb]=rref(A)
R =
   1   0   0  -3   1
   0   0   1   3   2
   0   0   0   0   0
jb =
   1   3
```

veiem que el sistema d'equacions $Ax = 0$ és equivalent a $Rx = 0$, d'on l'espai nul ve donat per les equacions

$$\begin{aligned} x_1 - 3x_4 + x_5 &= 0, \\ x_3 + 3x_4 + 2x_5 &= 0 \end{aligned}$$

que ens deixen expressar la solució en forma parametritzada en termes de les variables independents x_2, x_4 i x_5 (les que no són pivot):

$$\text{Nucli}\,(A) = \{(3x_4 - x_5, x_2, -3x_4 - 2x_5, x_4, x_5) \in \mathbb{R}^5; (x_2, x_4, x_5) \in \mathbb{R}^3\}.$$

Si volem obtenir una base d'aquest espai, podem donar valors linealment independents a les variables independents. Per exemple, una base és:

$$\text{Nucli}\,(A) = \langle (0,1,0,0,0), (3,0,-3,1,0), (-1,0,-2,0,1) \rangle$$

i podem comprovar que la matriu B que té aquests elements com a columnes compleix que $AB = 0$

```
>> B=[0,3,-1;1,0,0;0,-3,-2;0,1,0;0,0,1]
B =
   0   3  -1
   1   0   0
   0  -3  -2
   0   1   0
   0   0   1
>> A*B
ans =
   0   0   0
   0   0   0
   0   0   0
```

Per estalviar-nos aquesta feina, tant `Matlab` com `Octave` disposen de la funció `null()`, que retorna una base del nucli de la matriu A. Per exemple, en `Matlab` obtindrem, per a la mateixa matriu que abans,

```
>> B=null(A)
B =
  -0.090670741455171  -0.155346175028831  -0.913407384037675
   0.984914997649172  -0.157835440158803  -0.070925462533632
   0.141913559500196   0.930181831028838  -0.099299537219049
  -0.035917226934504  -0.137874909009611  -0.191946136761811
  -0.017080939348342  -0.258278552000002   0.337568973752241
```

mentre que a `Octave`

```
>> B=null(A)
B =
   0.10178535502051834  -0.07020580394497834   0.92270122643380781
  -0.94696563078230744  -0.31101088166404989   0.08079805445440508
   0.27724713365307185  -0.88843922982695855  -0.17043431879680729
  -0.00818626929022618   0.08311418021522245   0.22398186351826926
  -0.12634416289119671   0.31954834459064563  -0.25075563587900018
```

Encara que la base surti diferent en tots dos casos, es compleix que `A*B` és (molt proper a) zero en ambdos casos. Això il·lustra el fet que hi ha moltes bases possibles per a un mateix subespai lineal.

Complement: Consideracions sobre l'estabilitat

És important tenir en compte que les ordres `rank()`, `null()` i `rref()` no es comporten bé per petites pertorbacions i que depenen d'una tolerància que fixem a priori com a argument opcional. Per exemple:

```
>> B=[1,1,2;1,1.0001,2]
B =
   1.0000   1.0000   2.0000
   1.0000   1.0001   2.0000
```

```
>> rank(B)
ans =  2
>> rank(B,1e-3) % precisio per al rang
ans =  1
```

Hi ha matrius, a més, que estan mal condicionades per a aquest tipus de problemes, és a dir que donen mals resultats. Per exemple, la matriu de Vandermonde en `1:12` sabem que és invertible perquè el seu determinant és diferent de zero (el vector `1:12` no té dues components iguals). En canvi si en calculem el rang,

```
>> rank(vander(1:12)) % problemes numerics...
ans =  11
```

cosa que és incorrecta.

6.4. Aplicació a la discussió de sistemes d'equacions lineals

Els conceptes que acabem de veure ens permeten discutir l'existència de solucions de sistemes lineals. En efecte, considerem un sistema $Ax = b$, amb

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

Si A_b és la matriu ampliada amb b com a darrera columna, en podem distingir diferents casos, segons el teorema de Rouché-Frobenius:

- Si $\text{rang}(A) = \text{rang}(A_b)$, el sistema és compatible.
  - Si $\text{rang}(A) = n$, el sistema és determinat i té una única solució, $x = A^{-1}b$, si $m = n$.
  - Si $\text{rang}(A) = \text{rang}(A_b) < n$, el sistema és compatible indeterminat i té infinites solucions.
- Si $\text{rang}(A) < \text{rang}(A_b)$, el sistema és incompatible i no té solucions.

Les solucions es calculen fàcilment a partir de la forma esglaonada reduïda de la matriu ampliada A_b, prenent com a variables dependents les pivot. Vegem-ho amb una sèrie d'exemples.

6.4.1 Sistema compatible determinat

Considerem el sistema d'equacions lineals següent:

$$\begin{aligned} x_1 + x_2 + x_3 &= 6, \\ x_1 - 2x_3 &= 4, \\ x_2 + x_3 &= 2 \end{aligned}$$

i la seva matriu augmentada en Matlab/Octave

```
>> A=[1,1,1;1,0,-2;0,1,1]
A =
   1   1   1
   1   0  -2
   0   1   1
>> b=[6;4;2]; % terme independent
>> Aamp=[A,b] % matriu ampliada
Aamp =
   1   1   1   6
   1   0  -2   4
   0   1   1   2
```

per a la qual calculem el rang i la forma esglaonada reduïda:

```
>> rank(Aamp)
ans =  3
>> [R,jb]=rref(Aamp)
R =
   1   0   0   4
   0   1   0   2
   0   0   1  -0
jb =
   1   2   3
```

de manera que de la forma reduïda obtenim que la solució és senzillament

$$x_1 = 4, x_2 = 2, x_3 = 0$$

i que és el mateix que obtindríem fent

```
>> A\b
ans =
   4
   2
  -0
```

6.4.2 Sistema incompatible

Considerem ara l'exemple de sistema d'equacions lineals següent

$$\begin{aligned} x_1 + x_2 + x_3 &= -6, \\ x_1 - 2x_3 &= 4, \\ 2x_1 - x_3 &= 18. \end{aligned}$$

Definim la matriu augmentada en Matlab/Octave:

```
>> A=[1,1,1;1,0,-2;2,1,-1]
```

```
A =
   1   1   1
   1   0  -2
   2   1  -1
>> b=[-6;4;18]; % terme independent
>> Aamp=[A,b] % matriu ampliada
Aamp =
    1    1    1   -6
    1    0   -2    4
    2    1   -1   18
```

i en calculem la forma esglaonada reduïda:

```
>> [R,jb]=rref(Aamp)
R =
   1   0  -2   0
   0   1   3   0
   0   0   0   1
jb =
   1   2   4
```

d'on veiem que el sistema és incompatible ja que obtenim l'equació $1 = 0$ (el rang de l'ampliada és 3, mentre que el rang de la matriu del sistema és només 2).

Ús de l'ordre *A\b* en sistemes incompatibles

Quan el sistema no tingui cap solució l'ordre `x=A\b` no ens pot donar cap solució que compleixi que `Ax` sigui `b`. Així, a l'exemple anterior, en `Matlab`

```
>> A\b
Warning: matrix singular to machine precision.
ans =
   -Inf
   -Inf
   -Inf
```

mentre que en `Octave` també ens avisa, però ens dóna un valor

```
>> A\b
warning: matrix singular to machine precision, rcond = 0
warning: attempting to find minimum norm solution
warning: dgelsd: rank deficient 3x3 matrix, rank = 2
ans =
   3.33333
   1.00000
  -3.66667
```

Podem comprovar que aquest valor no és cap solució, sinó que ens tria un valor especial que té a veure amb quant Ax s'aproxima més a b sobre tots els vectors x de $\mathbb{R}^3$.

Això també ens passarà quan tinguem un sistema sobredeterminat, amb més equacions que incògnites i que no tingui solució (cosa que succeeix, en general, en els sistemes sobredeterminats). Per exemple, si intentem resoldre

$$\begin{aligned} 3x_1 + 2x_2 + 5x_3 &= 1 \\ 4x_1 + 3x_2 + 6x_3 &= 0 \\ 5x_1 + 4x_2 + 7x_3 &= 0 \\ 6x_1 + 7x_2 + 8x_3 &= 0 \end{aligned}$$

farem

```
>> A=[3,2,5;4,3,6;5,4,7;6,7,8]
A =
   3   2   5
   4   3   6
   5   4   7
   6   7   8
>> b=[1;0;0;0];
>> v=A\b % atencio A no es quadrada!!!
v =
  -2.16667
   0.33333
   1.33333

>> A*v % comprovem que no es la solucio
ans =
   8.3333e-01
   3.3333e-01
  -1.6667e-01
   4.7740e-15
```

La raó és que `Matlab/Octave` interpreta que l'usuari ja sap que el sistema no té solució i, per tant, dóna el vector que minimitza el valor `A*v` en un cert sentit. És per això que cal anar amb compte amb els sistemes no quadrats perquè no podem usar l'ordre `A\b` cegament i cal comprovar-ne sempre la solució.

6.4.3 Sistema compatible indeterminat

Considerem, com a exemple, el sistema

$$\begin{aligned} x_1 + x_2 + x_3 &= 6, \\ x_1 - 2x_3 &= 4, \\ 2x_1 + x_2 - x_3 &= 10, \end{aligned}$$

definim de nou la matriu i la seva ampliada

```
>> A=[1,1,1;1,0,-2;2,1,-1];
>> b=[6;4;10];
>> Aamp=[A,b] % matriu ampliada
```

```
Aamp =
     1     1     1     6
     1     0    -2     4
     2     1    -1    10
```

i en calculem la forma esglaonada reduïda

```
>> [R,jb]=rref(Aamp)
R =
    1    0   -2    4
    0    1    3    2
    0    0    0    0
jb =
    1    2
```

El sistema és compatible indeterminat, ja que el rang de la matriu del sistema i el de la matriu ampliada són iguals, però menors que el nombre d'incògnites. Les solucions vénen donades per les equacions que satisfan les variables independents (les que no són pivot)

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

o, equivalentment, per

$$\begin{aligned} x_1 &= 4 + 2x_3 \\ x_2 &= 2 - 3x_3 \end{aligned}$$

on veiem clarament que x_3 és la variable independent.

Per veure'n un altre exemple, considerem el sistema lineal següent:

$$\begin{aligned} -4x_1 - 2x_2 + 2x_4 - 4x_5 + 4x_6 &= 2, \\ 4x_1 + x_2 - 3x_4 + 4x_5 - 4x_6 &= -3, \\ x_1 - 2x_2 - 3x_4 + x_5 - x_6 &= -3, \\ -2x_2 - 2x_4 &= -2. \end{aligned}$$

que té sis incògnites i quatre equacions. Definim directament la matriu ampliada

```
>> Aamp=[-4,-2,0,2,-4,4,2;4,1,0,-3,4,-4,-3; ... %...= partim la linia
 1,-2,0,-3,1,-1,-3;0,-2,0,-2,0,0,-2]
A =
   -4   -2    0    2   -4    4    2
    4    1    0   -3    4   -4   -3
    1   -2    0   -3    1   -1   -3
    0   -2    0   -2    0    0   -2
```

i en calculem la forma reduïda

```
>> [R,jb]=rref(Aamp)
```

```
R =
   1   0  -0  -1   1  -1  -1
   0   1  -0   1  -0  -0   1
   0   0   0   0   0   0   0
   0   0   0   0   0   0   0
jb =
   1   2
```

El sistema és, doncs, compatible indeterminat i les solucions vénen donades per les equacions que ens indica

```
>> R(jb,:)
ans =
   1   0  -0  -1   1  -1  -1
   0   1  -0   1  -0  -0   1
```

és a dir,

$$\begin{aligned} x_1 - x_4 + x_5 - x_6 &= -1 \\ x_2 + x_4 &= 1 \end{aligned}$$

o bé

$$\begin{aligned} x_1 &= -1 + x_4 - x_5 + x_6 \\ x_2 &= 1 - x_4 \end{aligned}$$

de manera que $x_4, x_5, x_6 \in \mathbb{R}$ són variables independents, i x_1 i x_2 són les dependents.

Ús de l'ordre $A\backslash b$ en sistemes compatibles indeterminats

Com abans, podríem aplicar ingènuament l'ordre `A\b` per trobar una de les solucions que té el sistema. En aquest cas, el comportament en `Matlab` i en `Octave` és diferent. Així, en `Octave`:

```
>> xpart=A(:,1:3)\A(:,4) % problemes amb la particular
warning: matrix singular to machine precision, rcond = 0
warning: attempting to find minimum norm solution
warning: dgelsd: rank deficient 3x3 matrix, rank = 2
xpart =
   3.71429
   2.42857
  -0.14286
>> A(:,1:3)*xpart % comprovem que es solucio
ans =
    6.0000
    4.0000
   10.0000
```

obtindrem `xpart` com a solució particular i un avís que el rang és 2 i que, per tant, hi ha un conjunt infinit de solucions. En canvi, en `Matlab`, el sistema ens dóna un avís però cap solució:

```
>> xpart=A(:,1:3)\A(:,4) % problemes amb la particular
Warning: matrix singular to machine precision.
xpart =
   NaN
   NaN
   NaN
```

De nou, cal ser molt curosos a l'hora d'aplicar `A\b` amb sistemes no quadrats!

6.5. Exercicis

6.1. Discutiu el sistema d'equacions següent:

$$\begin{aligned} 3x_1 + 2x_2 + 5x_3 &= 1 \\ 4x_1 + 3x_2 + 6x_3 &= 2 \\ 5x_1 + 4x_2 + 7x_3 &= 3 \\ 6x_1 + 7x_2 + 8x_3 &= 4 \end{aligned}$$

Resposta: El sistema és compatible determinat i la seva solució val $(2, 0, -1)$.

6.2. Donat el sistema d'equacions següent:

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ -1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} 90 \\ -50 \\ 130 \\ -150 \\ 80 \\ -100 \end{pmatrix}$$

trobeu una solució que compleixi que $x_6 = 0$.

Resposta: $x_1 = 100, x_2 = 10, x_3 = 60, x_4 = -70, x_5 = 80$.

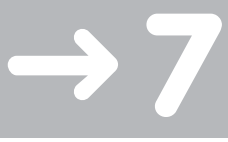
6.3. Donada la matriu A següent:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 2 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 4 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 7 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 & 0 & 4 & 5 & 6 & 7 \\ 0 & 0 & 0 & 0 & 0 & 8 & 9 & 10 & 11 \\ 0 & 0 & 0 & 0 & 0 & 12 & 13 & 14 & 15 \end{pmatrix}$$

Quin és el seu rang?

6.6. Esquema de la pràctica

- Resolució de sistemes quadrats compatibles determinats $Ax = b$: `A\b`, on `A` és una matriu quadrada no singular i `b` és un vector columna amb el mateix nombre d'elements que la dimensió de la matriu. Comprovació del resultat: `A*(A\b)` ha de ser proper a zero.
- Rang d'una matriu: `rank(A)`.
- Forma esglaonada reduïda d'una matriu: `[R,jb]=rref(A)`, on `R` és la forma esglaonada reduïda, de manera que:
  - `r = length(jb)` dóna el rang de `A`.
  - `jb` conté els índexs de les columnes pivot.
  - `R(1:r,jb)` és la identitat.
- Càlcul de la imatge de `A`: `A(:,jb)` és una base de la imatge de `A`.
- Càlcul del nucli de `A`: `null(A)` dóna una base del nucli de `A`. També podem trobar-ne una usant la forma esglaonada reduïda `[R,jb]=rref(A)`: el sistema d'equacions $Rx = 0$ ens dóna una expressió parametritzada del nucli de `A`, prenent com a variables independents les no pivot. Només cal, doncs, assignar-los valors linealment independents.
- Resolució de sistemes d'equacions generals $Ax = b$:
  - És incompatible si `rank(A)` $<$ `rank([A,b])`. L'ordre `A\b` pot retornar un vector `x` que minimitza $Ax - b$ en algun sentit però que no és la solució.
  - És compatible si `rank(A)` $=$ `rank([A,b])` i el sistema equivalent `R(:,1:end-1) x= R(:,end)`, on `[R,jb]=rref([A,b])` és la forma esglaonada reduïda de la matriu ampliada, dóna directament la solució parametritzada prenent com a variables dependents les indicades per `jb`.



Mètodes d'integració numèrica

Els mètodes d'integració numèrica per a integrals definides permeten obtenir-ne una aproximació numèrica sense haver de conèixer una primitiva. Això és important, ja que molts cops no coneixem una primitiva de la integral (penseu en $\int e^{-x^2} dx$) o bé és molt complicada d'avaluar. Els mètodes que veurem són de dos tipus: simples i composts. Els mètodes simples que presentarem aproximen una integral en un interval compacte a través d'una combinació lineal de valors de la funció en els extrems (en el cas dels trapezis) i d'un punt intermedi (en el cas de la fórmula de Simpson). Els mètodes compostos divideixen l'interval en un nombre de subintervals més petits i en cadascun d'aquests usen un mètode simple.

7.1. El mètode dels trapezis

El mètode dels trapezis aplicat a una funció $f : [a,b] \to \mathbb{R}$ consisteix a aproximar el valor de la integral per la mitjana dels valors en els dos extrems:

$$\int_a^b f(x)\,dx \approx (b-a)\frac{f(a)+f(b)}{2}.$$

i té la interpretació gràfica que es mostra a la figura 7.1. Per trobar aquesta fórmula, busquem els coeficients W_0 i W_1 de manera que la fórmula

$$\int_a^b f(x)\,dx = W_0 f(a) + W_1 f(b)$$

sigui exacta per als polinomis de grau més gran possible. En aquest cas, com que només tenim dues incògnites, W_0 i W_1, només podem imposar que sigui exacta sobre els polinomis de grau 1. Com que la relació és lineal, n'hi ha prou a imposar-ho sobre una base, com per exemple 1 i $x-(a+b)/2$. Així, tenim un sistema de dues equacions amb dues incògnites:

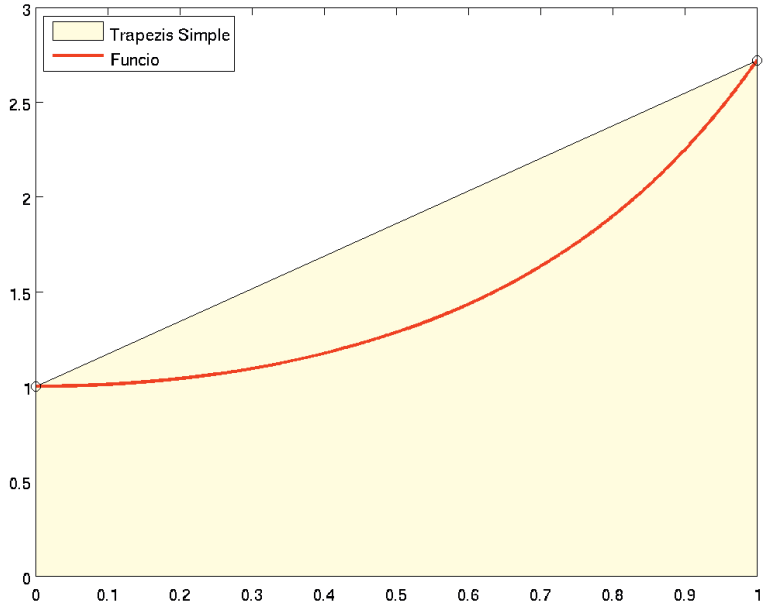


Fig. 7.1
Il·lustració del mètode dels trapezis simple.

$$\int_a^b 1\,dx = b-a = 1\cdot W_0 + 1\cdot W_1$$

$$\int_a^b \left(x - \frac{a+b}{2}\right) dx = 0 = \frac{a-b}{2}\cdot W_0 + \frac{b-a}{2}\cdot W_1$$

que dóna com a solució $W_0 = W_1 = (b-a)/2$.

Error en el mètode dels trapezis

Per a una funció $f : [a,b] \to \mathbb{R}$ dos cops derivable amb continuïtat, l'error d'aplicar la fórmula dels trapezis simple s'obté de la igualtat següent:

$$\int_a^b f(x)\,dx = \underbrace{(b-a)\frac{f(a)+f(b)}{2}}_{\text{Fórmula trapezis simple}} \underbrace{-\frac{h^3}{12}f^{(2)}(\xi)}_{\text{Error trapezis simple}}$$

on $f^{(2)}(\xi)$ indica la derivada segona de f en un cert punt $\xi \in (a,b)$ i $h = b-a$ és la longitud de l'interval. Per exemple, si tenim la funció $f(x) = (x-a)^2$ en $[a,b]$, aleshores l'error seria

$$\text{Error trapezis simple de}(x-a)^2 = -\frac{h^3}{12}f^{(2)}(\xi) = -\frac{h^3}{6}$$

ja que la derivada segona de f és constant igual a 2.

7.1.1 El mètode compost

Com hem dit, la idea d'un mètode compost és dividir l'interval $[a,b]$ en n subintervals, iguals en el nostre cas, i en cadascun d'aquests apliquem un mètode d'integració numèrica (per exemple, el del mètode dels trapezis), tal com s'indica a la figura 7.2. Sumant les fórmules de cadascun dels intervals, obtenim la *fórmula composta de trapezis* en n subintervals o, equivalentment, de pas $h = (b-a)/n$,

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2n}\left(f(x_0)+2f(x_1)+2f(x_2)+\cdots+2f(x_{n-1})+f(x_n)\right)$$

on

$$x_k = a + k\frac{b-a}{n}, \text{ per a } k = 0,1,\ldots,n$$

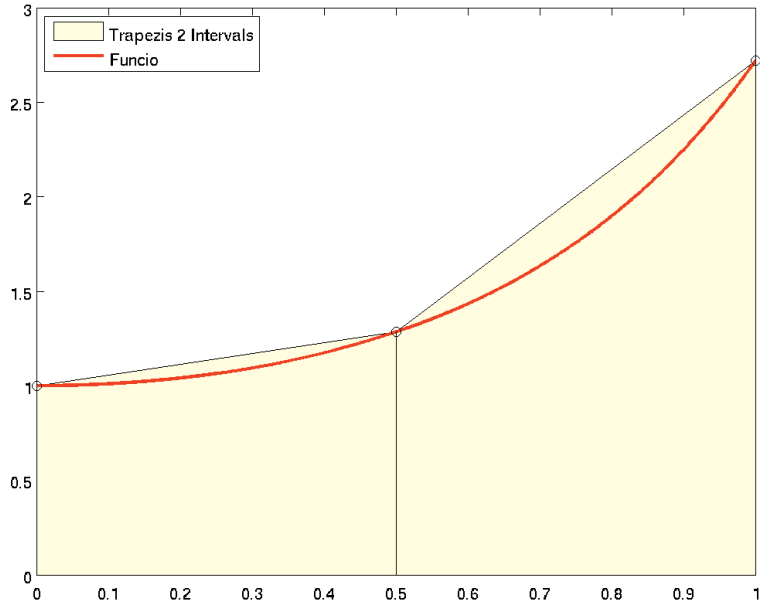


Fig. 7.2
Il·lustració del mètode dels trapezis compost amb dos intervals.

Error en el mètode dels trapezis compost

Es pot veure que la fórmula que ens dóna l'error en el mètode dels trapezis compost per a una funció $f : [a,b] \to \mathbb{R}$, dos cops diferenciable amb continuïtat, obtingut en dividir l'interval $[a,b]$ en n subintervals de longitud $h = (b-a)/n$, $E_T(h)$, és la següent:

$$E_T(h) = \frac{b-a}{12}h^2 f^{(2)}(\xi)$$

on $\xi \in (a,b)$. Observem que, a mesura que prenem més subintervals (augmentem la n), l'error és cada cop més petit (direm que és d'odre h^2). Al final d'aquest apartat es demostra aquesta fórmula.

Com a exemple, calculem ara, amb un error menor a 10^{-2}, la integral

$$\int_0^1 e^{x^2} dx$$

En aquest cas, $f(x) = e^{x^2}$. Per poder aplicar la fórmula de l'error d'abans, cal que n'acotem la seva derivada segona en $[0,1]$:

$$f''(x) = 2e^{x^2} + 4x^2 e^{x^2} \Rightarrow |f''(x)| \leq 6e, \quad x \in [0,1]$$

Podem usar la fórmula de l'error:

$$|E_T(h)| = \frac{h^2}{12}|f''(\xi)| \leq \frac{h^2}{12} \sup_{x\in[0,1]} |f''(x)| \leq \frac{h^2}{12} 6e = \frac{e}{2}h^2$$

Si volem que l'error sigui menor a 10^{-2}, cal que trobem un $h = (1-0)/n$ de manera que

$$\frac{e}{2}h^2 < 10^{-2} \Leftrightarrow h^2 < \frac{2}{100e} \Leftrightarrow h < \sqrt{\frac{2}{100e}} \Leftrightarrow n > 1/\sqrt{\frac{2}{100e}} = \sqrt{50e} \approx 11.7$$

Per tant, prenent una partició de dotze intervals podem assegurar que l'error és inferior a 10^{-2}:

$$\int_0^1 e^{x^2} dx \approx \frac{h}{12}\left(f(0) + \sum_{k=1}^{11} f(\frac{k}{12}) + f(1)\right) = 1.4658\ldots$$

Per fer aquest càlcul, podem usar l'expressió següent en `Matlab/Octave`:

```
>> h=1/12;x=0:h:1; f= exp(x.^2); trapz(x,f)
ans =
   1.465794273401113
```

que veurem amb més detall a la pràctica següent.

Complement opcional: Demostració de la fórmula de l'error

Per obtenir la fórmula de l'error, podem acotar l'error comès en cada interval $[x_{k-1}, x_k]$ si hi apliquem la fórmula simple dels trapezis

$$\int_{x_{k-1}}^{x_k} f(x)dx = \frac{h}{2}\left(f(x_{k-1}) + f(x_k)\right) - \frac{h^3}{2} f^{(2)}(\xi_k), \quad \xi_k \in (x_{k-1}, x_k).$$

Així, la fórmula total per a l'error total és

$$\int_a^b f(x)dx = \sum_{k=1}^{n} \int_{x_{k-1}}^{x_k} f(x)dx = \sum_{k=1}^{n} \frac{h}{2}\left(f(x_{k-1}) + f(x_k)\right) - \frac{h^3}{12}\sum_{k=1}^{n} f^{(2)}(\xi_k) = T(h) - E_T(h).$$

Ara usarem el resultat següent:

Teorema 1 (Teorema del valor mitjà per a sumes) *Sigui $F : I \to \boldsymbol{R}$ contínua i $\xi_k \in I$ una successió creixent de punts diferents i $\alpha_k > 0$, per a $k = 1, \ldots, n$. Aleshores, existeix un $\xi \in I$ de manera que*

$$\sum_{k=1}^{n} \alpha_k F(\xi_k) = F(\xi) \sum_{k=1}^{n} \alpha_k.$$

Usant aquest resultat, podem escriure

$$E_T(h) = \frac{h^3}{2} \sum_{k=1}^{n} f^{(2)}(\xi_k) = f^{(2)}(\xi) \sum_{k=1}^{n} 1 = \frac{b-a}{12} h^2 f^{(2)}(\xi)$$

ja que

$$h = \frac{b-a}{n}.$$

7.2. El mètode de Simpson

La idea del mètode simple de Simpson és trobar una fórmula del tipus

$$\int_a^b f(x)dx \approx W_0 f(a) + W_1 f\left(\frac{a+b}{2}\right) + W_2 f(b)$$

on W_0, W_1 i W_2 són unes constants per determinar de manera que la fórmula sigui exacta fins a polinomis de grau 2. Com abans, només cal que ho imposem sobre una base, en aquest cas la formada per 1, $x-c$ i $(x-c)^2$, on $c = (a+b)/2$ és el punt mitjà de l'interval $[a,b]$:

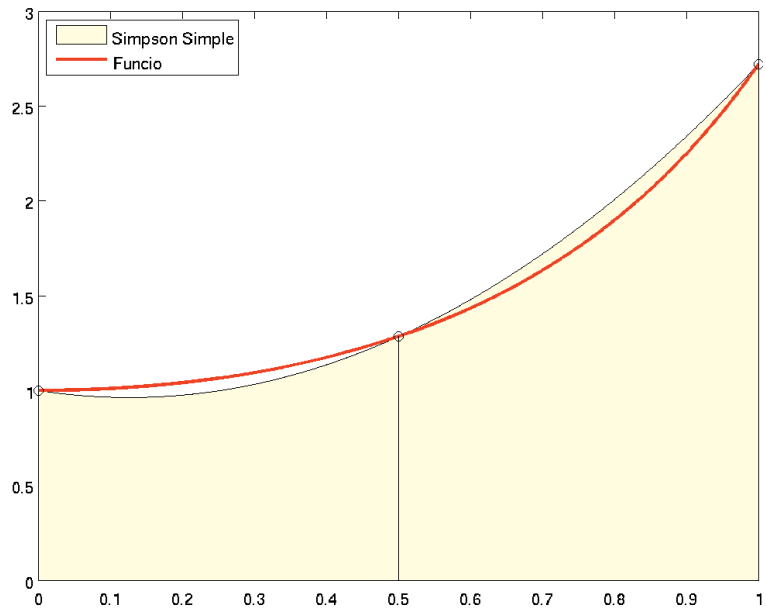


Fig. 7.3
Il·lustració de la fórmula simple de Simpson.

$$\begin{cases} f(x)=1 \Leftrightarrow b-a=W_0+W_1+W_2 \\ f(x)=x-c \Leftrightarrow 0=(a-c)W_0+(b-c)W_2 \\ f(x)=(x-c)^2 \Leftrightarrow \dfrac{(b-c)^3}{3}-\dfrac{(a-c)^3}{3}=\dfrac{(b-a)^2}{4}W_0+\dfrac{(b-a)^2}{4}W_2 \end{cases}$$

La solució d'aquest sistema és $W_0=W_2=(b-a)/6$ i $W_1=2(b-a)/3$, de manera que la fórmula simple de Simpson és

$$\int_a^b f(x)dx \approx \frac{b-a}{6}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right).$$

tal com s'il·lustra a la figura 7.3.

7.2.1 El mètode compost de Simpson

Per trobar la fórmula composta, prenem un nombre parell $n=2m$ i $n+1$ punts $x_0,\ldots,x_n$ equispaïats de manera que $x_0=a$ i $x_n=b$. Apliquem la fórmula de Simpson simple que hem obtingut abans als m intervals següents:

$$I_1=[x_0,x_2], I_2=[x_2,x_4],\ldots \quad I_m=[x_{n-2},x_n].$$

A cada interval $I_j=[x_{2j-2},x_{2j}]$ $(j=1,\ldots m)$, tenim

$$\int_{x_{2j-2}}^{x_{2j}} f(x)dx \approx \frac{h}{3}\left(f(x_{2j-2})+4f(x_{2j-1})+f(x_{2j})\right), \quad h=\frac{x_{2j}-x_{2j-2}}{2}$$

(això comporta que $h=\frac{b-a}{n}$). Així

$$\int_a^b f(x)dx = \sum_{j=1}^{m}\int_{x_{2j-2}}^{x_{2j}} f(x)dx \approx \frac{h}{3}\left(f(x_0)+4f(x_1)+2f(x_2)+4f(x_3)+\cdots+4f(x_{n-1})+f(x_n)\right).$$

7.2.2 Estudi de l'error

Cal distingir entre el mètode "clàssic" i el "compost":

- En el mètode clàssic, es pot veure que, si $f\in C^4[a,b]$, l'error és quíntic:

$$\int_a^b f(x)dx-\frac{h}{3}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right)=-\frac{h^5}{90}f^{(4)}(\xi),$$

  on $\xi\in(a,b)$ i $h=(b-a)/2$.

- En el mètode compost, de la mateixa manera, és d'un ordre menys:

$$\int_a^b f(x)dx - S(h) = -\frac{b-a}{180}h^4 f^{(4)}(\xi) = E_S(h),$$

on ara $h = (b-a)/n$ (n és el nombre d'intervals).

Com a exemple, calculem, per mitjà del mètode compost de Simpson i amb un error inferior a 10^{-2}, la integral d'abans

$$\int_0^1 e^{x^2} dx$$

Per poder aplicar la fórmula de l'error d'abans, cal que n'acotem la seva derivada quarta de $f(x) = e^{x^2}$ en $[0,1]$:

$$f^{(4)}(x) = (12 + 48x^2 + 16x^4)e^{x^2}, \quad |f^{(4)}(x)| \leq 76e, \quad x \in [0,1]$$

Podem usar la fórmula de l'error de Simpson:

$$|E_S(h)| = \frac{h^4}{180}|f^{(4)}(\xi)| \leq \frac{h^4}{180}\sup_{x\in[0,1]}|f^{(4)}(x)| \leq \frac{h^4}{180}76e = \frac{19e}{45}h^4$$

Si volem que l'error sigui inferior a 10^{-2}, cal que trobem un $h = (1-0)/n$ de manera que

$$\frac{19e}{45}h^4 < 10^{-2} \Leftrightarrow h^4 < \frac{45}{1900e} \Leftrightarrow h < \left(\frac{45}{1900e}\right)^{1/4} \Leftrightarrow n > 1/\left(\frac{45}{1900e}\right)^{1/4} \approx 3.327\ldots$$

En principi, volem que n sigui el menor possible (el nombre mínim d'intervals que garanteix l'error) i, per tant, podem triar $n = 4$ (cal que sigui parell). Llavors, sabem que l'error per Simpson és inferior a 10^{-2}:

$$\int_0^1 e^{x^2} dx \approx \frac{h}{3}\left(f(0) + 4f(0.25) + 2f(0.5) + 4f(0.75) + f(1)\right)$$

Per fer aquest darrer càlcul en `Matlab/Octave` (usant-lo com a calculadora; a la pràctica següent, veurem com fer-ho de manera més compacta), faríem:

```
>> n=4;h=1/n;x=0:h:1;f=exp(power(x,2));
>>(h/3)*(f(1)+4*f(2)+2*f(3)+4*f(4)+f(5))
ans =
   1.463710760445597
```

7.2.3 Relació entre els mètodes de Simpson i dels Trapezis

Dividint l'interval $[a,b]$ en $2n$ subintervals i en n subintervals, tenim que la partició en $2n$ intervals té per extrems

$$a = x_0 < x_1 < x_2 < \cdots < x_{2n-1} < x_{2n} = b,\ h = \frac{b-a}{2n},$$

mentre que la de n intervals té per extrems

$$a = x_0 < x_2 < x_4 < \cdots < x_{2n-2} < x_{2n} = b,\ 2h = \frac{b-a}{n}.$$

Així, les fórmules amb pas $h = (b-a)/2n$ i $2h = (b-a)/n$ són

$$\begin{aligned} T(h) &= \frac{h}{2}\left(f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{2n-1}) + f(x_{2n})\right) \\ T(2h) &= h\left(f(x_0) + 2f(x_2) + 2f(x_4) + \cdots + 2f(x_{2n-2}) + f(x_{2n})\right). \end{aligned}$$

Sumant adequadament aquestes fórmules, obtenim la relació següent:

$$\begin{aligned} S(h) &= \frac{h}{3}\left(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \cdots + 4f(x_{n-1}) + f(x_n)\right) \\ &= T(h) + \frac{1}{3}\left(T(h) - T(2h)\right) = \frac{4}{3}T(h) - \frac{1}{3}T(2h) \end{aligned}$$

si $S(h)$ representa la fórmula de Simpson amb pas $h = (b-a)/(2n)$ en $2n$ intervals.

7.3. Esquema de la pràctica

- Donada una funció $f : [a,b] \to \mathbb{R}$, es tracta d'aproximar la integral definida $\int_a^b f(x)dx$.

- Mètode dels trapezis

  - Fórmula simple dels trapezis:

    $$\int_a^b f(x)\,dx \approx \frac{b-a}{2}\left(f(a) + f(b)\right).$$

  - Fórmula composta dels trapezis en n subintervals o, equivalentment, de pas $h = (b-a)/n$,

    $$\int_a^b f(x)\,dx \approx \frac{h}{2}\left(f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{N-1}) + f(x_n)\right)$$

    on

    $$x_k = a + k\frac{b-a}{n} = a + kh, \text{ per a } k = 0, 1, \ldots, n$$

  - Error en la fórmula composta dels trapezis en n subintervals, pas $h = (b-a)/n$,

    $$E_T(h) = \frac{b-a}{12}h^2 f^{(2)}(\xi)$$

    on $\xi \in (a,b)$, si f és dos cops derivable amb continuïtat en $[a,b]$.

- Fórmula de Simpson

  - Fórmula simple de Simpson:

$$\int_a^b f(x)dx \approx \frac{b-a}{6}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right)$$

  - Fórmula composta de Simpson en un nombre parell de subintervals, $n = 2m$ o, equivalentment, de pas $h = (b-a)/n$,

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left(f(x_0)+4f(x_1)+2f(x_2)+4f(x_3)+\cdots+4f(x_{n-1})+f(x_n)\right)$$

    on

$$x_k = a+k\frac{b-a}{n} = a+kh, \text{ per a } k = 0,1,\ldots,n$$

  - Error en la fórmula composta de Simpson en n subintervals, pas $h = (b-a)/n$,

$$E_S(h) = -\frac{b-a}{180}h^4 f^{(4)}(\xi),$$

    on $\xi \in (a,b)$ si f és quatre cops derivable amb continuïtat en $[a,b]$.

- Relació entre la fórmula composta de Simpson amb pas $h = (b-a)/n$ i les compostes de trapezis amb pas h, $T(h)$ i amb pas $2h$, $T(2h)$,

$$S(h) = \frac{4}{3}T(h) - \frac{1}{3}T(2h)$$

Integració numèrica en Matlab/Octave

En aquesta pràctica, veiem com implementar els mètodes compostos de trapezis que hem vist a la pràctica anterior en `Matlab/Octave`. Tot i que seria possible fer-ho a través dels iteradors o de l'ordre `sum()` per a vectors (podeu provar de fer-ho), aquí usarem les ordres pròpies del sistema. Finalment, veurem com cridar el mètode de quadratura adaptativa `quadl()`, que ens permet especificar un error determinat.

8.1. Càlcul de sumes de Riemann

Observem que, amb la notació vectorial de `Matlab/Octave`, és fàcil obtenir una aproximació per a la integral basant-nos en les sumes de Riemann

$$\int_a^b f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

per a una tria de punts equiespaiats $x_i = a + (b-a)\frac{i}{N}$, com s'il·lustra a la figura 8.1. En efecte, per calcular

$$\int_0^1 e^{x^2} dx$$

podríem fer

```
>> N=10; h=1/N; x=0:h:1-h;
>> fx=exp(x.^2);
>> int_riem10=sum(fx)/N
int_riem10 =
   1.381260601315846
```

i millorar-ne la precisió prenent molts més punts:

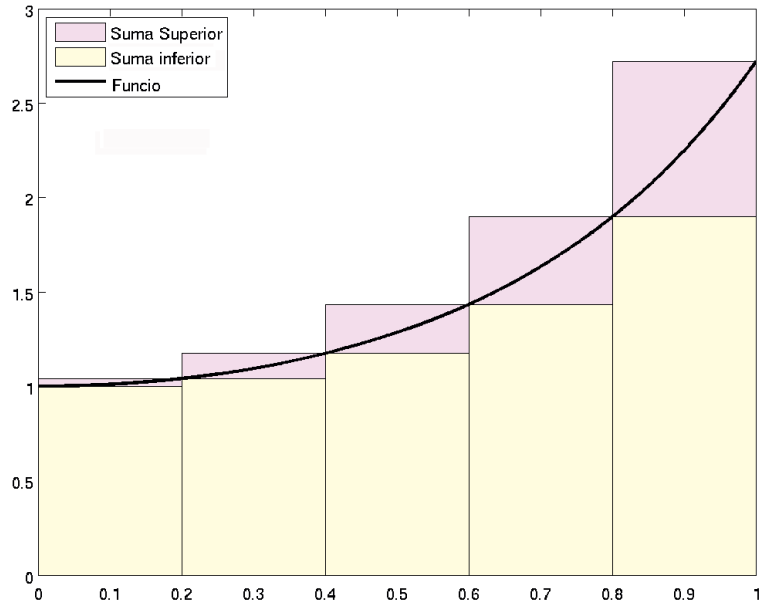


Fig. 8.1
Il·lustració de l'ús de les sumes de Riemann per a aproximar una integral.

```
>> N=1000; h=1/N; x=0:h:1-h;
>> fx=exp(x.^2);
>> int_riem1000=sum(fx)/N
int_riem1000 =
   1.461793058039847
```

que encara és lluny del valor real de la integral, aproximadament `1.46265174590718`. En millorarem la precisió amb els mètodes que hem vist veure a la pràctica anterior.

8.2. Trapezis i Simpson compostos en Matlab/Octave

Per veure com faríem el càlcul dels trapezis i de Simpson compostos en `Matlab/Octave`, calculem

$$\int_0^1 e^{x^2} dx$$

amb els mètodes dels trapezis i de Simpson compostos i amb quatre intervals. En primer lloc, definim la funció a `funcio.m`

```
function y=funcio(x)
y=exp(x.^2);
```

Per calcular trapezis amb $n = 4$ intervals, només cal que li passem dos vectors a la instrucció `trapz()`: el vector dels extrems dels intervals (els $x_0, \ldots, x_n$ d'abans) i les imatges per la funció en aquests punts ($f(x_0), \ldots, f(x_n)$):

```
>> n=4 % sobre quatre punts
n =
     4
```

```
>> h=1/n % pas que considerarem
h =
   0.250000000000000
>> x=0:h:1; % punts sobre els quals farem trapezis
>> f=funcio(x); % imatge de la funcio en els punts
>> trapezis4=trapz(x,f) % trapezis
trapezis4 =
   1.490678861698855
```

Per calcular el valor de la fórmula composta de Simpson, utilitzem la relació

$$S(h) = \frac{4}{3}T(h) - \frac{1}{3}T(2h)$$

i, per tant, en primer lloc calculem el valor de la fórmula composta dels trapezis per a $n = 2$ subintervals (és a dir, amb pas $2h$):

```
>> n=2;h=1/n;
>> x=0:h:1;
>> f=funcio(x);
>> trapezis2=trapz(x,f)
trapezis2 =
   1.571583165458632
```

Finalment, apliquem la relació entre trapezis i Simpson:

```
>>simpson4=trapezis4+(trapezis4-trapezis2)/3 % formula!
simpson4 =
   1.463710760445596
```

que ens dóna l'aproximació de la fórmula composta de Simpson

8.3. Quadratura adaptativa de Gauss-Legendre-Lobato

Els mètodes de quadratura adaptativa permeten "controlar" l'error en usar una fórmula numèrica d'integració i decidir passos adequats d'integració. Nosaltres seguim un mètode adaptatiu que s'anomena de Gauss-Legendre-Lobato (i que podeu esbrinar fent `help quadl` i a la referència [16]). A `Matlab/Octave`, això s'implementa amb l'expressió

```
intNum = quadl(@funcio, a, b, tol)
```

- `funcio` és el nom de fitxer on tenim definida la funció ('`funcio.m`').
- `a` és l'extrem inferior de la integral.
- `b` és l'extrem superior de la integral.
- `tol` és opcional i especifica la tolerància de l'error. Tot i que és opcional, les toleràncies per defecte poden variar segons `Matlab` i `Octave` i és millor especificar-les.

Recordem que 10^{-6} s'escriu com `1e-6`. Per exemple, per calcular la integral d'abans mitjançant el mètode de la quadratura adaptativa en `Matlab/Octave`,

$$\int_0^1 e^{x^2} dx$$

i suposant que al fitxer `funcio.m` hi tenim

```
function y=funcio(x)
y=exp(x.^2);
```

podem usar el mètode de la quadratura adaptativa per a la tolerància `1e-6` i obtindrem amb `Matlab`

```
>> integracio6=quadl(@funcio,0,1,1e-6)
integracio6 =
   1.462651763477989
```

mentre que amb `Octave` la mateixa ordre donaria pràcticament el mateix:

```
>> integracio6=quadl(@funcio,0,1,1e-6)
integracio6 =  1.46265176347799
```

Podem millorar la tolerància a 10^{-12}, de forma que amb `Matlab` obtindríem

```
>> integracio12=quadl(@funcio,0,1,1e-12)
integracio12 =
   1.462651745907185
```

mentre que amb `Octave` tindríem `1.46265174590718`.

Complement opcional: Estudi de la tolerància

Aquestes diferències de tolerància es poden veure millor a l'exemple següent, que té "punxes" (valors on la funció no és derivable). Per exemple, fent

$$\int_{-1}^{3} |x^2 - 2x - 1| dx = \frac{16}{3}\sqrt{2} - \frac{8}{3} \approx 4.87580566598984\ldots$$

i definint la funció $f(x) = |x^2 - 2x - 1|$ en el fitxer `funcio.m`, tindríem amb `Matlab`:

```
>> integracio6=quadl(@funcio,-1,3,1e-6)
integracio6 =  4.875806487222102 % error= 8.2e-7
>> integracio12=quadl(@funcio,-1,3,1e-12)
integracio12 = 4.875805665989855 % error= 1.5e-14
```

i, amb Octave, uns errors similars:

```
>> integracio6=quadl(@funcio,-1,3,1e-6)
integracio6 =  4.87580648722210 % error 8e-7
>> integracio12=quadl(@funcio,-1,3,1e-12)
integracio12 = 4.87580566599395 % error 4e-9
```

8.4. Repàs d'integració en Matlab/Octave

A tall d'exemple, calculem ara la integral

$$\int_{10^{-22}}^{\pi/3} \cos(x)e^x dx$$

1. Amb trapezis compostos, $N = 10$.
2. Amb Simpson compost, $N = 20$.
3. Amb quadratura adaptativa de Gauss-Legendre-Lobato i error inferior a 10^{-10}.

Per començar, definim els extrems d'integració per no haver-los d'escriure cada cop:

```
>>a=10^(-22); b=pi/3
```

Mètode dels trapezis ($n = 10$)

A 'funcio.m': definim la funció $\cos(x)e^x$:

```
function y= funcio(x)
  y= cos(x).*exp(x);
```

el punt .* (com ./ o .^) és perquè volem que operi component a component.

