Introductie GAUSS

Gerrit Draisma
Marius Ooms

Econometrisch Instituut
Erasmus Universiteit Rotterdam
Postbus 1738
NL - 3000 DR Rotterdam

http://move.to/econometrics

Rotterdam, 25 januari 2000.

Voorwoord

Dit is een hands-on introductie van het pakket GAUSS. De teksten van opdrachten en GAUSS code zijn als kleine GAUSS-programmabestanden: (`INTRO012.E`, `INTRO013.E` enz.) per paragraaf beschikbaar. Het is belangrijk dat u die in de volgorde van deze introductie uitvoert.

Uit te voeren opdrachten zijn steeds met ⇒ in de kantlijn aangegeven.

De volgende conventies zijn min of meer gehanteerd in de tekst. Commando's, bestandsnamen enz. worden met `typemachine letters` weergegeven. Toetsaanslagen in de geactiveerde GAUSS-windows worden zo weergegeven: `<toets>`.
`<alt-F>`,`<E(x)it>` betekent bijvoorbeeld dat u `alt-F` en daarna de `x` moet indrukken.

In syntax diagrammen betekent

- `<arglist>` een lijst van variabelenamen gescheiden door komma's,
- `[optional parameter]` een optioneel deel van het commando, en
- `[choice1|choice2]` een keuze.

GAUSS is een numerieke gereedschapskist waarin al heel veel aanwezig is, en wat ontbreekt kan eenvoudig toegevoegd worden. Reeds in 1991 besloot de vakgroep Econometrie om die reden GAUSS te gebruiken bij haar werkcolleges. We hopen dat deze introductie duidelijk kan maken dat ook de hier behandelde versie van GAUSS een handig hulpmiddel is.

Deze introductie behandelt de studentenversie 3.2.34 voor Windows-NT (April 1998). Dit is nog steeds een zogenaamde beta-versie. Dat betekent o.a. dat nog niet alle onderdelen goed zijn geïmplementeerd. Met name de editor, de Help-faciliteit, het importeren en exporten van grafieken en de "debugger"zijn nog niet goed geïmplementeerd.

U kunt de GAUSS-programma's natuurlijk ook met uw favoriete editor aanpassen.

Met het gratis programma `playw` kunnen GAUSS-graphics vanuit het .tkf-formaat worden omgezet naar formaten die in moderne Windows-applicaties kunnen worden gebruikt.

Gerrit Draisma

Marius Ooms

Rotterdam, 25 januari 2000.

Inhoudsopgave

HOOFDSTUK 1

Programmeeromgeving

1.1. Beginnen en ophouden

U start GAUSS op via het Windows-NT Start-menu → Programs → Mathematics door te klikken op de GAUSS-icoon met de hamer die op een matrix slaat. U eindigt GAUSS met het commando `system`, met het `<x>`-icoon op het 'command-window' of met de toetsen `<Alt-F><E(x)it>`. GAUSS bestaat naast het basisprogramma met het 'Command-window' uit een eenvoudige programma-editor GAUSS-Edit en uit een GAUSS Browser. In dit hoofdstuk maakt u kennis met deze drie basisprogramma's en met de GAUSS Help-bestanden.

1.2. Command-mode (gauss-window) en Edit-mode (gauss-Edit-window)

GAUSS kent traditioneel twee werkomgevingen: 'command-mode' en 'edit-mode'. In de windows-versie hebben deze twee 'mode's ieder een eigen window. In het 'command-window' met de titel GAUSS kan men interactief werken: commando's geven en standaard uitvoer bekijken. Met het commando `cdir(0) <enter>` kunt u zien welke folder (directory) "current" is voor GAUSS. Met het commando `chdir <folder> <enter>` kunt u interactief van "current" folder (directory) veranderen.

In het 'edit window' met de titel GAUSS-Edit kan men programmabestanden maken of wijzigen. Men kan gemakkelijk van het ene window naar het andere window omschakelen. Dit kan door het klikken op de betreffende velden in de Task-Bar onderaan het scherm, maar ook met commando's, toetsen of toetscombinaties:

Van command-mode naar edit-mode: U gaat van command-mode naar edit-mode met

- `edit <file><enter>`: met dit commando wordt `<file>` geladen in het GAUSS-Edit-window.
- `<F4>` of `<Alt-W><E>`: met deze toetsen roept men de ingebouwde editor van GAUSS in het GAUSS-Edit-window aan met daarin het laatst bewerkte bestand.
- `<Edit>`: met de grote Edit-button gebeurt hetzelfde.

Van edit-mode naar command-mode: U gaat van edit-mode naar command-mode met

- `<F3>` of `<Alt-F><R(u>n>`; Met deze toetsaanslagen verlaat men de edit-mode (het GAUSS-Edit-window) en wordt in de meeste gevallen in de command-mode (het GAUSS-window) direct het hele bestand als programma uitgevoerd:
- `<Run>`-button; met de grote Run-button gebeurt hetzelfde.

⇒ Laadt het bestand `INTRO012.E` in de GAUSS-editor: `<alt-F><Ed(i)t> intro012.e <enter>`. Voer vervolgens onderstaande regels uit dat bestand uit met `<F3>`. Kom terug in de editor met `<F4>`

```
format 8, 2 ;
x = {1 1 1,1 0 0,0 0 1} ;
b = {1,2,3} ;
y = x*b ;
"x=" x ; "b=" b ; "y=" y ;
"return to editor with <F4>" ;
end ;
```

Er wordt automatisch een logfile `COMMAND.LOG` met alle interactief gegeven commando's bewaard.

1.3. Help

GAUSS heeft een help-faciliteit. Veel informatie over werkomgeving, operatoren en functies is *online* aanwezig. GAUSS wordt regelmatig aangepast en de bijbehorende informatie is niet altijd up-to-date of volledig.

De belangrijkste Help-functie, de Browser, is direct bereikbaar met `<Ctrl-B>` of via het menu. `<alt-F>`; `<B>`. De GAUSS Browser is een makkelijke ingang tot het GAUSS Help systeem. Bovendien zoekt en vindt u met de Browser de source code en voorbeelden van veel GAUSS-procedures, keywords en functions. Verder heeft GAUSS een conventionele Window Help-functie. Tenslotte is het handig om met de geavanceerde optie (Advanced) van het Zoek/Find programma van Windows op tekst te zoeken in de source- (`src-`) en `examples`-folder van GAUSS. Via Engelse woorden in de documentatie van de bestanden vindt u dan vaak wat u zoekt. Op het FEW-netwerk staan de GAUSS-bestanden in `T:\APP32\GauL3235`.

⇒ Roep het helpscherm op met `<alt-h>` of `<F1>`. Hiermee krijg je de "Contents"te zien: een overzicht van de belangrijkste commando's in het GAUSS Window. Merk op dat het Search-commando hier niet direct werkt. Je kan wel direct zoeken vanuit het GAUSS of GAUSS-Edit window:

⇒ Roep een overzicht van editor keys op met:
- `<alt-h>` ; `<S)earch>` for help on
- `<alt-T>ype>` de woorden: `editor keys <enter>`
- zoek op wat de "Hot-Key"`<Ctrl-F4>` doet

⇒ Roep help op over de procedure "OLS"met
- `<alt-h>` ; `<S)earch for help on>`
- `<alt-T>ype>` het woord: `OLS <enter>`

⇒ Roep help op over de GAUSS-procedure voor "descriptive statistics" met
- type `*stat*` in de command-mode of in de edit-mode
- markeer `*stat*` met de linker muisknop
- `<Ctrl-B>`
- dubbel-klik met linker muisknop

⇒ Roep help op over de "Publication Quality Graphics"met
- `<alt-h>`
- Zoek het onderwerp op in de "Contents"

⇒ Roep een overzicht van de operatoren op met:

- `<alt-h>` ; `<S)earch for help on>`
- `<<alt-T>ype>` het woord: `OPERATORS <enter>`
- zoek op wat de "./-operator doet

⇒ Roep informatie op over de operator "./"met:

- `<Ctrl-B>`

⇒ Zoek bestanden op in de `SRC`-folder van GAUSS m.b.t. minimalisatie:

- Windows-NT Start-menu → Find → Files → `<<alt-L>ookin>:T:\APP32\GauL3235\SRC`, Advanced, `<<alt-C>ontaining text>`:minimi, `<<alt-i>nd Now>`.

⇒ Zoek informatie op over singuliere waarden:

- Gebruik één van bovenstaande methoden

1.4. Besturingssysteem (Operating System, NT commando's)

U kunt vanuit GAUSS tijdelijk naar het besturingssysteem gaan met het commando `shell`. Met `exit` keert u weer terug naar GAUSS. U kunt ook direct een commando `cmd` van het besturingssysteem uitvoeren met `shell cmd`. De uitvoer van dit shell-commando krijgt u niet in het GAUSS-window te zien. Met `shell cmd & pause` kunt in Windows-NT wel de output van `cmd` in het window van het besturingssysteem bekijken.

Met `shell del TMP.TXT` kunt u vanuit GAUSS het bestand `TMP.TXT` verwijderen.

⇒ Vraag in het besturingssysteem met het commando `dir` de bestanden in de current folder (directory) op.

1.5. Compile-time / Run-time

Statements worden in twee fasen uitgevoerd: In beide fasen kunnen fouten optreden.

compile time: De programma statements worden gelezen en gecompileerd. Fouten die hierbij optreden zijn *compile-time errors*. Als deze fouten optreden, wordt geen enkel statement uitgevoerd.

run time: De gecompileerde code wordt uitgevoerd. Fouten die nu optreden zijn *runtime errors*. In dit geval worden de statements tot aan de fout uitgevoerd.

.

⇒ Voer het onderstaande programma `intro015.e` uit. Gebruik `<F3>` of de `Run`-button in de edit-mode (of `<Shift-F3>` als de filenaam in het tweede venster staat) of typ `run intro015.e` in de command-mode. Vraag (in 'command-mode') met `show` op welke variabelen aangemaakt zijn: Treedt tijdens het compileren al een fout op, dan wordt geen enkel statement uitgevoerd, en worden x en y niet geï nitialiseerd.

```
new ;
x = 2 ;
y = 0 ;
x ; y ;
" Nu proberen we z te genereren";
z = x#y ;
z ;
"<F4> - terug naar editor" ;
```

⇒ Verander in bovenstaande regels `#` in `/` en herhaal de bovenstaande opdracht. Als de fout optreedt tijdens het runnen, dan worden x en y wel geïnitialiseerd.

1.6. Fouten vinden en verwijderen (Debuggen)

Een fout in een programma geeft vaak aanleiding tot een waslijst van foutmededelingen van de compiler.

De belangrijkste tip: Let op het *regelnummer* van de *eerste* foutmededeling! Breek verdere uitvoer van de compiler af met de `STOP`-button en verbeter die regel eerst.