```
>> format long
>> a=10^(-22); b=pi/3; n=10; h=(b-a)/n;
>> x=a:h:b; f=funcio(x);
>> trapezis=trapz(x,f)
trapezis =
   1.444483740029542
```

Mètode de Simpson ($n = 20$)

Primer calculem trapezis amb $n = 20$ intervals.

```
>>a=10^(-22); b=pi/3
>>n=20;h=(b-a)/n;x=a:h:b;f=funcio(x);
>>trapezis20=trapz(x,f)
trapezis20 =
   1.445883115227841
```

Fem el mateix per a $n = 10$ (és a dir, amb pas $2h$). Aquest és el mateix càlcul que hem fet a l'apartat anterior, però el repetim aquí:

```
>>n=10;h=(b-a)/n;x=a:h:b;f=funcio(x);
>>trapezis10=trapz(x,f)
trapezis10 =
   1.444483740029542
```

i, finalment, apliquem la relació entre Simpson i trapezis compostos

```
>>simpson20=4*trapezis20/3-trapezis10/3
simpson20 =
   1.446349573627274
```

Quadratura adaptativa de Gauss-Legendre-Lobato (error$< 10^{-10}$)

Com que al fitxer `'funcio.m'` tenim definida la funció, podem cridar l'ordre `quadl()` directament:

```
>>a=10^(-22); b=pi/3;
>>quadratura=quadl(@funcio,a,b,1e-10)
quadratura =
   1.446349815311582
```

Així, recapitulant, hem obtingut els valors següents:

Mètode emprat	**Aproximació**	**Error**
Integració exacta	1.446349815315410	0 (exacte)
Trapezis compostos ($n = 10$)	1.444483740029542	1.910^{-3}
Simpson compost ($n = 20$)	1.446349573627274	2.410^{-7}
Quadr. adapt. GLL (tol=10^{-10})	1.446349815311582	3.810^{-12}

Observem que, encara que havíem demanat un error de 10^{-10}, l'error real en la quadratura és inferior.

8.5. Exercicis

8.1. Calculeu la integral següent

$$\int_0^{\pi/3} \log(\cos x)dx,$$

usant la regla de Simpson amb 16 intervals (és a dir, amb pas $h = \frac{\frac{\pi}{3}-0}{16}$).

Resposta: -0.218392609216664.

8.2. Donada la integral següent

$$\int_1^{\pi+1} \frac{1}{x+e^x}dx,$$

sigui T el resultat d'aplicar la regla dels trapezis amb 8 intervals (és a dir, amb pas $h = \frac{\pi}{8}$) i Q el resultat d'aplicar la quadratura adaptativa de Gauss-Legendre-Lobato amb una tolerància de 10^{-12}. Digueu quant val $|T - Q|$.

Resposta: 0.003253621006893.

8.3. Calculeu la integral següent

$$\int_{10^{-22}}^{\pi/3} \cos(x) e^{-x^2} dx$$

1. Amb el mètode dels trapezis compostos, $N = 10$.
2. Amb el mètode de Simpson compost, $N = 20$.
3. Amb quadratura adaptativa i error inferior a 10^{-10}.

8.6. Esquema de la pràctica

- Donada una funció $f : [a,b] \to \mathbb{R}$, es tracta d'aproximar la integral definida $\int_a^b f(x)$. Suposem definida la funció al fitxer `funcio.m` (amb la notació vectorial: `./`, `.*`, `.^`).
- dels trapezis: `trapz(x,fx)` dóna la formula de trapezis compostos en les abscisses del vector `x` i `fx=funcio(x)` el vector de les seves imatges.
- Per calcular la fórmula composta de Simpson amb pas $h = (b-a)/n$, calculem el resultat de les fórmules compostes dels trapezis amb pas h, $T(h)$ i amb pas $2h$, $T(2h)$, i usem la relació

$$S(h) = \frac{4}{3}T(h) - \frac{1}{3}T(2h).$$

- El mètode de quadratura adaptativa de Gauss-Legendre-Lobato es crida de la forma

```
intNum = quadl(@funcio, a, b, tol)
```

  on:

  - `funcio` és el nom de fitxer on tenim definida la funció ('`funcio.m`').
  - `a` és l'extrem inferior de la integral.
  - `b` és l'extrem superior de la integral.
  - `tol` és la tolerància per a l'error. Tot i que és opcional, les toleràncies per defecte poden variar segons `Matlab` i `Octave` i, per tant, és millor especificar-les.

Control de flux

Per a molts problemes de matemàtiques (successions, sèries, equacions en diferències, ...), cal repetir una mateixa instrucció uns quants cops, potser actualitzant una variable a cada iteració. En aquesta pràctica, veiem com s'escriuen iteradors en `Matlab/Octave` i ho apliquem a equacions en diferències i a la suma de sèries.

9.1. Iteradors i successions recurrents

Considerem la successió $(x_n)_n$ definida recurrentment de la manera següent:

$$x_{n+1} = \cos x_n$$

amb $x_1 = 0$. Usant l'assignació de variables:

```
>> x=0;
>> x=cos(x)
x =  1
>> x=cos(x)
x =  0.540302305868140
>> x=cos(x)
x =  0.857553215846393
```

podem escriure uns quants iterats, però per calcular-los per a valors elevats de n hem d'usar algun iterador. Els iteradors en `Matlab/Octave` s'escriuen mitjançant l'expressió `for variable =rang` i s'acaben amb la paraula `end`. Per exemple, per calcular el terme x_{10} de la successió anterior, faríem

```
x = 0
>  for n=2:10 % comencem a x_1
=0 x=cos(x); % el nou x es x_n
end
>  x
x =  0.750417761763761
```

i `x` conté ara el darrer valor de la variable. És molt important no oblidar l'ordre `end` ja que la iteració no s'executarà fins que premem Retorn després d'haver-la escrit.

Si volem crear un vector que contingui tots els valors la successió fins a un determinat enter hem de crear un vector que vagi recollint els termes de la successió a mesura que els anem calculant. Seguint el mateix exemple,

```
>> N=100 % nombre d'iterats
N =  100
>> x(1)=0; % x recollira les dades i fem que x(1)=0
>> for i=2:N
x(i)=cos(x(i-1));
end
plot(x,'o--')
```

on en la darrera instrucció hem dibuixat els termes de la successió x_n com a funció de n i veiem que tendeixen cap a un valor aproximadament igual a `x(N)= 0.739085133215161`, i que podem veure a la figura 9.1.

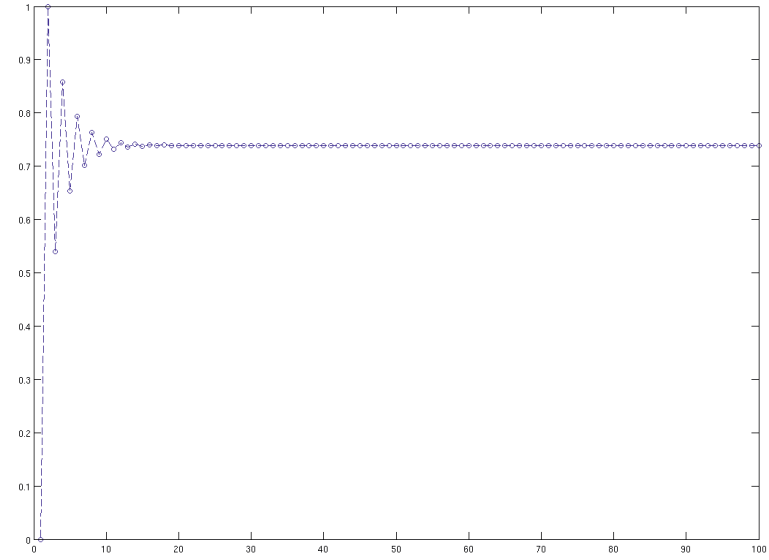


Fig. 9.1
Il·lustració de la convergència de la successió recurrent $x_{n+1} = \cos x_n$ amb $x_1 = 0$.

Si volem que, en comptes de començar en 0, la successió comenci en un altre valor, per exemple 0.3, podem fer que el vector inicial sigui igual a 0.3:

```
N=100; % nombre d'iterats
x(1)=0.3
for i=2:N
x(i)=cos(x(i-1));
end
```

Quan fem iteracions pot resultar un pèl feixuc haver d'escriure totes les ordres a la línia d'instruccions i haver de repetir-ho tot quan ens equivoquem. Podem editar un fitxer `.m`

que inclogui la iteració i cridar-lo des de la línia d'ordres com si fos una funció. Per exemple, fem `edit 'iteracio.m'` i hi posem el següent

```
function x=iteracio
N =  100;
x(1)=0.3;
for i=2:N
  x(i)=cos(x(i-1));
end
```

En cridar-la per línia d'ordres, veurem el resultat del vector `x` amb els elements de la iteració.

Podem fer també iteracions de segon ordre, en què el valor x_n depèn dels dos valors anteriors. Un exemple molt conegut és el de la successió recurrent de Fibonacci, que s'escriu com

$$f_n = f_{n-1} + f_{n-2}, \quad f_1 = 1 \text{ i } f_2 = 1.$$

i que obtenim escrivint a la consola les ordres següents:

```
N =  10;
fib(1)=1;
fib(2)=1; % f_1 i f_2 valen 1
for i=3:N
  fib(i)=fib(i-1)+fib(i-2);
end
disp(fib)
```

Com abans, també ho podríem escriure com una funció que ens retorna la successió de Fibonacci fins a un N arbitrari, que es rep com a argument. Fem `edit 'fibonacci.m'` i al fitxer `'fibonacci.m'` hi posem

```
% Retorna la successió de Fibonacci fins a N
function fib = fibonacci(N)
fib(1)=1;
fib(2)=1;
for i=3:N
  fib(i)=fib(i-1)+fib(i-2);
end
```

que podem cridar senzillament amb

```
>> fibonacci(10)
ans =
     1     1     2     3     5     8    13    21    34    55
```

Finalment, observem que el procediment que hem descrit ens permet resoldre problemes de valors inicials per a equacions en diferències. Per exemple, si tenim el problema de valors inicials següent:

$$x_{n+2} + \left(1 + \cos(\sqrt{2}\pi n)\right)x_{n+1} + x_n = 0, \qquad x(1) = 1,\ x(2) = 0$$

podem passar-lo a una recurrència de segon ordre aïllant el terme d'ordre més gran

$$x_{n+2} = -\left(1 + \cos(\sqrt{2}\pi n)\right)x_{n+1} - x_n, \qquad x(1) = 1,\ x(2) = 0$$

amb la qual cosa podem conèixer de forma recurrent x_n per a qualsevol n finit. Per exemple, per calcular el terme x_{100}, faríem:

```
>  x(1)=1; x(2)=0; % condicions inicials
>  for n=3:100
x(n)=-(1+cos(sqrt(2)*pi*(n-2)))*x(n-1)-x(n-2);
end
>  x(100)
ans =
     -2991630401.97087
```

9.2. Sumes de sèries

Un cas molt particular de successions recurrents el trobem en les sèries,

$$S_N = \sum_{n=1}^{N} x_n$$

que es poden escriure com una successió recurrent fent

$$S_n = S_{n-1} + x_n, \quad S_1 = x_1.$$

En el cas que coneguem una expressió tancada per als termes de la sèries, és millor usar l'ordre `sum()` que ens dóna la suma de les components d'un vector. Per exemple, considerem la suma

$$S_N = \sum_{n=1}^{N} \frac{1}{n^2}$$

de la qual se sap que

$$\lim_{N\to\infty} S_N = \sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}.$$

Calculem ara S_N definint un vector:

```
>> N =  1000
N =
        1000
>> indexs=1:N; x=1./(indexs.^2);
>> suma = sum(x)
suma =
   1.643934566681561
>> suma - pi^2/6
ans =
    -9.995001666649461e-04
```

Si ho volem fer amb un `for`, podem fer la iteració següent:

```
suma=1; % primer terme
for i=2:N
  suma=suma + 1/(i^2);
end
```

que ens dóna com a resultat:

```
suma =
   1.643934566681561
```

el mateix que amb la forma recursiva d'abans.

9.2.1 Complement opcional: Temps d'execució

A mesura que augmentem la N, per exemple $N = 10^6$, veurem que el mètode amb la iteració és molt més lent que el mètode vectorial

```
>> suma=1; N=1e6;
>> t= cputime;for i=2:N;suma=suma + 1/(i^2);end; temps=cputime-t
temps = 0.699999999999999
>> t= cputime; N=1e6;suma= sum(1./(1:N).^2); temps=cputime-t
temps = 0.060000000000002
```

Això il·lustra un dels desavantatges principals dels llenguatges interpretats (com `Matlab`, `Octave`, `Python`,...) respecte dels llenguatges compilats (com el `c`). En aquest cas però, també, hem de tenir en compte la memòria de què disposa el nostre ordinador i que pot fer que no puguem emmagatzemar vectors molt grans (proveu de fer `zeros(100000)`).

9.3. Complement opcional: Condicionals

Un altre dels elements importants amb vista a modificar el flux d'execució d'un conjunt de sentències és l'ús de condicionals a través de `if`, `else` i `elseif`. L'esquema bàsic és el següent:

```
if condicio
  ordres si es certa la condicio
else
  ordres si es falsa la condicio
end
```

Per exemple, la funció `signe.m`, definida a continuació ens retornarà un 1 si l'argument és positiu o zero i un -1 si és negatiu:

```
% funcio signe.m
function valor = signe(x)
```

```
if x >= 0
  valor = 1;
else
  valor = -1;
end
```

Si volem que ens retorni 0 en el cas en què `x` sigui exactament zero, la modificarem de la manera següent:

```
% funcio signe0.m
function valor = signe0(x)
if x > 0
  valor = 1;
elseif x<0
  valor = -1
else
  valor = 0;
end
```

on veiem que l'efecte de l'ordre `elseif()` és afegir una condició quan no es compleix la condició de l'ordre `if` anterior.

9.4. Exercicis

9.1. Considereu la successió recurrent següent:

$$x_{k+1} = 1 - \frac{1}{2}x_k^2, \quad x_1 = 0$$

Quant val x_{30}?

Resposta: 0.732095372376809.

9.2. Indiqueu quant val la suma següent:

$$\sum_{n=1}^{100} \frac{\cos(n)}{n^3}$$

Resposta: 0.44857294750852.

9.3. Donada la successió recurrent

$$\begin{aligned} x_{n+1} &= 3.8x_n(1-x_n) \\ x_1 &= 0.3 \end{aligned}$$

quant val el terme x_{10}?

Resposta: 0.565676868266769.

9.4. Calculeu el valor de la suma següent:

$$\sum_{k=0}^{100} \frac{(-1)^k}{(2k+1)^3}$$

Resposta: 0.968946206912337.

9.5. Esquema de la pràctica

- Els iteradors en `Matlab/Octave` s'escriuen mitjançant l'expressió `for variable =rang` i s'acaben amb la paraula `end`. Per exemple:

```
x=0.3;
for i=2:10
  x=cos(x);
end
```

  calcula x_{10}, on $x_{n+1} = \cos(x_n)$ i $x_1 = 0.3$. Si volem guardar en un vector tots els $x_1,\ldots,x_N$ farem:

```
N =  10;
x(1)=0.3;
for i=2:N
  x(i)=cos(x(i-1));
end
```

  Quan un terme depèn dels dos anteriors, com, per exemple, la successió de Fibonacci definida per $x_{n+2} = x_{n+1} + x_n$ amb $x_1 = x_2 = 1$, farem

```
N =  10;
fib(1)=1;
fib(2)=1;
for i=3:N
  fib(i)=fib(i-1)+fib(i-2);
end
```

- Per sumar els termes d'una sèrie $\sum_{k=1}^{N} a_n$ fins a N podem definir els a_n en un vector `x` a partir de `1:N` i fer `sum(x)`. Per exemple, per fer $\sum_{n=1}^{100} 1/n^2$ podem fer:

```
indexos=1:N;
x=1./(indexos.^2);
sum(x)
```

- Condicionals (opcional). Es representen de la forma:

```
if condicio
   comandes si es certa la condicio
else
   comandes si es falsa la condicio
end
```

  Per exemple, per definir la funció `signe.m` faríem:

```
% funcio signe.m
function valor = signe(x)
if x >= 0
   valor = 1;
else
   valor = -1;
end
```

Valors i vectors propis

El càlcul de valors i vectors propis és un dels problemes que requereixen més l'ús d'alguna eina de càlcul qual la dimensió de les matrius és gran. En aquesta pràctica, aprenem a trobar el polinomi característic d'una matriu i a calcular-ne els valors i vectors propis fent ús de les rutines de `Matlab/Octave`. Acabarem aplicant-ho a la diagonalització de matrius.

10.1. Polinomis característics i valors propis

Els valors propis d'una matriu quadrada A, és a dir, els valors $\lambda \in \mathbb{C}$ en què $A - \lambda I$ no és invertible, són les arrels del polinomi característic. A `Matlab/Octave`, calculem el polinomi característic a través de l'ordre `poly(A)`, que ens donarà els coeficients del polinomi $p(x) = \det(xI - A)$. A tall d'exemple, considerem la matriu

```
>> v=[1, 1,10,100, 10,1, 1]; A=compan(v)
A =
    -1   -10  -100   -10    -1    -1
     1     0     0     0     0     0
     0     1     0     0     0     0
     0     0     1     0     0     0
     0     0     0     1     0     0
     0     0     0     0     1     0
```

que té com a polinomi característic precisament `v` ja que n'és la *matriu companion*

```
>> polcar=poly(A) % polinomi caracteristic de A
polcar =
    1.0000   1.0000  10.0000  100.0000  10.0000   1.0000   1.0000
```

que recordem que representa el polinomi

$$p(x) = x^6 + x^5 + 10x^4 + 100x^3 + 10x^2 + x + 1$$

i trobem les seves arrels a través de la funció `roots()`:

```
>> format long
>> arrels = roots(polcar) % arrels del polinomi caracteristic
arrels =
    1.642706139564033 + 4.577461930900899i
    1.642706139564033 - 4.577461930900899i
    -4.185394608959343
    -0.238926097400560
    0.069454213615920 + 0.193536756885749i
    0.069454213615920 - 0.193536756885749i
```

on observem que n'hi ha dues de reals i les quatre restants són complexes.

Haver de calcular primer el polinomi característic i després les arrels és poc eficient i, per tant, hem d'usar la funció específica que `Matlab/Octave` té per al càlcul dels valors propis, que és `eig(A)` (dels *eigenvalues*):

```
>> vaps=eig(A) % valors propis de A
vaps =
  1.642706139564033 + 4.577461930900899i
  1.642706139564033 - 4.577461930900899i
  -4.185394608959342
  -0.238926097400560
  0.069454213615920 + 0.193536756885749i
  0.069454213615920 - 0.193536756885749i
```

que ens dóna el mateix resultat. Podem comprovar que la condició sobre el determinant que defineix el polinomi característic, tant per a un valor propi:

```
>> det(A-vaps(1)*eye(6))
ans =
  -3.6484e-12 - 2.3322e-11i
```

com per a tots els valors propis usant un iterador

```
>> for i=1:6
  det(A-vaps(i)*eye(6))
  end
ans =
     -3.648428024881165e-12 - 2.332158727803473e-11i
ans =
     -3.648428024881165e-12 + 2.332158727803473e-11i
ans =
    -9.482436049068880e-13
ans =
    -5.082090240683109e-16
ans =
     -1.574599496269894e-15 + 1.304001776077714e-16i
ans =
     -1.574599496269894e-15 - 1.304001776077714e-16i
```

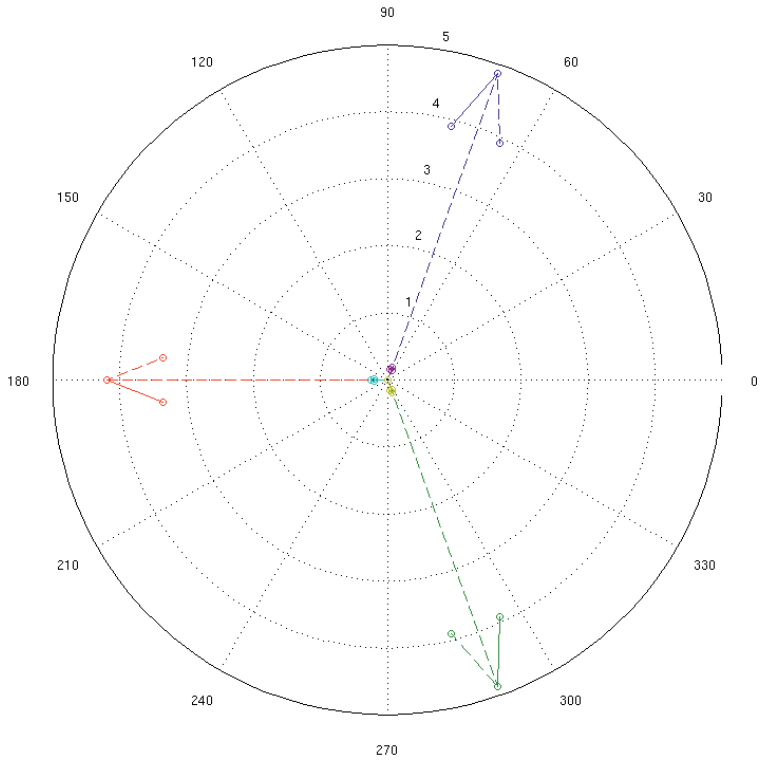


Fig. 10.1
Gràfica dels valors propis de la matriu companion associada al polinomi $p(x) = x^6 + x^5 + 10x^4 + 100x^3 + 10x^2 + x + 1$ usant l'ordre compass(vaps) en Matlab.

Per fer-nos-en una idea, en podem dibuixar els valors propis

```
>> compass(vaps)
```

Podem comprovar que el seu producte val 1 (per què?)

```
>> prod(vaps)
ans =   1.0000
```

i que els que són complexos apareixen per parelles conjugades

```
>> vaps(1:2:6)
ans =
   1.642706139564033 + 4.577461930900899i
  -4.185394608959342
   0.069454213615920 + 0.193536756885749i
>> vaps(2:2:6)
ans =
   1.642706139564033 - 4.577461930900899i
  -0.238926097400560
   0.069454213615920 - 0.193536756885749i
```

Per veure quin és el que té major norma

```
>> abs(vaps) % valor absolut de cada element
ans =
   4.863295301522246
   4.863295301522246
   4.185394608959342
   0.238926097400560
   0.205621895854647
   0.205621895854647
```

faríem

```
>> [maxim,imaxim]=max(abs(vaps)) % maxim i la seva posicio
maxim =
   4.863295301522246
imaxim =
     1
```

en aquest cas, l'argument `imax` ens dóna l'índex del primer element que assoleix el valor màxim del valor absolut de les components del vector.

10.2. Espais propis i diagonalització de matrius

Per trobar l'espai propi associat a un valor propi λ, podríem intentar calcular el nucli de $A - \lambda I$, per exemple per al valor propi `vap(4)`,

```
>> format short % per veure be els resultats
>> vep4 = null(A-vaps(4)*eye(6))
vep4 =
   -0.0008
    0.0032
   -0.0132
    0.0554
   -0.2320
    0.9710
```

i comprovar que és efectivament el vector propi

```
>> A*vep4-vaps(4)*vep4
ans =
   1.0e-15 *

   -0.2623
   -0.1415
    0.0056
    0.0017
   -0.0347
    0.0555
```

ja que és un valor molt proper a zero. Per cert, observem que

```
>> vep4(1:5)./vep4(2:6)
```

```
ans =
   -0.2389
   -0.2389
   -0.2389
   -0.2389
   -0.2389
>> vaps(4)
ans =
   -0.2389
```

la qual cosa indica que un vector propi associat al valor propi `vap(4)` ve donat per les potències d'aquest valor propi per a aquest tipus de matriu.

A la pràctica, el que farem és usar que la funció `eig()` que hem vist abans ens pot donar opcionalment també els vectors propis

```
>> [V,D]=eig(A) % V vectors propis en cols. D=diagonal de vaps.
V =
0.9786           0.9786           0.9710  -0.0008   0.0004-0.0001i
  0.0004+0.0001i
0.0680-0.1894i   0.0680+0.1894i  -0.2320   0.0032   0.0003-0.0017i
  0.0003+0.0017i
-0.0319-0.0263i -0.0319+0.0263i  0.0554  -0.0132  -0.0073-0.0044i
  -0.0073+0.0044i
-0.0073+0.0044i -0.0073-0.0044i -0.0132   0.0554  -0.0319+0.0263i
  -0.0319-0.0263i
0.0003+0.0017i  0.0003-0.0017i   0.0032  -0.2320   0.0680+0.1894i
  0.0680-0.1894i
0.0004+0.0001i  0.0004-0.0001i  -0.0008   0.9710   0.9786
0.9786

D =
1.6427+4.5775i     0             0           0     0              0
     0           1.6427-4.5775i  0           0     0              0
     0              0           -4.1854      0     0              0
     0              0             0      -0.2389   0              0
     0              0             0           0   0.0695+0.1935i   0
     0              0             0           0     0       0.0695-0.1935i
```

on ara `V` conté com a columnes els vectors propis de la matriu A i D és una matriu diagonal amb els valors propis com a elements a la diagonal. Comprovem, en primer lloc, que la diagonal de `D` és el que obtindríem amb l'ordre simple:

```
>> diag(D)-eig(A) % val 0 si [V,D]=eig(A)
ans =
   0
   0
   0
   0
   0
   0
```

i que les columnes de V són els vectors propis:

```
>> vaps=diag(D)
vaps =
   1.6427 + 4.5775i
   1.6427 - 4.5775i
  -4.1854
  -0.2389
   0.0695 + 0.1935i
   0.0695 - 0.1935i
>> A*V(:,1)-vaps(1)*V(:,1) % comprovacio vaps i veps
ans =
   1.0e-14 *
  -0.1998 + 0.2665i
   0.0777 - 0.0444i
   0.0167 + 0.0111i
  -0.0007 - 0.0042i
  -0.0015 + 0.0011i
  -0.0001 - 0.0004i
```

i així amb la resta de vectors propis. Si volem expressar aquesta condició de valors i vectors propis de manera més compacta, en format matricial, només cal que comprovem

$$AV = VD$$

que és equivalent a la condició de vector i valor propis quan la matriu `V` és no singular, ja que aleshores:

$$A = VDV^{-1}$$

i D és una matriu diagonal. Comprovem-ho

```
>> A*V-V*D % comprovacio
```

que dóna una matriu de nombres petits (i que omitirem aquí). Per comprovar que tots els elements d'aquesta matriu són efectivament petits fem

```
>> max(abs(A*V-V*D)) % maxim per columnes
ans =
   3.5527e-15   1.0413e-15   1.0413e-15   6.1776e-16
>> max(max(abs(A*V-V*D))) % maxim per columnes i files
ans =  3.5527e-15
```

Així doncs, el cas diagonalitzable el podem identificar amb $rang(V) = n$ (o, equivalentment, $\det(V) \neq 0$), on n és la dimensió de la matriu quadrada A. La matriu V ens dóna precisament el canvi de base que ens diagonalitza la matriu. En efecte, de la relació $AV = VD$ n'obtenim que $D = V^{-1}AV$, i així V, quan és invertible, és la matriu que proporciona el canvi. Comprovem-ho:

```
>> rank(V)
```

```
ans =
     6
>> inv(V)*A*V-D % matriu de nombres propers a zero
ans =
   1.0e-14 *
...
>> max(abs(inv(V)*A*V-D))
ans =
   1.0e-14 *
    0.4529    0.4578    0.4378    0.0888    0.1226    0.1228
```

En cas que un valor propi tingui multiplicitat superior a 1, aquest apareixerà repetit, tant a l'ordre simple `eig(A)` com a la diagonal de `D` amb `[V,D]=eig(A)`. En aquest cas, no podrem assegurar que totes les columnes siguin vectors propis independents, cosa que detectem calculant el rang de la matriu V.

Veiem primer un exemple on no diagonalitza:

```
>> A=[-3,0,-4;-1,-1,-1;1,0,1]
A =
  -3   0  -4
  -1  -1  -1
   1   0   1
>> [V,D]=eig(A)
V =
   0.00000  -0.00000  -0.00000
   1.00000   1.00000  -1.00000
   0.00000   0.00000   0.00000
D =
  -1.00000   0.00000   0.00000
   0.00000  -1.00000   0.00000
   0.00000   0.00000  -1.00000
```

d'on veiem que $\lambda = -1$ és un valor propi de multiplicitat 3, però que, en canvi, només té un únic vector propi linealment independent.