Enkele algemene fouten zijn:

- `;` vergeten aan einde van een statement,
- spaties rond een binaire operator in een print statement : `print x + y` wordt gelezen als `print x` gevolgd door `print + y`. De oplossing: haakjes.
- string operatoren vergeten: `string1 + string2` i.p.v. `string1 $+ string2`.

De fouten krijgt u direct te zien in de command-mode. Ze worden ook bijgehouden in een bestand (`GAUSS.ERR`).

⇒ Schrijf het bestand met onderstaande regels, `INTRO016.E`, weg als een nieuw bestand met de naam `TEST.E`. Voer daarna het programma uit:

- `<alt-F>`, `<Sa(v)e as>` `TEST.E` `<alt-S><alt-S>` - wegschrijven,
- `<F3>`, - voer het programma uit, let op het regelnummer van de fout
- `<F4>`, - ga terug naar de editor en verbeter de fout(en):
- `<ctrl-G>` regelnummer - ga naar de plaats van de fout, voer het programma weer uit met `<F3>` etc. tot de gewenste grafiek verschijnt.

⇒ Laadt de graphics library indien nodig met een `library pgraph` statement.

```
x = seqa(0,.1,21)
y = x^2 ;
x ~y ;
title( "x tegen y= x^2" ) ;
xy(x,y) ;
```

HOOFDSTUK 2

Syntax

Dit hoofdstuk geeft een korte introductie op de syntax van de taal GAUSS. GAUSS lijkt op C of Pascal. De belangrijkste verschillen zijn:

- De *matrix* is het standaard datatype in GAUSS. *Vectoren* zijn matrices met 1 kolom of 1 rij; *scalairen* zijn 1×1 matrices.
- variabelen en parameters van procedures worden *niet expliciet gedeclareerd*. Dimensies van variabelen kunnen in de loop van een programma wijzigen; de dimensies van de invoerparameters van een procedure bepalen dikwijls de dimensies van het resultaat.

Samen met hoofdstuk 3 geeft dit hoofdstuk een basis om met GAUSS aan de slag te gaan.

2.1. Commentaar

GAUSS kent twee soorten commentaar.

```
/*
**
/* Dit is STANDAARD commentaar: het mag genest voorkomen */}
**
*/
new ; @ clear memory @
      @Dit is KORT commentaar: het mag niet genest voorkomen @
```

Het commentaar bij procedures is bij GAUSS meteen de documentatie. Elke (niet triviale) procedure wordt voorafgegaan door een beschrijving van doel, invoer, uitvoer enz. Hierbij een sjabloon; U kunt het inlezen als U een procedure schrijft. De `**` aan het begin van elke regel zijn niet noodzakelijk, maar wel handig. Met het Windows NT commando `find` of met het hulpprogramma `grep` van Borland Pascal (C) kunt u in één keer uw documentatie bij elkaar verzamelen. Het eerste woord in de regel met `**>` kan met de bijbehorende locatie worden toegevoegd aan de indexering van de GAUSS Browser. Als u de procedures later in een in een groter `.src`-bestand bijeen voegt en vervolgens in een `library` stopt, zie §5.6, dan wordt de documentatie automatisch bereikbaar met de GAUSS Browser.

⇒ Schrijf onderstaande regels weg als `KOP.G`.

```
/*
**> <procedure>
**
** purpose:
**
```

```
** format:
**
** input:
**
** output:
**
** globals:
**
** author:
**
** date:
**
*/
```

2.2. Statements

Statements moeten worden afgesloten met ;. GAUSS maakt standaard geen onderscheid tussen hoofd- en kleine letters.

⇒ Hieronder staat een stukje programma, waarin een X matrix met drie kolommen, een vector β, en een vector e van normaal verdeelde storingen worden gegenereerd.

⇒ Voer deze regels uit met <F3>.

```
n = 4 ; sigma = 0.25 ;
X = ones(n,1)~rndu(n,2) ;
beta = { 1,2,3 } ;
e = sigma * rndn(n,1) ;
print "x    " X ;
   "beta " beta ;  @ "print" mag weggelaten worden @
   "e    " e ;
   "ga terug naar de editor met <F4>" ;
```

⇒ Zoek met help de functies ones(), rndn(), rndu() op.

⇒ Voeg regels toe waarmee u $y = X\beta + e$ uitrekent en afdrukt.

⇒ Voer die regels uit met <F3>.

2.3. Namen

Namen mogen maximaal 32 tekens lang zijn, de tekens [A-Z], [a-z], [_] en [0-9] bevatten, en niet met [0-9] beginnen.

Afspraak: Voor *globale variabelen* gebruikt men namen die beginnen met met '_' , bijv. '_tol'; veel procedures gebruiken globale variabelen in plaats van argumenten of parameters. Dit voorkomt ellenlange parameterlijsten bij de aanroep van die procedures.

⇒ De globale variabele _fcmptol wordt door de procedures feq() ('fuzzy equal'), fne(), etc. gebruikt om te beslissen wanneer twee getallen 'ongeveer gelijk' zijn. Voeg hieronder statements toe, zodat de eerste aanroep onwaar (0), en de tweede waar (1) oplevert, en voer de regels uit.

```
feq(0,1E-6) ; "onwaar" ;
feq(0,1E-6) ; "waar" ;
```

2.4. Matrixconstanten

Matrixconstanten worden gebruikt om waarden aan variabelen toe te kennen. Hier volgt een aantal manieren waarop u dat kan doen.

⇒ Voer onderstaande regels uit en bekijk het het resultaat.

```
v = { 1,2,3 } ; h = { 1 2 3 } ;
m = { 11 12 13, 21 22 23, 31 32 33 } ;
let v2 = 11 12 13 ;
let m2[3,2] = 1 2 3 4 5 6 ;
```

2.5. Procedures definiëren en documenteren

Een GAUSS procedure komt overeen met functie in Pascal of C. Een procedure voert berekeningen uit op de invoerargumenten en geeft het resultaat daarvan terug aan het aanroepende programma, toont het op het scherm of doet beide: Een procedure als `eigrs()` geeft alleen resultaten (eigenwaarden) terug, `xy()` toont ze (grafieken) alleen op het scherm en `ols()` doet beide. GAUSS kent twee soorten procedures: *intrinsieke* procedures die in het programma zelf, dus al gecompileerd, aanwezig zijn en daarnaast de procedures (veel meer) die als *source code* meegeleverd worden (de *runtime library*) of door de gebruiker zelf ontwikkeld worden.

Een procedure wordt als volgt gedefinieerd:

```
PROC <pname>(<arglist>) ;
LOCAL <varlist> ;
   <statements>
RETP(<return value>) ;
ENDP ;
```

Een voorbeeldje:

```
PROC MYOLS( y, X ) ;
LOCAL b ;
   b = y/X ;
RETP(b) ;
ENDP ;
```

Deze procedure `myols` schat de coëfficienten van een lineair model. De afhankelijke (`y`) en de onafhankelijke (`X`) worden als argumenten meegegeven. Locale variabelen (`b`) moeten apart gedeclareerd worden. Met `RETP` wordt het resultaat teruggegeven.

⇒ Bekijk met `help` hoe GAUSS zijn procedures documenteert. Kijk bijvoorbeeld naar de documentatie van de procedure cdfn. Voeg op dezelfde manier commentaar in een "kop" toe bovenaan `myols`. Schrijf de procedure weg als `MYOLS.G`.

⇒ Test de procedure op onderstaande data (Judge, George G., Carter Hill, R. Griffith, William E., Lütkepohl, Helmut, Lee Tsoung-Chao (1988), Introduction to the Theory and Practice of Econometrics, John Wiley & Sons, New York, 2nd Ed., table 5.3, p. 195): Gewicht braadkippen (`y`) tegen voeder hoeveelheid (`x1`).

```
new ;
format /rz 10, 3 ;
let y =  .58    1.1     1.2     1.3     1.95
        2.55    2.6     2.9     3.45    3.5
        3.6     4.1     4.35    4.4     4.5  ;
let x1=   1      2       3       4       5
          6      7       8       9      10
         11     12      13      14      15  ;
X = ones(15,1)~x1~x1^2 ;
y~x ;
@ check MYOLS  @ ;
end ;
```

Sommige procedures geven een *meervoudig resultaat* terug. De definitie van een procedure die een dubbel resultaat teruggeeft ziet er bijv. zo uit:

```
PROC (2) = <pname> ( <arglist> ) ;
LOCAL <varlist> ;
   <statements>
RETP (<retval1,retval2>) ;
ENDP ;
```

⇒ Schrijf een procedure `MYOLS2(y,X)` met als uitkomsten :

- de geschatte coëfficiënten, `b`, en
- de geschatte variantie van de storingen, `s2`,

natuurlijk goed gedocumenteerd, en weggeschreven als `MYOLS2.G`.

⇒ Behalve resultaten teruggeven aan het hoofdprogramma, kunnen procedures ook resultaten afdrukken op het scherm. Voeg print-statements toe aan `MYOLS2`, zodat de procedure de volgende informatie op het scherm zet:

```
Aantal waarnemingen: xx
Coefficienten:
   xx
   xx
Variantie storingen:
   xx
```

2.6. Procedures aanroepen

Procedures kan men op drie manieren aanroepen.

⇒ Probeer onderstaande aanroepen:

```
"(1): Resultaat wordt toegekend aan een variabele:" ;
bhat = MYOLS(y,X) ;
"(2): Resultaat wordt op het scherm gezet:" ;
MYOLS(y,X) ;
"(3): Resultaat wordt niet gebruikt"
"     in het aanroepende programma " ;
call MYOLS(y,X) ;
"Nog eens:" ;
"(1)" ;
{ bhat, s2 } = MYOLS2(y,X) ;
"(2)" ;
MYOLS2(y,X) ;
"(3)" ;
call MYOLS2(y,X) ;
```

De grafische procedures geven een goed inzicht in het gebruik van globale variabelen in procedures. De aanroep zelf is heel simpel; met behulp van globale variabelen kunt u de grafieken 'verfraaien'. Veel informatie kunt vinden in het handboek: System and Graphics Manual Volume 1. Het leesbare bestand `PGRAPH.DEC` in de `SRC`-folder van GAUSS beschrijft ook hoe u deze globale variabelen kunt gebruiken.

⇒ Voer onderstaand plot-statement uit.

⇒ Vraag `<H)elp>` over `xy()`. Zoek procedures waarmee U labels bij de assen kunt plaatsen.

⇒ Vraag `<H)elp>` over de globale variabelen: `_plctrl` (line control), `_pltype` (line type) `_pstype` (symbol type), `_pcolor`

⇒ Maak uiteindelijk één grafiek van:

- `x1` (voer) versus `y` gewicht): losse symbolen, geen lijnen
- `x1` (voer) versus `yhat` (model gewicht) : geen symbolen, lijnen

```
xy(x1,y) ;
@ construct yhat     @
@ plot x1 vs. y~yhat @
end ;
```

2.7. Functies van 1 regel

Naast procedures kent GAUSS ook functies van 1 regel. Ze worden zelden gebruikt. Deze functies worden als volgt gedefinieerd:

```
FN <functie>(<arglist>) = <expression> ;
```

⇒ Voer onderstaande regels uit:

```
FN area(r) = pi * r * r ;
area(2) ;
```

2.8. Keywords

Naast procedures kent GAUSS keywords. Het argument voor een keyword is de hele string achter het keyword tot het eind van de regel. Er hoeven geen haakjes om het argument. Voorbeelden zijn `format`, `load` en `save`. De file `GAUSS.SRC` bevat een aantal keyword-definities om zonder GAUSS te verlaten commando's aan het operating system (de shell, bijv. DOS) te geven vanuit GAUSS.

Keywords worden als volgt gedefinieerd:

```
KEYWORD <kword> (<str_arg>);
LOCAL <varlist> ;
   <statements>
ENDP ;
```

⇒ Zoek in `HELP.G` in de sub-folder `LOCALSRC` van de GAUSS-folder op hoe wij het keyword `help` hebben gedefinieerd.

HOOFDSTUK 3

Operatoren

Een matrixtaal als GAUSS heeft globaal drie soorten operatoren nodig:

1. index-operatoren verwijzen naar de elementen van een matrix of vector,
2. matrix-operatoren voor de standaard matrix operaties en
3. ExE-operatoren voor elementsgewijze operaties op de operanden.

Hieronder wordt een paar matrices aangemaakt waarmee we de werking van de operatoren in GAUSS onderzoeken.

⇒ Voer de onderstaande regels uit en bekijk het resultaat.

```
new ;
format /rz 8, 3 ;
r = 3 ; k = 5 ;
v = seqa(1,1,r) ;
h = seqa(0.1,0.1,k)' ;
A = v+h;
M = rndn(r,r) ;
I = eye(r) ;
wait ;
"v" v;  "h " ; h ;
"A" A; "M" M; "I" I;
wait ;
```

3.1. Index-operatoren

Met `[i]` wordt het i-de element van een vector aangeduid; met `[i,j]` het (i, j)-de element van een matrix. Hierbij kunnen zowel `i` als `j` door een *range* vervangen worden. Een range geeft men aan als '`i1 i2 i3 ...` ' of als '`i1:i2`' Tenslotte verwijst een '`.`' naar alle rijen of kolommen van een matrix.

Let op: Spaties binnen de index operator `[]` zijn dus significant. Gebruik ze alleen om twee rij- of kolom-indices te scheiden.

⇒ Probeer onderstaande voorbeelden:

```
"vectoren:" ;
"v " v ;
"v[1] "  v[1] ; wait ;
"h " ; h ;
"h[1 3] " ; h[1 3] ; wait ;
"h[2:4]"  ; h[2:4] ; wait ;
"matrices:" ;
```

```
"A " A ; "A[2,3] " A[2,3];
"A[1:2,3] " A[1:2,3] ;
"A[.,1 3] " A[.,1 3] ;
```

⇒ Hoe krijgt u
- de eerste en tweede kolom van A?
- de 2×2 matrix gevormd door de laatste twee elementen van de eerste en derde rij ?

3.1.1. Vectoren als indices. Tenslotte kunnen ook *vectoren* als indices gebruikt worden. In combinatie met routines die zulke vectoren berekenen vormt dat een heel krachtig mechanisme.

⇒ Voer onderstaande regels uit en bekijk het resultaat.

```
r = { 2,3 } ; k = { 1,2 } ;
"A " A ;
"A[r,k] " A[r,k] ; wait ;
M ;
"wijzig submatrix[r,k] " ;
M[r,k] = eye(2) ; M ;wait ;
```

3.2. Matrix-operatoren

In GAUSS zijn de operanden van alle binaire operaties matrices. U moet zelf als programmeur er voor zorgen dat de operanden *conform* zijn, d.w.z. dat de dimensies kloppen. Tabel 3.1 geeft een overzicht.

TABEL 3.1. Matrix operatoren

operator	operatie
`+`	(uitgebreide) matrix/vector-optelling
`-`	(uitgebreide) matrix/vector-optelling
`*`	standaard matrix-vermenigvuldiging
`/`	deling, oplossing lineair stelsel vergelijkingen, regressie, afhankelijk van de operanden
`~`	horizontaal samenvoegen
`|`	vertikaal samenvoegen
`'`	transponeren, inprodukt `X'y` is kort voor `X'*y`

Optellen en aftrekken zijn uitgebreid in de zin dat de dimensies van de operanden niet gelijk hoeven te zijn (*conform* is voldoende. zie sectie 3.3).

⇒ Voer onderstaand voorbeeld uit. Hierin komen de meeste operatoren aan bod.

```
/* een lineair modelletje */
t = 4 ;
X = ones(t,1)~rndu(t,1) ;
beta = { 1,1 } ; e = 0.1*rndn(t,1) ;
y = X*beta + e ;
"y = X*beta + e " ;
"X " X ; "beta" beta ; "e " e ; wait ;
"y " y ;
"bdak=y/X" y/X ;
```

Dit voorbeeld levert onderstaande output:

```
y = X*beta + e
X
       1     0.83
       1    0.941
       1   0.0136
       1   0.0676
beta
       1
       1
e
 -0.0388
     0.2
  -0.182
  0.0249
y
    1.79
    2.14
   0.832
    1.09
bdak=y/X
   0.903
    1.21
```

⇒ Probeer alle operatoren uit. Hieronder staat een begin. Verbeter zonodig!

```
"v " v ;
"v+v "  v+v ;
"A+M "  A+M ;
"M*v "  M*v ;
"M*h "  M*h ;
```

⇒ De /-operator is een gecompliceerde operator. Afhankelijk van de dimensies van de operatoren is het resultaat een normaal quotient, de oplossing van een stelsel vergelijkingen of de oplossing van een regressie vergelijking zoals hierboven.

```
"v"  v ; "v/3" v/3 ;
"v/M"  v/M ; "M*(v/M)" M*(v/M); wait ;
```

⇒ Ga na wat de de / operator doet bij verschillende dimensies van de operanden.

⇒ Voor de operatoren ~ en | moeten de operanden *conform* zijn. Wat zijn daar de regels voor?

⇒ Tenslotte een voorbeeld met de '-operator. `inv()` levert de inverse van een matrix. Merk op dat GAUSS geen operator heeft voor matrix-machtsverheffen; bijv. `(X'X)^-1` niet werkt.

```
"X" X ;  "X' " X' ;wait ;
"X'X" ; X'X ;
"inv(X'X) "; inv(X'X) ; wait ;
"inv(X'X)*X'y" inv(X'X)*X'y ;
"y/X" y/X ;
```

3.3. ExE- of punt-operatoren

`ExE`- of *element by element* operatoren werken steeds op twee elementen van de operanden. Daarvoor moeten de operanden **ExE-conform** zijn: de elementen van de operanden moeten op een logische manier twee aan twee gepaard kunnen worden. Tabel 3.2 geeft een overzicht. Merk op dat er geen operator is voor het standaard matrix-machtsverheffen.

TABEL 3.2. `ExE` operatoren

operator	operatie
+	optellen
-	aftrekken
.*	vermenigvuldigen
./	delen
^	machtsverheffen
.^	machtsverheffen

⇒ Probeer de volgende voorbeelden en probeer voorbeelden met matrices en vectoren.

⇒ Gebruik in het lineair modelletje hierboven een matrix e i.p.v. een vector.

```
 "v     " v ;
"v+2 " v+2 ;
"v+v " v+v ;
"v+h " v+h ;
"v.*h " v.*h ;
```

⇒ Wanneer zijn operanden `ExE` conform ?

Let op: In GAUSS zijn * en / *matrix*-vermenigvuldiging resp. -deling. Als u in uw programma *scalaire* vermenigvuldiging wilt hebben, gebruik dan altijd .* en ./ !

⇒ Hierbij een demonstratie van het belang van puntoperatoren in procedures. Vergelijk het resultaat van de twee oppervlakte functies hieronder met argumenten van verschillende dimensies.

```
FN opp1(l,b) = l*b ;
FN opp2(l,b) = l.*b ;
opp1(h,v) ;
opp2(h,v) ;
```

3.4. Vergelijkingsoperatoren (Comparison operators)

Vergelijkingsoperatoren geven als resultaat 0 (onwaar) of 1 (waar). Van deze operatoren bestaan twee varianten.

- `<`, `<=`, `==`, `=>` en `>` vergelijken beide operanden in hun geheel; het resultaat is scalair. Alternatieve notatie: `LT`, `LE`, `EQ`, `GE` en `GT`.
- `.<`, `.<=`, `.==`, `.=>`, `.>` vergelijken beide operanden elementsgewijs; het resultaat is een matrix. Alternatieve notatie: `.LT`, `.LE`, `.EQ`, `.GE` en `.GT`.

⇒ Bekijk de volgende voorbeelden:

```
"v  > 2" ; (v > 2) ;
"v .> 2" ; (v .> 2) ;
"v .>= (10*h)" ; (v .>= (10*h)) ;
```

3.5. Logische operatoren

Logische operatoren geven als resultaat 0 (onwaar) of 1 (waar). Bij de operanden worden alle waarden ongelijk aan 0 als waar beschouwd. Ook hier weer twee varianten:

- `NOT`, `AND`, `OR`, `EQV` en `XOR` vergelijken beide operanden in hun geheel; het resultaat is *scalair*.
- `.NOT`, `.AND`, `.OR`, `.EQV` en `.XOR` vergelijken beide operanden elementsgewijs; het resultaat is een *matrix*.

⇒ Bekijk het volgende voorbeeld:

```
a1 = { 0,0,1,1} ;
a2 = { 0,1,0,1} ;
"waarheidstabel .OR :";
   a1~a2~(a1 .or a2) ; wait ;
"gecombineerd met vergelijkingsoperatoren" ;
"v ~ (v.>1).and (v.<3) " ;
   v~((v.>1).and (v.<3)) ; wait ;
"en zonder punten " ;
"(v >1) or (v <3) " ;
   ((v >1) or (v <3))  ;
```

Een aantal GAUSS-procedures gebruikt een 0/1 vector als argument. Zo'n vector maakt men met behulp van *puntoperatoren*. Voorbeelden van dergelijke procedures zijn `substute()`, `selif()`, `delif()`.