```
>> rank(V)
ans =  1
```

que és $(0,1,0)$, com qualsevol columna de V. Això significa que la matriu no diagonalitza.

Considerem un altre exemple,

```
>> A=[51   4 -40   9;38 115  10 -24;-18   2  67 -10;4   6  -2  28]
A =
    51     4   -40     9
    38   115    10   -24
   -18     2    67   -10
     4     6    -2    28
>> format short
```

```
>> eig(A)
ans =
   29.0000
   87.0000
  116.0000
   29.0000
```

d'on veiem que 29 és un valor propi doble. Vegem ara quants vectors propis ens troba la rutina:

```
>> [V,D]=eig(A)
V =
    0.8149   -0.5774   -0.0711    0.1892
   -0.4220    0.5774   -0.9949    0.1347
    0.3929    0.5774   -0.0000    0.3239
   -0.0582   -0.0000   -0.0711    0.9171
D =
   29.0000         0         0         0
         0   87.0000         0         0
         0         0  116.0000         0
         0         0         0   29.0000
>> rank(V)
ans =
     4
```

la qual cosa vol dir que totes les columnes són linealment independents i corresponen a vectors propis linealment independents. Podem comprovar que A diagonalitza:

```
>> inv(V)*A*V-D
ans =
   1.0e-13 *
    0.1421   -0.0589    0.1144         0
    0.1243    0.7105    0.2287         0
    0.1088    0.6317    0.1421   -0.0711
   -0.0111    0.0524   -0.0755         0
```

10.2.1 Complement: Canvis de base

Acabem de veure que l'ús de la notació vectorial és particularment útil en el canvi de base, no necessàriament associat a la diagonalització. Vegem ara un altre exemple de canvi de base. Considerem la matriu A següent, que representa una aplicació lineal $F : \mathbb{R}^6 \to \mathbb{R}^8$:

$$A = \begin{pmatrix} 9 & 4 & 1 & 6 & 12 & 7 \\ 4 & 0 & 4 & 15 & 1 & 14 \\ 7 & 0 & 7 & 8 & 10 & 9 \\ 16 & 0 & 16 & 3 & 13 & 2 \\ 0 & 2 & -4 & 0 & 0 & 0 \\ 0 & 6 & -12 & 0 & 0 & 0 \\ 9 & 0 & 9 & 6 & 12 & 7 \\ 5 & 0 & 5 & 10 & 8 & 11 \end{pmatrix}$$

Calculem la matriu de F en les noves bases de sortida $S = (e_1, e_2, e_3, e_4, e_1 + e_2 + e_5 - e_6, e_3 - e_5 - e_6)$ i d'arribada $B = (e_1 + e_7 + e_8, e_2, e_3, e_4, e_5, e_6, e_7, e_1 - e_8)$.

```
>> A=[9,4,1,6,12,7;4,0,4,15,1,14;7,0,7,8,10,9;16,0,16,3,13,2;...
        0,2,-4,0,0,0;0,6,-12,0,0,0;9,0,9,6,12,7;5,0,5,10,8,11]
A =
     9     4     1     6    12     7
     4     0     4    15     1    14
     7     0     7     8    10     9
    16     0    16     3    13     2
     0     2    -4     0     0     0
     0     6   -12     0     0     0
     9     0     9     6    12     7
     5     0     5    10     8    11
```

Construirem les matrius dels canvis B i S de manera que la nova matriu vindrà donada per $B^{-1}AS$ (comprovarem que B és invertible). Per no haver d'escriure tots els elements un a un, observem que tant una com l'altra són molt semblants a la identitat i només caldrà canviar-ne alguns elements concrets. Així

```
>> B=eye(8); % partim de B identitat 8x8
>> B(8,1)=1; B(1,8)=1; B(7,1)=1; B(8,8)=-1
>> B
B =
    1    0    0    0    0    0    0    1
    0    1    0    0    0    0    0    0
    0    0    1    0    0    0    0    0
    0    0    0    1    0    0    0    0
    0    0    0    0    1    0    0    0
    0    0    0    0    0    1    0    0
    1    0    0    0    0    0    1    0
    1    0    0    0    0    0    0   -1
>> S=eye(6); %identitat 6x6
>> S(:,5)=[1;1;0;0;1;-1]; % canviem la col. 5
>> S(:,6)=[0;0;1;0;-1;-1] % canviem la col. 6
S =
    1    0    0    0    1    0
    0    1    0    0    1    0
    0    0    1    0    0    1
    0    0    0    1    0    0
    0    0    0    0    1   -1
    0    0    0    0   -1   -1
```

i la matriu en el nou sistema és:

```
>> C=inv(B)*A*S % tambe C= B\A*S
C =
     7     2     3     8    10   -16
     4     0     4    15    -9   -11
     7     0     7     8     8   -12
    16     0    16     3    27     1
     0     2    -4     0     2    -4
     0     6   -12     0     6   -12
     2    -2     6    -2     4     6
     2     2    -2    -2     8    -2
```

10.3. Exercicis

10.1. Donada la matriu

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 1 & 1 & 1 \\ 3 & 1 & 1 & 1 & 1 \\ 4 & 1 & 1 & 1 & 1 \\ 5 & 1 & 1 & 1 & 1 \\ 6 & 1 & 1 & 1 & 1 \\ 7 & 1 & 1 & 1 & 1 \end{pmatrix}$$

doneu el valor propi real més gran del producte $A^T A$.

Resposta: 176.667471788751.

10.2. D'entre els valors propis reals de la matriu següent, digueu quin és el més gran:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

Resposta: 3.4494897427831...

10.3. Per a qualsevol n natural, definim la matriu A_n com

$$A_n = \begin{pmatrix} 1 & 1 & 0 & \dots & 0 & 0 \\ 1 & 2 & 1 & \dots & 0 & 0 \\ 0 & 1 & 3 & \dots & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & n-1 & 1 \\ 0 & 0 & 0 & \dots & 1 & n \end{pmatrix}$$

que és simètrica, té 1 per sobre i per sota de la diagonal principal i la seva diagonal val $1, 2, 3, \dots, n$. Quant val el valor propi de mòdul més gran? Trobeu-lo per a A_{10} i A_{100}.

10.4. Esquema de la pràctica

- Polinomi característic d'una matriu quadrada `A` ($p(x) = \det(xI - A)$): `poly(A)`.
- Valors propis d'una matriu quadrada `A`, `vaps=eig(A)`, on `vaps` és un vector amb els valors propis de la matriu repetits segons multiplicitat.
- Valors i vectors propis d'una matriu quadrada `A`:

```
[V,D]=eig(A)
```

  on

  - `D` és una matriu diagonal amb els valors propis com a elements a la diagonal. És el mateix que `diag(eig(A))`.
  - `V` és una matriu que compleix $AV = VD$. En cas que sigui invertible (A diagonalitzable), compleix que $A = VDV^{-1}$ i les columnes de `V` són els vectors propis de la matriu `A`.

→ Apèndix A

Com fer-ho en Python?

Tant Matlab com Octave s'utilitzen àmpliament en el món de l'enginyeria. Tanmateix, l'ús de llenguatges de propòsit general, en especial Python [5], és cada cop més habitual en aquest àmbit. En el cas de Python, els avantatges són clars: es tracta d'un llenguatge de programació molt establert, clar, amb moltíssimes extensions o mòduls de gran qualitat i que s'integra bé en grans projectes de *software*.

Tot i que es podrien programar directament els mètodes que hem presentat en Python pur, és molt més còmode i eficient fer-ho usant mòduls desenvolupats específicament per a aquest ús, que faciliten la feina i acceleren considerablement els procediments. En aquest apèndix explicarem breuement com usar l'entorn Python amb el mòdul Pylab per fer les pràctiques del taller de matemàtiques. Suposem que es tenen uns coneixements bàsics de Python (podeu usar el llibre [3] com a referència) i que s'han seguit les pràctiques en Matlab/Octave.

A.1. Configuració de l'entorn

El mòdul Pylab, el podem instal·lar en la majoria de sistemes operatius i és, a grans trets, un paraigües que crida tres grans mòduls, que també podem fer separadament

- Numpy [7]: un mòdul de Python que defineix els tipus de vectors numèrics i operacions matemàtiques, semblantment a Matlab/Octave.
- Scipy [9]: un altre mòdul que estén Numpy per a problemes matemàtics d'optimització, integració, àlgebra lineal ...
- Matplotlib [6]: un mòdul per a la representació gràfica de dades en una sintaxi semblant a Matlab/Octave.

La millor manera d'instal·lar aquests paquets en GNU/Linux és instal·lant aquests tres paquets des de l'administrador de paquets (en Debian/Ubuntu aquests són python-numpy, python-scipy i python-matplotlib), així com els paquets suggerits. També és possible instal·lar-se una distribució de Python orientada a l'enginyeria, com, per exemple, Python(x,y), que inclogui per defecte aquests paquets. Aquesta darrera opció és també la que us recomanem per a MS/Windows.

Tot i que es poden importar separadament en Python, en aquest apèndix ens centrarem en l'ús de Python com a consola interactiva alternativa a Matlab/Octave. És per això que, des de la línia d'ordres, cridarem

```
ipython -pylab -classic
```

que ens engegarà una consola IPython, que és una consola Python modificada que ens permet recuperar ordres amb les fletxes cap amunt, autocompletació amb tabulador, ...i que s'instal·la per defecte amb els altres paquets anteriors. L'opció -pylab ens carrega els mòduls numèrics i gràfics que hem esmentat abans i l'opció -classic, tot i que no és necessària, la posem perquè s'assembli més a una consola Python clàssica. Tot i que és millor cridar directament els mòduls de Phyton que fem servir un cop dins de la consola, usarem aquí aquestes opcions per comoditat. Anem a veure molt breument algunes de les possibilitats que ens ofereix per al càlcul numèric. Per a aprendre més del seu ús en enginyeria i càlcul científic us recomanem els llibres [10] i [11].

A.2. Ús com a calculadora científica

Un cop engeguem IPython, se'ns informa de la versió i del maneig bàsic de la consola interactiva. Es tracta d'una consola Python; per tant, podem fer el mateix que en una consola "clàssica":

```
Python 2.6.5
Type "copyright", "credits" or "license" for more information.

IPython 0.10 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object'. ?object also works,

  Welcome to pylab, a matplotlib-based Python environment.
  For more information, type 'help(pylab)'.
>>> 2+2
4
>>> (1+2*3+4**5 -6)/(7*8) # incorrecte: resultat enter
18
```

on observem que el caràcter de comentari és # i que a^b s'expressa com a**b. Molts mòduls matemàtics estan ja cridats i, per tant, podem usar-los directament sense haver d'importar cap mòdul. Si ho comparem amb el resultat de Matlab/Octave, veurem que el resultat no és correcte, ja que dóna 18.3035714285714. Això és degut al fet que el numerador i el denominador són enters (tipus int) i entén que el resultat ha de ser un enter i no pas un nombre real (float) com esperaríem. Per tant, cal indicar-li que o bé el numerador o bé el denominador són flotants:

```
>>> (1+2*3+4**5-6.0)/(7*8) # numerador flotant
18.303571428571427
>>> (1+2*3+4**5-6)/float(7*8) # denominador flotant
18.303571428571427
```

Els noms de les funcions matemàtiques són molt semblants:

```
>>> cos(pi/3) # pi es una constant predefinida
0.50000000000000011
>>> exp(1)-e # e es exp(1)
0.0
>>> pow(10,log10(2)) # pow() =power() en Matlab/Octave
2.0
>>> 5.2e7 # la notacio cientifica es valida
52000000.0
```

Com és propi de Python, els nombres complexos s'indiquen en forma binomial de la forma `a+bj`, essent `a` la part real i `b` la part imaginària. Un cop l'hem definida, podem accedir a la part real i a la part imaginària de l'estructura a través de l'operador `.` (punt):

```
>>> z=3+2j
>>> z.real # part real
3.0
>>> z.imag # part imaginaria
2.0
>>> abs(z) # modul
3.6055512754639891
```

Per accedir a l'ajuda de les funcions, n'hi ha prou a posar un interrogant al final de la funció sobre la qual volem informar-nos o bé fer `help(ordre)`

```
>>> floor?
>>> help(floor)
```

Finalment, per sortir de la consola n'hi ha prou amb fer `quit()` o `exit()`.

A.3. Vectors i polinomis

L'entorn de Python i, concretament, el seu mòdul Numpy, defineix els vectors a través del tipus `array()`, que pren com a argument un element iterable (com pot ser una llista o una tupla en Python) i el converteix en un tipus numèric adequat per a representar vectors:

```
>>> llista=[1,2,3,1]
>>> tupla=(5,6,7,5)
>>> v=array(llista)
>>> v
array([1, 2, 3, 1])
>>> v[0] # primer element
1
>>> v[-1] # darrer element
1
>>> v[4] # fora de rang - Error
------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython console>", line 1, in <module>
IndexError: index out of bounds
```

```
>>> u=array(tupla)
>>> u
array([5, 6, 7, 5])
```

i, declarat com a `array`, podem manipular-lo de manera molt semblant a `Matlab/Octave`, tenint en compte que els índexs en `Python` comencen en `0` i no pas en `1` com en `Matlab/Octave`.

```
>>> cos(v) # operacio element a element, diferent de Matlab/Octave
array([ 0.54030231, -0.41614684, -0.9899925 ,  0.54030231])
>>> v**2 # el quadrat element a element, diferent de Matlab/Octave
array([1, 4, 9, 1])
>>> v+1 # suma 1 a tots els elements, com Matlab/Octave
array([3, 3, 4, 2]
>>> v*u # * entre arrays vectors es el producte element a element
array([ 5, 12, 21,  5])
>>> dot(v,u) # producte escalar, el mateix que sum(v*u)
43
>>> v[1:3] # accedim a subconjunts de v
array([2, 3])
```

Un cop tenim declarat un `array`, per exemple `v`, fent `v.<TABULADOR>` veurem quins mètodes se li apliquen, com per exemple

```
>>> u.sum() # el mateix que sum(u)
6
>>> u.cumsum() # el mateix que cumsum(u)
array([1, 3, 6])
>>> u.prod() # el mateix que prod(u)
6
>>> u.cumprod() # el mateix que cumprod(u)
array([1, 2, 6])
```

i molts d'altres. Podem assignar elements per valor

```
>>> v[0]=2
>>> v
array([2, 2, 3, 1])
```

però cal anar amb compte a copiar directament els *arrays*

```
>>> v # recordem el valor de v
array([2, 2, 3, 1])
>>> w=v # NO ho farem per a arrays
>>> w # el contingut de v i w es igual
array([2, 2, 3, 1])
>>> w[0]=3 # modifiquem el valor de w[0]
>>> w # w s'actualitza correctament
array([3, 2, 3, 1])
>>> v # ATENCIO:  v tambe es modifica
array([3, 2, 3, 1])
```

Això s'explica pel fet que les *arrays* són objectes mutables (com les llistes), així que la instrucció `w=v` no resulta en un nou objecte, sinó que senzillament crea una nova referència a `v`. Per tal de fer una còpia independent d'una `array`, cal usar el mètode `copy()` dels objectes `array` com segueix

```
>>> v=array([1,2,3,1])
>>> w=v.copy() # el mateix que w=copy(v)
>>> v
array([1, 2, 3, 1])
>>> w[0]=2
>>> v
array([1, 2, 3, 1])
>>> w
array([2, 2, 3, 1])
```

En definitiva, podem pensar que les *arrays* es comporten de manera molt semblant al tipus `list` de `Python`, tot i que amb algunes modificacions, com ara la concatenació, que no té el símbol `+`, reservat per a la suma