⇒ Hieronder staat een voorbeeld van de functie `selif()`. Gebruik de functie `substute()` om alle negatieve waarden van v op 0 te stellen.

```
u = rndn(10,1) ;
w = selif(u, u.>0) ;
u' ; w' ;
```

Let op: Met de *punt*-operatoren kunt u dus vergelijkingen en logische operaties *vectoriseren*. In veel gevallen levert dat een enorme rekentijdwinst op.

HOOFDSTUK 4

Strings en formats

4.1. Strings en matrices

GAUSS kent slechts twee datatypes: *matrices* en *strings*. Type-declaraties zijn niet nodig (zie echter ook hoofdstuk 5).

Strings zijn nuttig voor titels, filenamen enz. Matrices kunnen gebruikt worden om zowel getallen als teksten op te slaan: Een matrix-element bestaat uit 8 bytes en kan dus behalve een real ook een blok van 8 characters representeren. Om een matrix-element als tekst af te drukken, moet er een `$`-teken voor geplaatst worden.

⇒ Hieronder wordt een charactervector gebruikt om koppen boven een tabel te plaatsen. Controleer met `show` dat `s` een string is, en `vnames` een matrix.

```
new;
s = "JULI 1989" ;
vnames = {"leeftijd" "salaris" "geslacht",
          "jaar" "fl." "m=0,v=1" } ;
let v[3,3] = 22 1435 1 10 0 0 85 1385 0 ;
s ;
$vnames ;
v ;
```

4.2. String-operatoren

Strings hebben hun eigen operatoren (zie tabel 4.1).

TABEL 4.1. String-operatoren

operator	operatie
`$+`	koppelen
`$==`,`$>=`, ...	string vergelijking
`^`	substitutie
`$`	matrix element als string weergeven

De substitutie-operator behoeft enige uitleg: alleen in argumenten van GAUSS-keywords (`save`, `load` etc.) wordt `^st` gesubstitueerd door de inhoud van de string- variabele `st` (`X = ^st ;` werkt bijvoorbeeld niet). Deze operator wordt vaak in combinatie met filenamen gebruikt.

In onderstaand voorbeeld worden string-operatoren gebruikt om tussenresultaten naar verschillende bestanden weg te schrijven, om ze later te bekijken. Vooral bij langdurige berekeningen de manier om de resultaten toch snel te presenteren. Let op het gebruik

van de functie `ftos()`: die converteert een real naar een string. `"%*.*lf"` is een zgn. *formatstring* (zie paragraaf 4.4).

⇒ Voer onderstaande code uit en bekijk het resultaat.

```
n = 4 ;
t = 50 ;
i = 0 ;
do while i < n ;
   i = i + 1 ;
   x = rndn(t,1) ;  @ Hier kan een tijdrovende routine staan @
   fnam = ftos(i,"FN_%*.*lf",1,0) ;
   save ^fnam = x ; @ ^fnam wordt vervangen door "FN_1..4" @
endo ;
```

⇒ Controleer of de genoemde bestanden gemaakt zijn. Voer dan onderstaande code uit.

```
/* dit kan later .... */
i = 0 ;
begwind ;
window(2,2,1) ;
do while i < n ;
   i = i + 1 ;
   fnam = ftos(i,"FN_%*.*lf",1,0) ;
   loadm x = ^fnam ; @ ^fnam wordt vervangen door "FN_1..4" @
   ttl  = "reeks " $+ ftos(i,"%*.*lf",1,0) ;
   title(ttl) ;       @ Hier geen ^ttl: title is geen keyword @
   xy(0,x) ;
   nextwind ;
endo ;
endwind ;
```

4.3. Speciale tekens in strings

Speciale tekens in strings worden voorafgegaan door een \ (zie tabel 4.2). Het belangrijkste teken is de \, zelf omdat u dat in bestandsnamen nodig hebt.

⇒ Voorbeelden:

```
"\\7 \7 " ;
"back \bspace" ;
"\ttab" ;
"nieuwe\lregel" ;
"c:\help.txt of c:\\help.txt" ;
```

TABEL 4.2. Speciale tekens in strings

code	teken
\b	<backspace>
\e	<escape>
\f	<form feed> (nw. blad)
\g	<beep>
\l	<line feed> (nwe. regel)
\r	<return>
\t	<tab>
\c	<c> zelf voor de overige tekens,bijv. \\
\\	is de backslash zelf! (bestandnamen!!)
\###"	ascii-code ###,
	bijv. gexp“ 7 = ascii code 7 (beep)

4.4. Uitvoerformaten (output formats)

De uitvoer van GAUSS ziet er het best uit met een font met een vaste breedte, bijvoorbeeld Courier. U kunt het font zelf selecteren met <alt-Fon(t)><S)elect>. Het FORMAT-statement bepaalt hoe matrices op het scherm getoond worden. De syntax luidt:

```
FORMAT [/mf] [/jnt] [f,p]
```

Hierin hebben de optionele argumenten de volgende betekenis:

/mf: matrix formaat. Mogelijke waarden /m0, /m1, /m2, /m3
/jnt: uitvoer formaat voor de elementen met
  j: justification: Right of Left.
  n: notation: Decimal, Exponential, Optimal, Zero's suppressed.
  t: trailing character: Space, Comma, Tab, None.
f: field width
p: precision

In de praktijk voldoet format /rzs 8,4 goed: integers worden zonder decimalen weergegeven, en bij grote of kleine getallen automatisch overgeschakeld op wetenschappelijke notatie.

⇒ Voer onderstaande regels uit, en bestudeer het resultaat van de format specifiers:

```
new;
v = { 0.1,0.11,0.111 };
h = { 1 20 300 4000 50000} ;
x = v.*h ;

/*     /mf /jnt f,p   */
/*-------------------------*/ ;
format     /rds 8,3 ; x ;
format     /ros 8,3 ; x ;
format     /rzs 8,3 ; x ;
format /m0 /rzc 1,5 ; x ; format /m1 /rzs 8,3 ;
```

4.4.1. Verschillende formaten per kolom. `Printfm()` is een procedure waarmee kolommen van een matrix met verschillende formaten kunnen worden afgedrukt.

De syntax luidt:

```
[call] printfm(x ,mask, fmt) ;
```

De procedure wordt aangroepen met 3 argumenten:

`x:` de af te drukken matrix $(m \times n)$.
`mask:` vector $(1 \times n)$ die aangeeft welke kolommen als string (`0`) en welke als getal (`1`).
`fmt:` een format matrix. Een 1×3 vector specificeert 1 format voor alle kolommen; een $n \times 3$ matrix geeft voor elke kolom het format weer. Elk format bestaat uit *format-string*, *veldbreedte* en *precisie*.

De strings zijn in principe de formaat-strings uit de computertaal C. GAUSS voegt hieraan de '%'-tekens toe. Bemerk het verschil met de `ftos()` procedure uit 4.2 waar men de '%'-tekens zelf moet meegeven.

NB: De simpeler procedure `printfmt(x, mask)` is voldoende als het er alleen om gaat tekst- en numerieke kolommen te combineren.

⇒ Een voorbeeld:

```
let xn[3,1] = "een" "twee" "drie" ;
let x[3,3] = .1 .1 .1
             2   2  2
       3001 3001 3001 ;

mask = { 0 1 1 1 } ;      @ kolom 1 als string afdrukken        @

fmt = {"-*.*s"   8 6,   @ kol.1 string left justified        @
     "*.*lf"   8 4,     @ kol.2 floating point 3 decimalen   @
     "*.*lG"   8 4,     @ kol.3 met 3 sign. cijfers zonder 0 @
     "#*.*lG"  8 4 } ;  @ kol.4 met 3 sign. cijfers          @

call printfm(xn~x,mask,fmt) ;
```

4.5. Ontbrekende waarden, Missing values

Ontbrekende waarden worden door GAUSS weergegeven met een '.' .

Er zijn een aantal procedures om met missing values om te gaan. De belangrijkste

`packr(x):` verwijdert alle rijen met missing values,
`ismiss():` geeft aan of het argument missing values bevat,
`scalmiss():` geeft aan of het argument een scalair missing value is,
`miss():` vervangt aangegeven waarden door de missing value en
`missrv():` vervangt de missing value door een bepaalde waarde.

⇒ Bekijk het volgende voorbeeld: `Packr()` verwijdert missing values uit een matrix. De `indexcat()` functie wordt hier gebruikt om een index-vector te maken. Hij geeft een missing value als resultaat, als de aangegeven categorieën) niet voorkomen in het argument. Daar wordt vervolgens op getest.

```
x = { 1 . 2,4 5 6, 7 8 . } ;
x ; "packr(x) " ; packr(x) ;
waitc ;
/* Vindt de tweevoudig extremen in Y */
Y = rndn(100,2) ;
v = indexcat((Y[.,1].>1) .AND (Y[.,2].>1),1) ;
if not scalmiss(v) ;
   print Y[v,.] ;
endif ;
```

De standaard bijgeleverde statistische procedures uit de *runtime library* van GAUSS, o.a. `DSTAT` en `OLS`, leveren voorbeelden voor het omgaan met missing values.

HOOFDSTUK 5

Programmeren

GAUSS is een complete programmeertaal, met IF-statements en DO- loops. Echter, veel zaken, die je in een traditionele taal met deze statements bereikt, zijn in GAUSS met de speciale vector operaties en functies voor elkaar te krijgen. *Vector-operaties en -functies zijn altijd efficiënter dan DO-loops.*

De voorbeelden in dit hoofdstuk zijn langer dan in de voorgaande hoofdstukken. Het werkt waarschijnlijk makkelijker, om ze afzonderlijk weg te schrijven, en dan als programma te runnen. Dat heeft bovendien het voordeel dat de compiler regelnummers vermeldt als er iets mis gaat.

5.1. Lussen (Loops)

`Do-` en `for`-loops gebruikt u voor iteraties. Soms heeft u loops nodig om geheugenproblemen te omzeilen, bijvoorbeeld als de matrices die u bij uw berekeningen nodig hebt te groot worden.

De syntax voor de do-while-loop luidt:

```
do while <expression> ;
  <statements>
  continue ;  @ start een nieuwe loop @
  break ;     @ spring uit de loop @
  <statements>
endo ;
```

`expression` moet een scalair zijn: 0 is `false`, iets anders is `true`.

Vaak moet een verzameling statements een bepaald aantal keer worden uitgevoerd. Men kan dat doen met behulp van een zelfgedefinieerde teller `j` en een conditie in een do-while-loop. Omdat GAUSS geen integers kent wordt `j` automatisch als reëel getal in dubbele precisie bijgehouden en bewerkt en dat kan bij lange meervoudige loops veel tijd kosten. Daarom is in recente versies van GAUSS ook de `for`-loop geïntroduceerd. Binnen deze loop worden er wel "integer" tellers gebruikt. Zo'n `for`-loop werkt veel sneller bij grote problemen.

De syntax voor de for-loop luidt:

```
for i (start, stop, step) ;
  <statements>
  continue ;  @ start een nieuwe loop @
  break ;     @ spring uit de loop @
  <statements>
endfor ;
```

`i` is een lokale teller, die alleen binnen de loop bestaat. `i` kan gebruikt worden om rij- en kolomindices van elementen in een matrix te bepalen. `start`, `stop` en `step` zijn expressies die ook alleen lokaal voor de loop bestaan. Ze worden alleen aan het begin van de loop geëvalueerd en naar integers getransformeerd. In paragraaf 5.3 vindt u een voorbeeld.

5.2. If ... elseif ... else ... endif

`If` wordt gebruikt om fatale fouten te voorkomen en om de argumenten van een procedure te controleren.

De syntax luidt:

```
if <expression> ;
    statements ;
elseif <expression> ;
    statements ;
elseif <expression> ;
    statements ;
    ....
else ;
    statements ;
endif ;
```

`<expression>` moet een scalair zijn: 0 is `false`, iets anders is `true`.

5.3. Vectorizeren

Logische operaties op matrices/vectoren kunnen met de punt-operatoren, en met functies als `delif()` en `selif()` gevectorizeerd worden. Met `indexcat()` kan men zonder do-loop indices van elementen die binnen bepaalde grenzen liggen selecteren.

⇒ In onderstaand voorbeeld worden een `do`-loop en een `if`-statement gebruikt om som en gemiddelde van de positieve gevallen van een steekproef uit een standaardnormale verdeling te berekenen. Met behulp van de functie `hsec` bepaalt men hoeveel tijd hiervoor nodig is.

- Schrijf de code naar schijf als `DOLOOP.E`.
- Voeg code toe die hetzelfde berekent, zonder gebruik te maken van `do` en `if`. Vergelijk de tijden.
- Herhaal dit voor $t = 2000, 8000$.

```
/*
** doloop.e : demo do-loop, for-loop en if-statement
*/
t = 25 ;            @ zet constanten bovenaan in je programma @
X = rndn(t,1) ;
h = hsec ;          @ bewaar de begin tijd @
i = 0 ;             @ tellers op nul @
n = 0 ;
s = 0 ;
do while i < t ;    @ tel alleen positieve x-en @
   i = i + 1 ;
   if x[i] > 0 ;
```

```
      s = s + x[i] ;
      n = n + 1;
   endif ;
endo ;
" Resultaten met do-loop: ";
"Aantal positieve gevallen " n " uit " t ;
"Gemiddelde waarde " s/n ;
"Dit duurde " (hsec-h)/100 " sec." ;
"";
s=0;
n=0;
h=hsec;
for i ( 1, rows(x), 1) ;    @ nu met for-loop @
   if x[i] > 0 ;
      s = s + x[i] ;
      n = n + 1;
   endif ;
endfor ;
" Resultaten met for-loop: ";
"Aantal positieve gevallen " n " uit " t ;
"Gemiddelde waarde " s/n ;
"Dit duurde " (hsec-h)/100 " sec." ;
```

5.4. Procedures als argument

De definitie en aanroep van procedures hebben we al eerder bekeken. In deze paragraaf bekijken we hoe functies en/of procedures als parameters kunnen worden meegegeven aan een procedure. Voorbeelden van dit soort procedures zijn optimalisatie-, integratie- en differentiatieprocedures.

Een procedure met een procedure-parameter wordt zó gedefinieerd:

```
proc (n) = pname( &f, <arglist>) ;
    local f: proc ;
    local <varlist> ;
    <statements>
    {<varlist>} = f(<arglist>) ;
    retp(<varlist>) ;
endp ;
```

Zó roept men hem aan:

```
{<varlist>} = pname( &f, <arglist>) ;
```

De `&`-operator geeft het *adres* van de functie. In de procedure zelf wordt met een `local`-statement aangegeven dat het om een procedure gaat. Ook bij de aanroep wordt weer het adres van de procedure meegegeven.

⇒ Bekijk eerst het hieronderstaande voorbeeld: Gegeven een dichtheidsfunctie, $f(x;\theta)$, berekent onderstaande procedure LOGLIK de loglikelihood:

$$LL = \sum_{i=1}^{n} \ln(f(x_i;\theta))$$

Let op de "kop" (header) van de procedure LogLik: hij specificeert precies hoe de dichtheidsfunctie gedefinieerd moet worden! (Hij maakt wel een beetje van een mug een olifant. Toch is het handig dat de gebruiker direct ziet wat de dimensies van in- en uitvoer zijn.) Voer daarna de volgende opdrachten uit:

- Schrijf de code weg naar `PROCPAR.E`.
- Schrijf de ontbrekende procedure `pdfnorm(x,theta)`, de dichtheidsfunctie van een normale verdeling, met `mu = theta[1,.]` en `sigma2 = theta[2,.]`. U kunt deze procedure achteraan toevoegen aan `PROCPAR.E` of als `PDFNORM.G` wegschrijven.

```
/*
** PROCPAR.E: demo PROCedure als PARameter
*/

/* dimensions etc */
n = 20 ;
mu = 21 ;
s2 = 49 ;

/* the sample */
y = mu + sqrt(s2)*rndn(n,1) ;

/* test of the loglik procedure */
theta = { 15 20 25,49 49 49 } ;
theta ;
loglik(&pdfnorm,y,theta) ;

/*
** LOGLIK:
** purpose: loglik returns the loglikelihood, given a
**          density-function, f(x,theta), a sample x and
**          parameter-vector theta
** format:   LL = loglik(&f,X,theta) ;
** input:
**            f: procedure, which returns the density
**           X : (n x 1), the random sample
**       theta: (p x k)  matrix of parameter-vectors
** output:  LL : (k x 1), a vector of loglikelihoodvalues,
**               each value corresponding to a column of theta.
** globals: none
** remarks: the density function f should have the following syntax:
**          y = f(x,theta) where x,and theta have the above
**          dimensions, and y:(n x k ) with
**          y[i,j] = f(x[i],theta[.,j])
```

```
** Date:    23-7-1993
** Author:  Gerrit Draisma
*/
proc LogLik( &f,X,theta) ;
    local f:proc ;
    retp(sumc(ln(f(X,theta)))) ;
endp ;
```

5.5. Globale variabelen

Globale variabelen zijn variabelen die in het hoofdprogramma worden gedefinieerd. Het zijn de variabelen die je met `show` te zien krijgt. Procedures gebruiken vaak globale variabelen om ellenlange parameterlijsten te voorkomen, of omdat andere procedures een bepaalde syntax voorschrijven. Het gebruik van globale variabelen in procedures levert ook problemen op: de variabele moet *wel bestaan* op het moment dat de compiler hem tegenkomt. Als de compiler een onbekende naam tegenkomt, neemt hij aan dat het de naam van een procedure is waarvan de definitie nog volgt. Om dat te voorkomen dienen het `declare`-statement en het `external`-statement.

⇒ Dit voorbeeld illustreert het een en ander.

- Schrijf het programma naar `GLOBVAR.E`. Voer het programma uit.
- Welke globale variabele wordt in de procedure `LL` gebruikt?
- Plaats nu `LL` bovenaan in het programma (achter het `new`-statement). Voer het programma weer uit.
- Voeg boven `LL` een `declare` statement toe: `declare _X = 0`.

```
/*
** globvar.e: demo globale variabelen in procedure
*/
new ;
/* genereer x uit n(10,2) */
mu = 10 ;
s2 =  2 ;
n  = 10 ;
_X = mu + sqrt(s2)*rndn(n,1) ;
/* test LL */
theta = seqa(5,2,5)'|(s2*ones(1,5)) ;
LL(theta) ;
/* LL returns for each column of theta
the loglikelihood of the normal pdf given a sample x */
proc LL(theta) ;
local mu, sig2, lnf;
   mu = theta[1,.] ;
   sig2 = theta[2,.] ;
   lnf = -0.5*(ln(pi.*sig2)+((_X-mu)^2)./sig2) ;
   retp(sumc(lnf)) ;
endp ;
```

5.5.1. Het declare-statement. Het declare-statement initialiseert globale variabelen in de compilatiefase. i.e. voordat het programma draait.

De syntax luidt:

```
declare x = <constante> ;  @ standaard @
declare x != <constante> ; @ overwrite existing definition @
declare x ?= <constante> ; @ maintain existing definition @
```

De eerste vorm is standaard: de compiler waarschuwt als de variabele al bestaat. De tweede vorm overschrijft een al gedefinieerde variabele, de laatste vorm handhaaft een reeds gedefinieerde variabele.

Als u procedures met globale variabelen in `.G`-bestanden bewaart, kunt u de `declare`-statements boven in het bestand opnemen (bij de GAUSS-sources staan declare-statements vaak in `.DEC`-files).

5.5.2. Het external-statement. Het external-statement waarschuwt de compiler dat de definitie van een variabele (of procedure) nog volgt. De syntax luidt:

```
external <type> <varname> ;
```

met `<type>` : `matrix`, `string`, `proc`, `fn`, of `keyword`.

(Bij de GAUSS sources staan external-statements in `.EXT`-files)

5.6. Programmaorganisatie en library's maken

5.6.1. Kleine projecten: 1 bestand. Bij kleine projecten kunnen hoofdprogramma en procedures in één bestand gezet worden.

Zet procedures achteraan in het bestand. Dat zorgt er voor dat eventuele globale variabelen die daarin gebruikt worden reeds bekend zijn. Zie hierboven in paragraaf 5.5).

5.6.2. Afzonderlijke procedure(`.G`)-files. Bij grotere projecten is de volgende opzet geschikt:

1. Maak een nieuwe folder (directory) voor het project, bijvoorbeeld met de shortcut My Computer, bovenaan de desktop, of met het Windows-NT commando `mkdir <dirname>`).
2. Schrijf de losse procedures onder hun eigen naam weg (als `*.G`).
3. Verbindt de losse procedures in een hoofdprogramma. Voor GAUSS programma's bestaat geen standaard extensie. De extensie `.E` die GAUSS zelf gebruikt voor de voorbeelden is wel praktisch.

Deze opzet heeft het voordeel dat de verschillende onderdelen onafhankelijk getest kunnen worden, terwijl de structuur van het geheel duidelijk blijft.

Bij deze opzet moet men bij elke procedure duidelijk vermelden welke globale variabelen gebruikt worden. U kunt ze het beste met een `declare`-statement boven in de `.G`-bestanden initialiseren.

⇒ Maak van de de procedure `LL()` uit `GLOBVAR.E` een apart `.G`-bestand.

- Verwijder uit het programma `GLOBVAR.E` de procedure `LL`, en schrijf die weg als `LL.G`.