```
>>> [1,2,3]+[4,5,6] # suma de llistes= concatenacio
[1, 2, 3, 4, 5, 6]
>>> u=array([1,2,3]) # definim les arrays
>>> v=array([4,5,6])
>>> u+v # suma d'arrays= suma de vectors
array([5, 7, 9])
>>> concatenate((u,v)) # (u,v,...) vectors a concatenar
array([1, 2, 3, 4, 5, 6])
```

A.3.1 Polinomis

Quant als polinomis, `Python/Pylab` té una sintaxi molt semblant a `Matlab/Octave` per tal com s'accepta que qualsevol iterable, ja sigui una llista, una tupla o una `array`, representa els coeficients d'un polinomi ordenats de major a menor grau. Així, per exemple, per representar el polinomi

$$p(x) = x^7 + 3x^2 - 1,$$

avaluar-lo en 0.5 i calcular-ne les arrels fem

```
>>> p= [1,0,0,0,0,3,0,-1] # coeficients del polinomi
>>> polyval(p,0.5) #l'avaluem en 0.5
-0.2421875
>>> polyval(array(p),0.5) # polyval pren arrays, llistes o tuples
-0.2421875
>>> arrels=roots(p); arrels #; dues instruccions en una linia
array([ 0.96979381+0.77159912j,  0.96979381-0.77159912j,
       -0.37392903+1.23052948j, -0.37392903-1.23052948j,
        0.57156842+0.j,-1.17929795+0.j, -0.58400002+0.j])
>>> abs(arrels) # calculem els moduls dels vectors.
array([ 1.2393003 ,  1.2393003 ,  1.28608932,  1.28608932,  0.57156842,
        1.17929795,  0.58400002])
```

Recordem que la multiplicació de polinomis no és la multiplicació dels vectors de coeficients element a element (cosa que correspondria al producte `p*q` per a `p` i `q` `array` que representin els coeficients), sinó que hem d'usar el producte de convolució

```
>>> p =[1 ,0 ,0 ,1]
>>> q =[1 , -2 ,2]
>>> convolve(p,q) # en Matlab/Octave seria conv()
array([ 1, -2,  2,  1, -2,  2])
```

o bé l'ordre `polymul()`, juntament amb altres de significat força evident

```
>>> polymul(p,q)
array([ 1, -2,  2,  1, -2,  2])
>>> polydiv(polymul(p,q),q) # quocient,divisio com a llista d'arrays
(array([ 1.,  0.,  0.,  1.]), array([ 0.]))
>>> quocient,residu=polydiv(polymul(p,q),q) # podem recollir-ho aixi
>>> quocient
array([ 1.,  0.,  0.,  1.])
>>> residu
array([ 0.])
>>> resultat=polydiv(polymul(p,q),q) # podem assignar-ho a una llista
>>> resultat[0] # primer element el quocient
array([ 1.,  0.,  0.,  1.])
>>> resultat[1] # segon element el residu
array([ 0.])
>>> quocient,residu=polydiv(polymul(p,q)+1,q) # divisio no exacta
>>> quocient,residu
(array([ 2.,  3.,  5.,  6.]), array([ 1., -9.]))
>>> polyder(polyint(q)) # derivem i integrem
array([ 1., -2.,  2.])
```

i altres funcions que es poden trobar a l'ajuda.

A.4. Representació gràfica

A.4.1 L'ordre `plot()`

El funcionament de l'ordre `plot` en l'entorn `Pylab` de `Python`, que està definida en el mòdul `matplotlib`, és molt semblant a `Matlab/Octave`. Hem de passar una llista (o `array`) de valors d'abscisses, una d'ordenades i una cadena de caràcters per al format. Per exemple, si fem

```
>>> x=[0,1,2,3,4]
>>> y=[1,2,4,1,1]
>>> plot(x,y)
```

ens apareixerà una finestra semblant a la gràfica de l'esquerra de la figura A.1, des d'on podrem exportar, ampliar o modificar la figura. Si fem un altre dibuix, com per exemple:

```
>>> plot(x,'o--') # els punts (0,0),(1,1),... units amb linies disc.
```

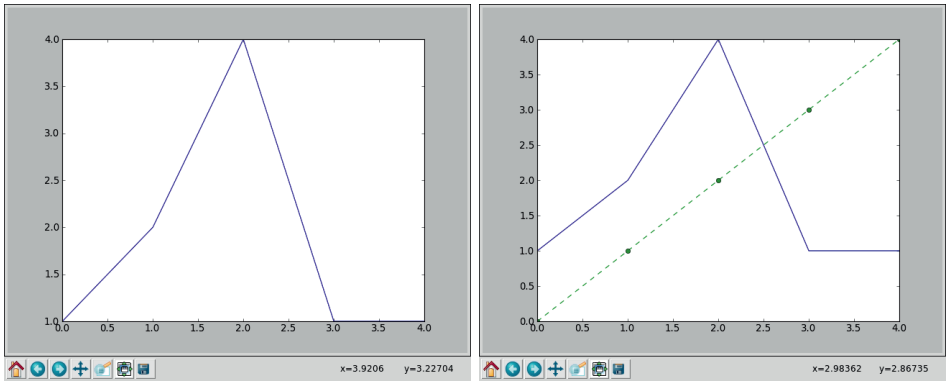


Fig. A.1
Exemples de la finestra gràfica del paquet Matplotlib amb Python/Pylab.

veurem, com s'indica a la gràfica de la dreta la figura 1, que el comportament per defecte és anar afegint les gràfiques amb colors diferents com a capes sobre una mateixa figura. Per deixar d'afegir elements, caldrà tancar la finestra o bé escriure, a la consola, l'ordre `close()` per tancar-la. Per tenir aquest comportament és important usar la consola `IPython` amb el mode `-pylab`, com hem indicat al començament, ja que si uséssim la consola `Python` normal i haguéssim cridat el paquet `Pylab` (fent, per exemple, `from pylab import *`) hauríem d'escriure l'ordre `show()` perquè es mostressin les ordres gràfiques que hi hem introduït fins aleshores.

A.4.2 Rangs de valors

Com en `Matlab/Octave`, amb vista a representar funcions és important saber definir rangs de valors (del tipus `inici:increment:condicio_parada` en `Matlab/Octave`). Això ho farem amb les instruccions `arange()` i `linspace()`

```
>>> arange(5) # comencem en 0. Incrementem en 1 mentre <5
array([0, 1, 2, 3, 4])
>>> arange(1,5) # comencem en 1. Incrementem en 1 mentre <5
array([1, 2, 3, 4])
>>> arange(1,5,0.5) # comencem en 1. Incrementem en 0.5 mentre <5
array([ 1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5])
>>> linspace(0,1,11) # 11 punts equiespaiats en [0,1]
array([ 0., 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.])
```

A diferència de `Matlab/Octave`, la condició de parada de l'ordre `arange()` és estricta.

A.4.3 Representació de funcions

Com hem vist abans, un dels avantatges de `Python/Pylab` és que, per defecte, les operacions indicades sobre vectors s'apliquen element a element i, per tant, dibuixar funcions és força natural. Així, per reproduir la figura 3.5 fem (no copiem el resultat a la consola de les ordres de dibuix):

```
>>> x =linspace(0,2,201) # definim les abscisses
>>> y1=1+x +0.5*x**2; y2 = exp (x) # definim les ordenades
```

```
>>> plot(x,y1,'r') # dibuixem la primera grafica
>>> plot(x,y2,'b') # dibuixem la segona grafica
>>> title('Aproximacio de Taylor de 2n ordre') # titol de la grafica
>>> xlabel('x') # etiqueta per a l'eix de les x
>>> ylabel('y') # etiqueta per a l'eix de les y
>>> legend(['1+x+0.5*x**2','exp(x)']) # es passa com una llista
```

i n'obtindrem com a resultat la figura A.2.

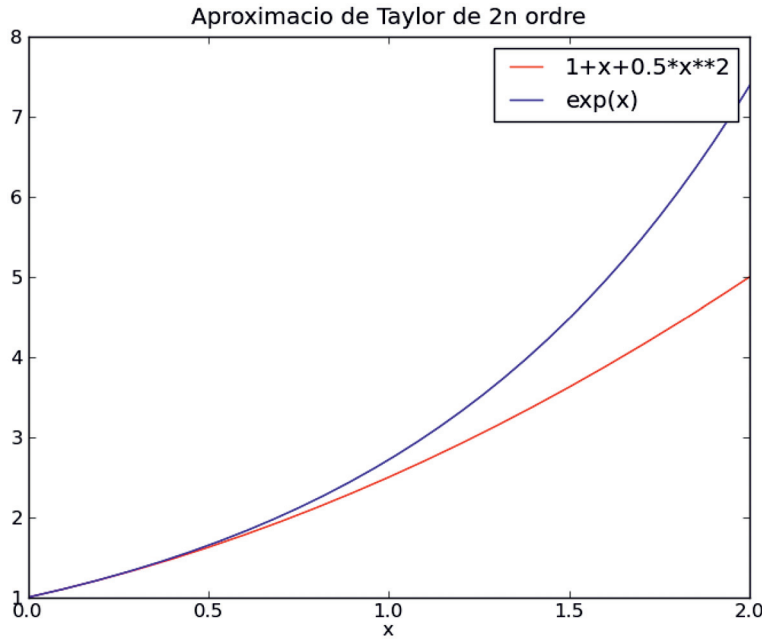


Fig. A.2 Exemple de figura amb gràfiques múltiples, llegenda, títol i etiquetes per als eixos.

A.4.4 Definició de funcions

Per definir una funció en `Python` perquè es pugui utilitzar en l'entorn `Pylab`, podrem usar la sintaxi habitual en `Python` a través de la paraula clau `def`. Així, per definir la funció $y = \cos(x^2)$, podríem escriure:

```
>>> def funcio(x):
...     return cos(x**2)
```

on, després d'introduir els dos punts, la consola `IPython` ja ens indentarà, amb quatre espais, la funció com correspon amb les funcions de `Python`. Prement Retorn, podem tancar la indentació. Si ho haguéssim definit en un fitxer, podríem carregar la funció a través de l'ordre `run nom_de_fitxer.py` (pròpia de `IPython`) o a través de l'ordre `execfile('nom_de_fitxer.py')` (general de `Python`). Un cop definida la funció la podem cridar des de la línia d'instruccions com qualsevol altra funció:

```
>>> funcio(sqrt(pi))
-1.0
```

```
>>> funcio(array([0,1,2])) # podem aplicar-ho a arrays
array([ 1.        ,  0.54030231, -0.65364362])
>>> funcio(array([sqrt(pi*x) for x in range(10)])) # exercici
array([ 1., -1.,  1., -1.,  1., -1.,  1., -1.,  1., -1.])
```

Recordem que no podem aplicar funcions directament a llistes sinó que, si ho necessitem, hem de fer servir l'ordre `map()`:

```
>>> map(funcio,[0,1,2])
[1.0, 0.54030230586813977, -0.65364362086361194]
```

Una altra possibilitat per definir funcions és fer-ho a través de les funcions anònimes o funcions `lambda`:

```
>>> funcio2=lambda x: cos(x**2)
>>> funcio2(sqrt(pi))
-1.0
>>> funcio2(arange(3))
array([ 1.        ,  0.54030231, -0.65364362])
```

A.5. Zeros i extrems de funcions

Les funcions de càlcul de zeros i de minimització es troben al submòdul `optimize` del mòdul `scipy` i, per tenir-hi accés, els hem d'importar; per exemple:

```
>>> from scipy.optimize import *
```

per importar totes les funcions, si bé és millor fer-ho només amb les que necessitem. Així, per al càlcul de zeros, el mòdul `scipy.optimize` ofereix moltes funcions de càlcul de zeros i minimització. Per exemple, el mètode de la bisecció està implementat com a `bisect` i un mètode semblant al que usen `Matlab` i `Octave` quan es crida la funció `fzero` el trobem a `brentq`. Tot seguit els provarem per trobar el zero de la funció

$$f(x) = e^x + 3x - x^2 - 2$$

a l'interval $[0,1]$. Importem el mòdul, definim la funció i cridem a `brentq`

```
>>> from scipy.optimize import brentq
>>> funcio=lambda x: exp(x) +3*x -x**2 -2 # definim la funcio anonima
>>> funcio(0)*funcio(1) # comprovem que hi ha un zero
-2.7182818284590446
>>> arrel=brentq(funcio,0,1) # calculem el zero. Similar a fzero()
>>> arrel,funcio(arrel) # comprovem el zero
(0.25753028543993162, 2.6778579353958776e-13)
```

De fet, no cal definir abans la funció (que també podríem haver definit de forma no anònima) sinó que podem usar les funcions anònimes directament dins de la rutina

```
>>> brentq(lambda x: exp(x) +3*x -x**2 -2,0,1) # funcio anonima
0.25753028543993162
```

Podem provar de modificar la precisió amb la rutina de la bisecció, `bisect`, amb diverses precisions:

```
>>> from scipy.optimize import bisect # importem biseccio
>>> bisect(lambda x: exp(x) +3*x -x**2 -2,0,1,xtol=0.5) # un pas
0.5
>>> eps=finfo(float64).eps # Determinem l'epsilon de la maquina
>>> bisect(lambda x: exp(x) +3*x -x**2 -2,0,1,xtol=eps) # tol=eps
0.2575302854398609
```

Per trobar mínims de funcions (i màxims, com vam veure a la pràctica 4), podem usar altres funcions del mateix mòdul. Concretament, si volem un algoritme semblant al que s'usa a `Matlab` i `Octave` en cridar la funció `fminbnd()`, usarem la funció `fminbound()`. Vegem-ho per a trobar el mínim de la funció $f(x) = \sin(\sin(x)+1)$ a l'interval $[1,2]$

```
>>> from scipy.optimize import fminbound
>>> fminbound(lambda x: sin(sin(x)+1),1,2)#definim la funcio anonima
array([ 1.57079658])
>>> fminbound(lambda x: sin(sin(x)+1),1,2,full_output=True)
(array([ 1.57079658]), array([ 0.90929743]), 0, 8)
>>> resultat= fminbound(lambda x: sin(sin(x)+1),1,2,full_output=True)
>>> resultat[0], resultat[1],sin(sin(resultat[0])+1)
(array([ 1.57079658]), array([ 0.90929743]), array([ 0.90929743]))
>>> fminbound(lambda x: sin(sin(x)+1),1,2,full_output=True,xtol=1e-8)
(array([ 1.57079633]), array([ 0.90929743]), 0, 9)
```

on en la darrera instrucció hem canviat la tolerància per defecte (`1e-5`) a `1e-8`.

A.6. Integració numèrica en `Python`/`Pylab`

Per a l'avaluació numèrica d'integrals, com per exemple

$$\int_0^1 e^{x^2} dx$$

els mètodes que hem vist són els compostos de trapezis, de Simpson i de quadratura adaptativa. Per als dos primers només, necessitem un vector de valors de `x` i `y` amb els abscisses i les ordenades de la funció en punts equiespaiats de l'interval d'integració. El mòdul `integrate` del paquet `scipy` conté les ordres `trapz` i `simps` amb els mètodes compostos de trapezis i de Simpson. Es criden de manera molt semblant a `Matlab`/`Octave`, però amb la diferència, important, que primer es posa la `y` i després la `x`:

```
>>> from scipy.integrate import trapz,simps
>>> funcio=lambda x: exp(x**2) # definim la funcio
>>> x=linspace(0,1,4+1) # considerem 4 intervals
>>> y=funcio(x) # definm les ordenades
>>> trapz(y,x) # trapezis en 4 intervals. Atencio ordre y,x
1.4906788616988553
```

```
>>> simps(y,x) # Simpson en 4 intervals. Atencio ordre y,x
1.4637107604455966
```

Per al mètode de quadratura adaptativa, cal que passem la funció a la rutina. Podem usar les rutines `quad` (que és la de propòsit més general) o `quadrature`, que donen resultats semblants a la rutina `quadl` en `Matlab` i `Octave`. Us animo a veure la documentació d'aquestes funcions per conèixer-ne els detalls i modificar les toleràncies:

```
>>> from scipy.integrate import quad,quadrature
>>> quadrature(funcio,0,1) # retorna  resultat i error
(1.462651745896296, 9.9209795934029898e-10)
>>> quadrature(funcio,0,1,tol=1e-6) # especifiquem una tolerancia
(1.462651744904198, 7.6885515509772517e-08)
>>> quad(funcio,0,1)
(1.4626517459071817, 1.6238696453143372e-14)
```

A.7. Control de flux

Els iteradors formen part de la sintaxi bàsica de `Python` i, per tant, podem usar-los a la consola `IPython` directament. Per exemple, per calcular el terme x_{10} de la successió recurrent $x_n = \cos(x_n)$ amb $x_0 = 1$, fem

```
>>> x=0
>>> for n in range(1,10):
...     x=cos(x)
>>> x
0.75041776176376052
```

si els volem guardar en una llista, l'hem d'anar augmentant amb l'ordre `append`

```
>>> x=[0]
>>> for n in range(1,10):
...     x.append(cos(x[n-1]))
>>> x
[0, 1.0, 0.54030230586813977,...]
>>> plot(x,'o--') # podem representar-ho
```

Per definir sèries, podem usar les mateixes tècniques que per a `Matlab/Octave`, però voldríem emfatitzar com n'és, de còmoda i eficient, la tècnica anomenada *list comprehension*. Així, per sumar la sèrie

$$\sum_{n=1}^{100} \frac{1}{n^2}$$

podríem fer, usant *list comprehension*:

```
>>> termes=[1.0/(n**2) for n in range(1,101)] # 1.0 volem divisio float
>>> sum(termes)
1.6349839001848923
```

com a alternativa a una iteració

```
>>> suma=0
>>> for n in range(1,101):
...     suma+=1.0/(n**2)
>>> suma
1.6349839001848923
```

o a funcions sobre els índexs (donat que `arange()` produeix una `array`)

```
>>> indexs=arange(1,101)
>>> sum(1.0/(indexs**2))
1.6349839001848923
```

o sense passar-ho usant la funció `map` sobre llistes

```
>>> sum(map(lambda x:1.0/(x**2),range(1,101)))
1.6349839001848923
```

A.8. Matrius i sistemes d'equacions lineals

A `Matlab/Octave`, els vectors i les matrius són essencialment la mateixa estructura. En l'entorn `Python/Pylab`, hem vist el tipus `array` per representar vectors. Encara que les matrius les introduïm mitjançant el tipus `array` també, el seu comportament és força diferent. Per representar matrius, passarem a `array` una llista de llistes amb cadascuna de les files i accedirem als elements de manera natural:

```
>>> A=array([[1,2,3],[4,5,6]])
>>> A
array([[1, 2, 3],
       [4, 5, 6]])
>>> A[1,2] # tambe A[1][2] com a llista de llistes Python
6
>>> A[:,0:2] # submatriu
array([[1, 2],
       [4, 5]])
>>> A[:,0] # primera columna
array([1, 4])
>>> A[1,:] # segona fila
array([4, 5, 6])
```

És important notar que el resultat de seleccionar una columna és una `array` unidimensional, en comptes del que succeeix en `Matlab/Octave`. Podem modificar elements de la matriu, subconjunts, o copiar-la, tot i que en aquest darrer cas és important recordar que es tracta d'objectes mutables i que, per tant, no les podem copiar directament sinó que ho hem de fer a través del mètode `copy`

```
>>> A[0,1]=7; A # assignem un element
array([[1, 7, 3],
       [4, 5, 6]])
```

```
>>> A[0,0:2]=arange(8,10); A # assignem un tros d'una fila
array([[8, 9, 3],
       [4, 5, 6]])
>>> B=A.copy()  # copiem A a B
>>> B[:2,:2]=[[0,0],[0,0]] # no cal posar array a la dreta
>>> B
array([[0, 0, 3],
       [0, 0, 6]])
```

Per conèixer la dimensió d'una matriu, podem fer `A.shape` o `shape(A)`. També podem modificar les dimensions de les matrius a través del mètode `reshape`, que ens permet especificar noves dimensions en una `array` sense modificar-ne les dades:

```
>>> array([2,3,4,5]).reshape(4,1)
array([[2],
       [3],
       [4],
       [5]])
>>> array([2,3,4,5]).reshape(2,2)
array([[2, 3],
       [4, 5]])
```

on la primera instrucció és útil per crear vectors columna. Altres operacions interessants són les que tenim a continuació:

- Ampliació de matrius: podem concatenar dues arrays `A` i `B` o bé verticalment mitjançant la instrucció `vstack((A,B))` (`[A;B]` en `Matlab/Octave`) o bé horitzontalment amb l'ordre `hstack((A,B))` (`[A,B]` en `Matlab/Octave`)

```
>>> vstack((A,B)) # [A;B] en Matlab/Octave
array([[8, 9, 3],
       [4, 5, 6],
       [0, 0, 3],
       [0, 0, 6]])
>>> hstack((A,B)) # [A,B] en Matlab/Octave
array([[8, 9, 3, 0, 0, 3],
       [4, 5, 6, 0, 0, 6]])
```

- Transposició de matrius: `A.transpose()` o bé `transpose(A)`.

A.8.1 Operacions amb matrius

La suma `+`, la resta `-` i el producte per escalars sobre *arrays* són naturals:

```
>>> A+B
array([[ 8,  9,  6],
       [ 4,  5, 12]])
>>> 2*A
array([[16, 18,  6],
       [ 8, 10, 12]])
```

però, en canvi, el símbol del producte és per a l'operació element a element:

```
>>> A*B # NO producte matricial
array([[ 0,  0,  9],
       [ 0,  0, 36]])
```

mentre que l'operació del producte matricial s'indica amb la funció `dot()`:

```
>>> C=dot(A,B.transpose()); C
array([[ 9, 18],
       [18, 36]])
```

De la mateixa manera, el símbol `C**2` representa l'operació element a element mentre que la potència enèsima de la matriu (com a producte matricial) s'indica amb la funció `matrix_power()`:

```
>>> matrix_power(C,2)
array([[ 405,  810],
       [ 810, 1620]])
```

i, si el determinant (`det()`) és diferent de zero, en podem calcular la inversa amb `inv()`:

```
>>> det(C)
0.0
>>> D=C+eye(2) #C+ Identitat 2x2 no singular
>>> inv(D) # calcul de la inversa
array([[ 0.80434783, -0.39130435],
       [-0.39130435,  0.2173913 ]])
```

A.8.2 Creació de matrius especials

La majoria de funcions per a la creació de matrius amb alguna estructura especial que hem vist en llenguatge Matlab/Octave tenen un anàleg molt directe en Python/Pylab:

- Matrius i vectors de zeros: `zeros(n)` crea un vector de zeros de dimensió `n` (diferent de Matlab/Octave) i `zeros(n,m)`, una matriu de zeros de dimensió `m x n` (com en Matlab/Octave).
- Matrius i vectors d'uns: `ones(n)` crea un vector de zeros de dimensió `n` (diferent a Matlab/Octave) i `ones(n,m)` una matriu de zeros de dimensió `m x n` (com en Matlab/Octave).
- Matriu identitat `eye(n)` (com en Matlab/Octave).
- Matrius diagonals `diag(v)` amb `v` vector i `diag(v,posicio)` amb `posicio` enter (com en Matlab/Octave).
- Matrius aleatòries `rand(n)` i `rand(n,m)` (com en Matlab/Octave).
- Convertir matrius a triangulars inferiors `tril(A)` i superiors `triu(A)` (com en Matlab/Octave).
- Matriu de Vandermonde associada a un vector `v`: `vander(v)` (com en Matlab/Octave).

- Matriu de blocs `A1`, `A2` i `A3`: `blk_diag(A1,A2,A3)` del mòdul `scipy.linalg` (per tant, farem `from scipy.linalg import blk_diag`).
- Matriu *companion* d'un vector `v`: `companion(v)` del mòdul `scipy.linalg` (per tant, farem `from scipy.linalg import companion`).

Per a aquestes dues darreres funcions, caldrà que la versió de `Scipy` sigui superior o igual a la `0.8`.

A.8.3 Sistemes d'equacions lineals

Per resoldre un sistema lineal $Ax = b$, on A és una matriu quadrada no singular i b és un vector de la mateixa dimensió usarem l'ordre `solve`. Així, seguint l'exemple de la pràctica 6:

```
>>> A=vander(arange(1,6))
>>> b=cos(arange(1,6))
>>> solve(A,b)
array([-0.03486786, 0.43661045, -1.55666444, 1.18028894, 0.51493522])
>>> dot(A,solve(A,b))-b # Comprovem el resultat
array([  0.00000000e+00,   5.55111512e-17,   4.44089210e-16,
         0.00000000e+00,   7.21644966e-16])
```

Una diferència important amb `Matlab/Octave` és que l'entorn `Pylab` per al `Python` no té rutines per calcular la forma escalonada reduïda (`rref`), així com les funcions associades (`null`, `rank`). Això es deu als problemes numèrics que vàrem veure a l'apartat 6.3.2 i que fa que aquests mètodes siguin poc pràctics o fiables per a dimensions grans. En cas que calgui calcular la forma esglaonada reduïda en `Python` recomanem usar algun dels paquets per a càlcul simbòlic, com `Sympy` [20] o l'entorn `SAGE` [8].

A.8.4 Valors i vectors propis

En l'entorn `Pylab`, podem usar la funció `eigvals` per obtenir només els valors propis o bé la funció `eig` per obtenir valors i vectors propis:

```
>>> A=vander(arange(1,6))
>>> eigvals(A)
array([  4.59604348e+01,  -2.89437019e+01,   6.79538792e+00,
        -8.49619485e-01,   3.74986442e-02])
>>> vaps,veps=eig(A)
>>> dot(A,veps[:,2])-vaps[2]*veps[:,2] # comprovem el resultat
array([  6.66133815e-16,   2.22044605e-15,   1.77635684e-15,
         3.77475828e-15,   0.00000000e+00])
```

i obtenim la forma diagonal a través de

```
>>> dot(inv(veps),dot(A,veps))
array([[  4.59604348e+01,   3.06569006e-15,  -1.46434514e-14,
         -5.25766225e-15,  -1.22396587e-14],
       [ -7.16180587e-15,  -2.89437019e+01,  -1.11560067e-14,
         -8.33466868e-16,  -5.28838117e-15],
```

```
       [  1.09547788e-15,  -5.43749074e-15,   6.79538792e+00,
         -2.29083788e-15,  -7.16729634e-17],
       [ -5.47131784e-15,   1.70263109e-15,  -1.81387023e-16,
         -8.49619485e-01,  -5.17729831e-17],
       [ -6.96057795e-15,  -2.78509854e-15,  -1.39780765e-15,
         -1.13143951e-15,   3.74986442e-02]])
```

Podem comprovar els valors propis calculant el polinomi característic i avaluant-lo en les arrels

```
>>> polinomi=poly(A)
>>> polyval(polinomi,vaps)
array([  1.56700253e-09,   4.53297844e-09,  -4.16662260e-11,
         7.38964445e-13,   0.00000000e+00])
```

Bibliografia

[1] J. Amorós. *Introducció breu al Matlab*.
http://www.ma1.upc.edu/~numeric/tema1matlab.pdf. [Recurs electrònic].

[2] Guillem Borrell. *Introducción a Matlab y Octave.*
http://iimyo.forja.rediris.es/tutorial. [Recurs electrònic].

[3] A. Downey, J. Elkner, C. Meyers. *How to think like a Computer Scientist: Learning with Python.* Segona edició. Green Tea Press, 2002.

[4] Javier García de Jalón, José Ignacio Rodríguez, Jesús Vidal. *Aprenda Matlab 7.0 como si estuviera en primero.* Escuela Técnica Superior de Ingenieros Industriales. Universidad Politécnica de Madrid, 2005.
http://mat21.etsii.upm.es/ayudainf/aprendainf/Matlab70/matlab70primero.pdf.

[5] Guido Van Rossum i altres. *Python Programming Language.*
http://www.python.org. [Recurs electrònic].

[6] John D. Hunter i altres. *Matplotlib: Python Plotting Library.*
http://matplotlib.sourceforge.net. [Recurs electrònic].

[7] Travis Oliphant i altres. *Numpy: Python for Scientific Computing.*
http://www.numpy.org. [Recurs electrònic].

[8] William A. Stein i altres. *Sage Mathematics Software.*
http://www.sagemath.org. [Recurs electrònic].

[9] Eric Jones, Travis Oliphant, Pearu Peterson i altres. *SciPy: Open Source Scientific Tools for Python*.
http://www.scipy.org. [Recurs electrònic].

[10] J. Kiusalaas. *Numerical Methods in Engineering with Python*. Cambridge University Press, 2010.

[11] H.P. Langtangen. *A Primer on Scientific Programming with Python*. Springer Verlag, 2009.

[12] R. Larson, B.H. Edwards. *Cálculo I de una variable*. McGraw-Hill, 9a edició, 2010.

[13] The MathWorks. *Matlab: The language of technical computing.* http://www.mathworks.com/products/matlab. [Recurs electrònic].

[14] Cleve B. Moler. *Numerical computing with Matlab.* Society for Industrial and Applied Mathematics (SIAM), 2004. http://www.mathworks.com/moler/chapters.html.

[15] J. Puig. *Programació gràfica d'ordinadors*. Crèdit variable d'informàtica. Departament d'Ensenyament, 1991.

[16] A. Quarteroni, F. Saleri. *Cálculo científico con Matlab y Octave.* Springer-Verlag, 2006.

[17] T. Susín. *Pràctiques de laboratori d'àlgebra lineal.* http://www.ma1.upc.edu/docencia/assignatures/algebra/lab_algebralineal.pdf. [Recurs electrònic].

[18] T. Susín. *Pràctiques de laboratori de càlcul infinitesimal II.* http://www.ma1.upc.edu/docencia/assignatures/calcul2/lab_calcul2.pdf. [Recurs electrònic].

[19] Octave Development Team. *GNU/Octave.* http://www.gnu.org/software/octave/. [Recurs electrònic].

[20] SymPy Development Team. *Sympy: Python Library for Symbolic Mathematics.* http://www.sympy.org. [Recurs electrònic].