- Voorzie de procedure van een header en voeg boven die header een declare-statement toe: `declare _X = 0`.
- Voeg aan `LL` eventueel een statement toe om na te gaan `_ X` al gedefinieerd is: Als u `declare _ X ?= 0` gebruikt kunt u daar in de procedure op testen.

Tip: Voor procedures die u bij meerdere projecten wilt gebruiken, kunt u een eigen GAUSS source folder (directory) maken. In het configuratie bestand `GAUSSI.CFG` geeft u op dat GAUSS ook in die folder (directory) naar procedures zoekt.

5.6.3. Organisatie met library files. Dit is de opzet die GAUSS zelf voor haar *runtime library* gebruikt.

1. Procedures m.b.t. een bepaald onderwerp staan bij elkaar in een `.SRC`-bestand gedefinieerd.
2. Globale variabelen worden in een `.DEC`-bestand gedeclareerd.
3. Bij globale variabelen behorende external statements staan in een `.EXT`-bestand.
4. Een library- (`.LCG`)-bestand vertelt de compiler waar procedures en variabelen gedefinieerd worden. (De extensie `.LCG` stamt uit de tijd dat niet alle versies van GAUSS complexe getallen ondersteunden.)
5. Tenslotte wordt een procedure meegeleverd om globale variabelen te 'resetten'. Zo reset `gausset` de globale variabelen van de GAUSS-runtime library; en `graphset` die van de graphics library.

Om gebruik te maken van één of meer GAUSS-library's neemt men een `library`-statement op in het programma.

```
library pgraph, nroptmum ; @ laad libraries @
library ; @ toont welke libraries geladen zijn @
```

De libraries `GAUSS` en `USER` worden automatisch geladen. U kunt dat uitschakelen in het scherm dat u met `<Alt-O(ptions)><P(rogram)>` bereikt vanuit de 'command-mode' of de 'edit-mode' van GAUSS.

⇒ Hierbij een kleine demonstratie hoe GAUSS procedures en variabelen zoekt. De globale variabele `_fcmptol` wordt door de fuzzy vergelijkingen `feq()`, `fne()` etc. gebruikt.

- Zet in het scherm met GAUSS-Debug options dat u bereikt met `<Alt-(O)ptions><D(ebug)>` de `compiler file trace` functie aan. Voer onderstaande regels uit, met en zonder de tweede regel.

```
new;
@ _fcmptol = 0.01 ; @
feq(0,0.001) ;
```

5.6.4. Zelf een library maken. Men kan als men GAUSS op een eigen PC heeft geïnstalleerd zelf een library maken met het GAUSS-commando `lib`. De syntax luidt:

```
lib <libfile> <sourcefile> ;
```

waarin `<libfile>` de naam is van de library die men wil maken, en `<sourcefile>` de naam van het bestand (`.DEC` of `.SRC`) waarvan men de globale variabelen en procedures aan de library wil toevoegen.

HOOFDSTUK 6

Invoer / Uitvoer (Input/Output)

6.1. ASCII-Bestanden: wegschrijven en inlezen

ASCII-bestanden kan men aanmaken door scherm-uitvoer naar een file te schrijven. Belangrijke statements zijn:

```
OUTPUT [file = <filename>] [ON|RESET|OFF]
FORMAT
SCREEN [ON|OFF]
```

⇒ Hieronder wordt een matrix X naar een ASCII-file weggeschreven:

```
new ;
m = 5 ; n = 3 ;
X = rndn(m,n) ;
format /rd 8,3 ; outwidth 132 ;
output file = x.asc reset ;
X ;
output off ;
```

Met het `load`-statement kunnen ASCII-bestanden ingelezen worden. Zie `<alt-H><S> Loading`.

⇒ Hieronder wordt de matrix weer ingelezen. Let op de dimensies.

```
load y[] = x.asc ;
x ; y ;
end ;
```

6.2. Invoer vanaf het toetsenbord (keyboard)

Meestal worden kleine matrices in programma's gedefinieerd, en grote van schijf ingelezen. Bij interactieve programma's kan men onderstaande procedures gebruiken:

`con()`: voor het invoeren van kleine matrices (bijv. parameters om de output van een programma te wijzigen),

`editm()`: voor het wijzigen van matrices.

- Met de pijltjes kunt u de matrix doorlopen.
- Met `<getal>,<getal>` ... `<enter>` kunt u één of meer elementen (rij voor rij) wijzigen.
- Met `;` kunt u het editen stoppen.

Daarnaast kunt u de utility `medit` gebruiken, die een soort spreadsheet interface geeft (bij GAUSS wordt een demo-versie meegeleverd voor matrices tot 64Kb).

⇒ Voorbeeld:

```
x = con(3,2) ;
x ;
x = editm(x) ;
x ;
```

6.3. Binaire opslag van matrices

Een matrix die u voor toekomstig gebruik wilt bewaren kunt u ook binair opslaan. Dit gebeurt snel en gecomprimeerd. Belangrijke statements zijn:

`save :` bijv.
`save x, y, M ; :` bewaart matrices in resp. `X.FMT`, `Y.FMT` en `M.FMT`;
`save <fname> = x ; :` bewaart matrix `x` in `<FNAME>.FMT`.

`loadm:` bijv.
`loadm x, y, M ; :` (laadt `x`, `y`, `M` uit resp. `X.FMT`, `Y.FMT` en`M.FMT`;
`loadm x = <fname> ; :` laadt `x` uit `<FNAME>.FMT`).

`load:` kan gebruikt worden in plaats van `loadm`.

Let op: Gebruik bestandsnamen *zonder extensie* voor binaire opslag!

⇒ Bekijk het volgende voorbeeld: Hieronder wordt een reeks van 1000 random trekkingen uit een normale verdeling bewaard, zodat men in simulatie-experimenten dezelfde reeks kan gebruiken.

```
/* voer deze regels door zelf intypen INTERACTIEF in command-mode uit */
new;
x = rndn(1000,1) ;
SAVE tmp = x ;
shell;
  dir tmp.* ; @ voer dit in een "DOS" window" uit om te controleren @
              @ of tmp.fmt is aangemaakt @
  exit        @ met dit commando keer je terug naar GAUSS @

/* en dan dit */
new ;         @ maak het geheugen van GAUSS leeg,  @
              @       verwijder oude variabelen    @

LOADM x = tmp ;
r = reshape(x,15,3) ;
shell del tmp.fmt ; @ met het "del" commando van het besturingssysteem @
                    @ wordt het bestand tmp.fmt weer verwijderd @
```

6.4. GAUSS-datasets

Data slaat u het beste op als GAUSS-datasets. GAUSS-datasets bieden het voordeel van *binaire* (snelle) opslag zonder beperkingen aan de omvang en het bewaren van *variabelenamen* bij de data. Veel standaard GAUSS-procedures accepteren de naam van een dataset als parameter (zie bijv. `ols` en `dstat`).

Een GAUSS-dataset bestaat uit twee bestanden [1]: een `.DAT` bestand met de data (binair) en een `.DHT` bestand met een beschrijving daarvan.

Let op: Gebruik namen (max. 16 letters/tekens) *zonder extensie* voor GAUSS dataset-bestanden! Gebruik voor namen van variabelen (max. 8 letters) alleen hoofdletters.

6.4.1. Kleine datasets. Voor kleine datasets is een aantal procedures beschikbaar, dat een dataset in zijn geheel laadt of weg schrijft. Wat klein is, hangt daarmee van de versie van GAUSS af: voor GAUSS *Light* is dat 64Kb, voor de overige versies wordt dat bepaald door het beschikbare geheugen.

`SAVED()`: bewaar een matrix als dataset,
`LOADD()`: lees een dataset in een matrix in,
`GETNAME()`: lees namen van de variabelen in.

⇒ Voorbeeld. Hieronder wordt een dataset gegenereerd met

$$y = 375 + 100x_1 + 100x_2 + u$$

waarin y het inkomen, x_1 de leeftijd en x_2 het geslacht. Daarna wordt de runtime procedure `ols` aangeroepen om een regressie uit te voeren.

```
new ;  m = 15 ;
names = { AGE SEX WAGE } ;
x      =(21+5*rndn(m,1))~(rndu(m,1).>0.5);
b      = { 375, 100, 100 } ;
y      = (ones(m,1)~x)*b + 100*rndn(m,1) ;
"x~y " x~y ; wait ;
if saved(x~y,"mydata",names) ;
   "weggeschreven naar mydata" ;
   call ols("mydata","SEX",0) ;
else ;
   errorlog "schrijf fout: geen data weggeschreven "  ;
endif ;
```

Let op: GAUSS gebruikt hoofdletters voor de namen van numerieke variabelen en kleine letters voor de namen van karakter-variabelen.

⇒ Voeg hieronder de regels toe waarmee de dataset "mydata" weer ingelezen wordt ; lees ook de namen van de variabelen in ; print een tabel, met de namen boven de kolommen.

```
new ;
```

[1] Bij andere versies van GAUSS is alle info in één bestand aanwezig. Onder Unix bestaat een utility `TRANSDAT` voor het uitwisselen van binaire matrix files tussen Unix en DOS.

6.4.2. Grote datasets. Voor grotere datasets gebruikt men de basisprocedures voor dataset input/output. `Do`-loops zijn nodig om grote matrices *in delen* in te lezen en/of weg te schrijven. Alle `i/o`-operaties zijn gebonden aan een zgn. file-handle (de variabele `fh` hieronder). Die bevat het adres van de procedures en buffers die het operating system gebruikt voor de `i/o` naar het betreffende bestand.

NB: Bij voldoende intern geheugen en een professionele versie van GAUSS heeft men deze procedures niet snel nodig en volstaan de procedures uit 6.4.1.

Belangrijke Statements zijn:

```
CREATE fh = <filename> with <varnames>,<cols>,<type> ;
OPEN fh = <filename> [for read|append|update] ;
CLOSE(fh) ;
CLOSEALL [<list of file handles>] ;
n = SEEKR(fh, r) ;
X = READR(fh, r) ;
n = WRITER(fh,X) ;
y = EOF(fh) ;
```

De `ATOG`-utility gebruikt men om ASCII-bestanden om te zetten naar GAUSS-datasets.

Met `dataloop` kan men op een eenvoudige manier variabelen van *zeer grote* GAUSS datasets selecteren en bewaren. Zie de online help (`Data Transformations` voor voorbeelden. Om `dataloop` te kunnen gebruiken moet wel de "Dataloop Translationïn de GAUSS-Program Option, te bereiken met `<alt-O><P>`, worden aangezet!

⇒ Voorbeeld: Hieronder wordt een dataset aangemaakt die groter is dan 64 K.
⇒ Probeer de dataset met `LOADD()` weer in te lezen. Gebruik de procedure `DSTAT()` om een aantal statistische grootheden van de dataset af te drukken. Bekijk hoe daarin met grote matrices wordt omgegaan.

```
new;
let vnames = V1 V2 V3 ;
fname = "mydata" ;
/* maak een dataset met 3 kolommen,
   single precision (4 bytes/var)
   Zo neemt de dataset minder geheugenruimte in
   dan bij de standaard double precision (8 )  */
create fh = ^fname with ^vnames, 3, 4 ;
/* controleer of dataset geopend is */
if fh == -1 ;
   errorlog "create mislukt" ;
   closeall ;
end ;
endif ;
obs = 0 ; nr = 100 ;
do while (obs < 5000) ;
   /* maak data aan in blokken van 100 */
   x = rndn(nr,3) ;
   /* schrijf blok weg.
   writer() telt aantal weggeschreven regels */
```

```
   obs = obs + writer(fh, x) ;
   obs ;
endo ;
close(fh) ;
/* eenvoudig bewerken van grote GAUSS datasets met dataloop */
dataloop ^fname mydataselection; @ zorg dat de "translator" bij de  @
                                 @ Program Options aanstaat!        @
    drop V1;
    make V4 = V2 + V3 ;
    keep V2 V3;
endata ;
```

6.5. Invoer en uitvoer van datasets in andere formaten.

GAUSS kan ook datasets inlezen uit en wegschrijven naar andere veelgebruikte formaten voor bijvoorbeeld spreadsheet-programma's als Excel. Zoek in de GAUSS Browser naar `export*` en `import*` voor nadere informatie.

⇒ Hieronder wordt een dataset aangemaakt en weggeschreven in Excel-formaat.

⇒ Voeg hieronder de regels toe waarmee de dataset `exceldat.xls` weer ingelezen wordt in een GAUSS dataset; lees ook de namen van de variabelen in! Zoek de betreffende informatie op met het zoekwoord `importf` in de GAUSS-browser. De Browser verwijst vervolgens naar het bestand `dataxchn.src`.

⇒ Bereken vervolgens gemiddelde, variantie, minimum en maximum per variabele met de procedure `dstat`.

```
new ;  m = 15 ;
let names = { AGE, SEX, WAGE };
x      =(21+5*rndn(m,1))~(rndu(m,1).>0.5);
b      = { 375, 100, 100 } ;
y      = (ones(m,1)~x)*b + 100*rndn(m,1) ;
"x~y " x~y ; wait ;
   fname = "x:\\exceldat.xls";
   if export(x~y,fname,names);
   "weggeschreven naar exceldat.xls" ;
else ;
   errorlog "schrijffout: geen data weggeschreven,
   sluit programma dat bestand reeds gebruikt "  ;
endif ;
@  voeg hier uw regels toe! @
```

HOOFDSTUK 7

Hoofdpunten

7.1. Programmaopzet

7.1.1. Werkomgeving.

- Controleer de `gauss_src` waarde in de file `GAUSSI.CFG`. Deze waarde bepaalt waar GAUSS naar procedures zoekt. Procedures (`.G`-files) in de *current folder (directory)* vindt GAUSS automatisch. Controleer de naam van de *current folder* met het commando `cdir(0)`.
- Controleer de `STARTUP`-file. Deze file wordt automatisch uitgevoerd. Een `library pgraph` statement maakt de grafische procedures automatisch beschikbaar.
- Geef elk project een eigen folder (directory). Zie paragraaf 5.6).

7.1.2. Stijl.

- Zet niet meer dan één statement op een regel. Dit vereenvoudigt debuggen en commentaar geven.
- Zorg voor overzichtelijke en begrijpelijke uitvoer: voorzie tabellen en grafieken van titels en labels.
- Gebruik duidelijke namen voor variabelen.

7.1.3. Initialisatie. Zet bovenaan in uw programma:

- alle probleemdimensies,
- format-statements, zie paragraaf 4.4),
- globale variabelen (zie paragraaf 5.5) en
- gebruikte library's (zie paragraaf 5.6).
- In de ontwikkelfase kan het handig zijn, om een `new`-statement op te nemen boven in uw programma. Dat is wel veilig. Bij het opnieuw uitvoeren van het programma worden alle procedures opnieuw gecompileerd. Dat kost tijd, maar u weet dan zeker dat GAUSS de laatste versie gebruikt van uw procedures.

7.1.4. Procedures.

- Splits uw programma op in deelprocedures (`.G`-files, zie paragraaf 5.6).
- Schrijf voor elke procedure eerst de documentatie (de header), waarin u vastlegt wat de procedure moet doen (zie bijv. de `loglik` procedure in paragraaf 5.4).
- Controleer elke procedure apart op kleine problemen.
- Vermijd het gebruik van globale variabelen in procedures. Indien nodig, gebruik dan variabelenamen met een underscore.

7.1.5. Einde programma.

- Sluit het programma af met een `end`-statement.
- Reset globale variabelen. Reset de globale graphics variabelen met `graphset` en eventuele *runtime* globals met `gausset`.

7.2. Debuggen (Fouten zoeken en verwijderen)

- Let alleen op de eerste foutmelding, zie paragraaf 1.6).
- Vraag met `show [/p] [/m] [<name>]` op welke variabelen er in het geheugen van GAUSS zijn. Zie paragraaf 1.5).
- Pas bij veranderingen eerst de documentatie en daarna de rest van de programmacode aan.
- Verwijder oude versies van foute procedures uit het geheugen met `delete <procname>` of `new`.
- Controleer elke procedure *apart* op *kleine* problemen.

7.3. Efficiënt programmeren

- Lees eerst de hele opdracht door. Maak een voorlopig plan voordat je programmeren gaat. Dit voorkomt onnodige verbouwingswerkzaamheden.
- Maak gebruik van de standaard GAUSS-procedures of ideeën of onderdelen daaruit. Gebruik de `help`-functies (zie paragraaf 1.3) of de `GREP`-utility (zie app. A) om onderdelen van uw gading in de bijgeleverde source-files te zoeken.
- Bestudeer de voorbeelden uit de `EXAMPLES` folder (directory). `PXY.E` toont bijvoorbeeld alle toeters en bellen van de (publication quality) graphics-library. In de lokale source folder `LOCALSRC` vindt u het programma `TOBIT.E` voor de analyse van een tobit-model, met bijbehorende datasets en procedures voor optimalisatie, grafieken en output in tabellen.
- Controleer Uw programma eerst op kleine problemen vóór u grote problemen en/of datasets aanpakt.

7.4. Efficiënt rekenen

- Vermijd `if`-statements, `do`-loops en `for`-loops zoveel mogelijk. Gebruik matrix- of `ExE`-operatoren (zie 3). `for`-loops zijn sneller dan `do`-loops.
- Schrijf eigen procedures ook zodanig dat ze zonder loops aan te roepen zijn. Een procedure die met een vector als invoer een scalair oplevert, is meestal gemakkelijk zo te schrijven dat hij met een matrix een kolomvector oplevert. Gebruik de standaardprocedure `sumc()` hierbij als voorbeeld.
- Start simulatie-experimenten klein. Gebruik `hsec` om de verwachte rekentijd voor grote experimenten te bepalen.
- Gebruik GAUSS-datasets en -matrices om (tussen-)resultaten te bewaren. Vermijd ASCII-files.
- Onderzoek welk onderdeel van uw programma de meeste rekentijd vergt. Gebruik hiervoor bijvoorbeeld `hsec`.

7.5. Data-analyse en grote datasets

- Converteer data zoveel mogelijk naar GAUSS datasets.
  - In GAUSS schrijft men datasets met `saved()` (makkelijk) of `writer` (flexibel).

  - Met `ATOG` kan men buiten GAUSS zonder geheugenrestricties ASCII-datasets naar GAUSS-datasets transformeren.
  - In GAUSS leest men data-sets met `loadd()` (makkelijk), of `readr` (flexibel).
- Controleer **ALTIJD** of data goed zijn ingelezen. Gebruik `xy()`, `show /m`, `dstat()`, vooral na commando's als `reshape()`. Gebruik alleen hoofdletters in namen van variabelen in GAUSS-datasets.
- Let op missing values. Gebruik `miss()`, `missrv()` om missing values op te sporen of te vervangen.
- Gebruik `ATOG` voor het converteren van *grote* ASCII-bestanden naar GAUSS-datasets.
- Gebruik `dataloop` voor het transformeren en selecteren van variabelen in grote GAUSS-databestanden.
- Zorg voor voldoende intern geheugen. Vermijd het gebruik van "virtueel" geheugen. Hef de gebruiksbeperking van "extended or expanded memory" zoveel mogelijk op door de waarde van de variabele `max_workspace` in de configuratie-file `GAUSS.CFG` aan de configuratie van uw computer aan te passen.
- Laat het gebruik van grote vectoren en matrices toe door aanpassen van de globale variabele `__maxvec`. Dit helpt natuurlijk niet bij de *Light* versie van GAUSS.

OPGAVE 1

Syntax en operatoren

Deze opgave is uit te voeren na hoofdstuk 3.

Doel van deze opdracht is de demonstratie van de standaard-theorie van het lineair model met twee variabelen. De gegevens en formules van deze opdracht zijn ontleend aan: Stewart, J. (1991), Econometrics, Pilip Allen, New York.

DATASET 1: (pag. 18)
Consumers Expenditure and Income UK, 1959-87

Year	RCONS	RPDI	PCONS
1959	118.547	124.694	0.137464
1960	123.119	132.91	0.139004
1961	125.848	138.462	0.143093
1962	128.69	139.964	0.148395
1963	134.838	146.279	0.150952
1964	138.985	152.504	0.156369
1965	141.107	155.66	0.164088
1966	143.617	159.072	0.170627
1967	147.162	161.455	0.175052
1968	151.271	164.266	0.183452
1969	152.112	165.769	0.193607
1970	156.336	172.246	0.204905
1971	161.208	174.463	0.222563
1972	171.052	189.069	0.237051
1973	179.852	201.034	0.256589
1974	177.233	199.421	0.300164
1975	176.273	200.353	0.371435
1976	176.853	200.029	0.429634
1977	176.016	195.606	0.493631
1978	185.95	209.864	0.538957
1979	193.794	221.673	0.612258
1980	193.806	224.885	0.711516
1981	193.832	222.254	0.792263
1982	195.561	221.709	0.861854
1983	204.318	227.931	0.903587
1984	207.927	232.426	0.949824
1985	215.267	237.802	1
1986	226.839	244.797	1.043718
1987	238.46	252.185	1.083750

```
RCONS: Real Consumer Expenditure (BP billion, 1985 prices)
RPDI : Real Personal Disposable Income       ,,
PCONS: Implicit Deflator, consumers Expenditure (1985=100)
```

⇒ De opdracht moet resulteren in een GAUSS-programma, met de naam `OPG1.E`. Gebruik de `help` van GAUSS, om de nodige procedures te vinden. Het programma moet met `run opg1.e` kunnen opstarten.

⇒ Maak onderstaande kop kompleet en neem hem op in uw programma.

```
/*
** OPG1.E
**
** Doel:
**
** Datum:
**
** Auteur:
**
*/
```

Met x = RPDI, y = RCONS, kunnen de parameters van het volgende model worden geschat:

$$y_t = \alpha + \beta x_t + u_t, \quad t = 1960 \ldots 1984 \tag{1}$$

⇒ Vorm de matrix `DSET1` m.b.v. bovenstaande tabel: Zet relevante informatie in commentaar, en voeg de nodige {`,`}-tekens toe.

⇒ Maak vectoren t, x, en y, gebruikmakend van de indexoperatoren.
(juiste kolommen / rijen).

⇒ Gebruik de procedure `xy()` om x en y tegen de tijd uit te zetten, en om een spreidingsdiagram van x tegen y te maken. Voorzie de grafiek van uitleg m.b.v. de procedures `title()`, `xlabel()` en `ylabel()`.

1.1. Som-notatie

Gebruikmakend van de gegevens van 1960 t/m 1984, kunnen de volgende zaken berekend worden:

$$\begin{aligned} n &= 25 \\ \sum x_t &= 4609.304 \\ \sum y_t &= 4136.760 \\ \sum x_t^2 &= 873878.4 \\ \sum y_t^2 &= 700535.43 \\ \sum x_t y_t &= 782274.14 \end{aligned} \tag{2}$$

en

$$b = \frac{[\sum x_t y_t - (\sum x_t)(\sum y_t)/n]}{[\sum x_t^2 - (\sum x_t)^2/n]} \tag{3}$$
$$= 19570/24051 = 0.813717$$
$$a = (\sum y_t - b \sum x_t)/n = 15.4436$$

⇒ Laat het programma de onder (2) en (3) genoemde sommen en grootheden op het scherm afdrukken. Maak gebruik van de procedure `sumc`.

1.2. Matrix-notatie

In matrix-notatie luidt dit model:

$$y = X\beta + u, \tag{4}$$

waarbij X bestaat uit een kolom 1-en en de vector x, en $\text{var}(u) = \sigma^2 I$, met als schatter van β:

$$b = (X'X)^{-1}X'y \tag{5}$$

en als voorspelde waarde van y:

$$\hat{y} = Xb \tag{6}$$

⇒ Vorm de matrix X en laat het programma $X'X$, de inverse, $X'y$ en b op het scherm tonen.

⇒ Maak nu een grafiek waarin y tegen x worden uitgezet, en bovendien de regressielijn wordt weergegeven.

1.3. Simulatie

Volgens de theorie van het lineair model heeft de schatter b een bivariaat-normale verdeling:

$$b \sim N(\beta, \sigma^2(X'X)^{-1}) \tag{7}$$

Dit is met een eenvoudig simulatie-experiment te demonstreren. We gebruiken

$$\beta = b \quad \text{en} \quad \sigma^2 = s^2 = (y - \hat{y})'(y - \hat{y})/(n - 2) \tag{8}$$

als parameters.

⇒ Bereken $\sigma^2 = s^2$.

⇒ Genereer nu een matrix van 200 y-vectoren en bereken de bijbehorende schattingen, b.

⇒ Bereken de covariantiematrix van de schattingen, en vergelijk deze met de theoretische covariantiematrix.

⇒ Maak een spreidingsdiagram van b_1 vs. b_2. Maak histogrammen van b_1 en b_2. Voorzie de grafieken van titels en labels.

OPGAVE 2

Een procedure met output

Deze opgave is uit te voeren na hoofdstuk 4.

Doel van deze opdracht is het schrijven en documenteren van een simpele procedure en het produceren van geformatteerde output. Eén en ander aan de hand van een simpele OLS-procedure. De procedure wordt toegepast op de data van opgave 1. (zie Stewart, J. (1991), *Econometrics*, Philip Allen, New York.)

⇒ Onderstaande header definieert de input en output van 'your ols' procedure. Vul de ontbrekende gegevens in, en neem de header op in de procedure die u gemaakt hebt in pargraaf 2.5 en pas deze procedure aan.

```
/*
**  MYOLS.G
**
**  purpose :
**
**  format  :  {b,s2} = myols(y,X,namen)
**
**             y      : (t x 1) afhankelijke variabele
**             X      : (t x k) verklarende variabelen
**             namen  : ((k+1) char. vector van variabele namen.
**
**  output  :
**             b      : geschatte coefficienten
**             s2     : geschatte variantie van de storingsterm
**
**  globals :
**
**  remarks :
**  date    :
**  author  :
*/
```

Als voorbeeld van de gewenste output van de procedure dient onderstaande tabel uit Stewart (table 2.2 Linear consumption Function. Data: Consumer Expenditure (`RCONS`) and Income (`RPDI`), 1960–84. Zie opg. 1).

```
Dependent variable is RCONS

Regressor     Coefficient   St. Error     t-value
-----------   -----------   -----------   -----------
```

```
INTER            15.44361       2.50154       6.17364
RPDI              0.81372       0.01338      60.81653

Degrees of freedom 23, from 25 observations

Residual SS      99.02977
Total SS         16024.10
Dist. variance   4.305642
```

⇒ Bewaar de procedure op schijf als `MYOLS.G`. Controleer of je met het help-systeem van GAUSS, `help` over de procedure kunt opvragen.

⇒ Schrijf nu een programma `OPG2.E`, waarin de procedure wordt gebruikt. Het commando `run opg2.e` moet bovenstaand resultaat opleveren. Voor de data: zie opgave 1.

OPGAVE 3

De log-aannemelijkheidsfunctie als voorbeeld

Deze opgave is uit te voeren na hoofdstuk 5

In deze opgave bestuderen we de log-aannemelijkheidsfunctie van een simpel lineair model. U schrijft een procedure, die de log- aannemelijkheidsfunctie van het model van opgave 1 bepaalt. De procedure moet zó geschreven worden, dat zij als invoerparameter van een specifieke minimaliseringsprocedure kan worden gebruikt.

U gebruikt de minimaliseringsprocedure `nelder`, die bij de intro-bestanden aanwezig is. De procedure berust op het 'amoebe'- of simplex-algoritme van Nelder en Mead (zie Press et al (1985), Numerical Recipes, Cambridge University Press). Het is (zoals de meeste optimaliseringsprocedures) een minimaliseringsprocedure. De procedure die u als parameter meegeeft aan de minimaliserings- procedure, moet dus de *min*-log-aannemelijkheid uitrekenen.

⇒ Bestudeer eerst de procedure `nelder`, gedefineerd in het bestand `NELDER.G`. Bekijk welke eisen de procedure stelt aan de te minimaliseren functie.

⇒ Schrijf een procedure `minLL`(θ), die de min-loglikelihood van het lineair model van opgave 1 berekent, met $\theta = (\beta', \sigma^2)'$:

$$\begin{aligned} \texttt{minLL} &= -\log L(\beta, \sigma^2) \\ &= -\log\left\{(2\pi\sigma^2)^{-n/2}\exp\{\frac{-(y-X\beta)'(y-X\beta)}{2\sigma^2}\}\right\} \end{aligned} \tag{9}$$

- Een aantal variabelen uit formule (9) kan alleen als globale variabele aan de procedure doorgegeven worden. Welke? en Waarom?
- Met één kolom-vector θ als invoer, moet `minLL` een scalair opleveren, en met een matrix van meerdere kolommen als invoer, een kolomvector (één uitvoerwaarde voor elke kolom van de invoer). Dat is de standaard in GAUSS voor procedures die een vector opleveren. (Probeer bijvoorbeeld de procedure sumc() uit op een matrix).
- De `nelder` procedure weet niet dat de variantie positief moet zijn. Programmeer uw `minLL` functie zodanig dat hij een 'slechte' (=hoge) waarde teruggeeft als de parameters een niet-toegelaten waarde aannemen. (boetefunctie).

⇒ Welke globale variabele bepaalt het stopcriterium van de `nelder` procedure? Kies daarvoor een geschikte waarde.

⇒ Schrijf tenslotte een programma waarin de minimaliserings- procedure `nelder()` en uw procedure `minLL()` gebruikt worden om de maximale-aannemelijkheidschattingen van de parameters van het lineaire model uit opgave 1 te bepalen. Uw programma dient het volgende te doen:

1. Het toont de gewone-kleinste-kwadratenschatting van de parameters (de GKK-schatting).
2. Het toont de maximale-aannemelijkheidschatting (ML-schatting) en de verschillen tussen beide schattingen.

3. Het toont een grafiek van de log-aannemelijkheid tegen elk van de parameters in de omgeving van het optimum. Gebruik `begwind`, `endwind`, en `window()` om de grafieken op 1 scherm te zetten. Zie bijvoorbeeld het voorbeeldprogramma `PXY2.E` uit de `examples` folder van GAUSS.

⇒ Vanzelfsprekend voorziet u programma en procedures van de standaard documentatie.

BIJLAGE A

INSTALLATIE EN CONFIGURATIE

A.1. Installatie van diskette

De Light versie van GAUSS voor Windows (32 bit) wordt gedistribueerd op 3 diskettes. Start de installatie met `setup.exe` van disk 1. Let op: installeer alleen in een folder met een korte naam (minder dan 9 karakters) zonder spaties!

GAUSS leest haar configuratie parameters in uit het bestand `GAUSS.CFG`. Wijzigingen in de configuratie moet U hier aanbrengen. Het bestand is goed gedocumenteerd en kan met een editor (bijvoorbeeld Notepad) gewijzigd worden.

De file `PQGRUN.CFG` beschrijft de 'graphics' configuratie. Als U rechtstreeks GAUSS grafieken wilt printen kunt u hier print-opties opgeven.

A.2. Installatie van lokale procedures

GAUSS installeert alle source files in de `SRC`-subdirectory. Voor uw eigen of lokale procedures kunt U een aparte folder aanmaken. Vervolgens moet u die folder toevoegen aan de `src_path` variabele in `GAUSS.CFG`. De procedures kunnen worden toegevoegd aan een *library*-bestand met het GAUSS commando `lib`. Bijvoorbeeld:

```
lib user nroptmum.src /n /b
```

voegt procedures uit de file `NROPTUM.SRC` toe aan de standaard user library `USER.LCG`.

A.3. Installatie hands-on intro

De `INTRO***.E`-files en `**.G`-files (met alle opdrachten en code segmenten) die bij deze introductie horen kunt u kopiëren naar een eigen `INTRO-` folder en deze folder *current folder* maken in GAUSS.

A.4. Gauss mailing list

Er bestaat een actieve mailing list waar U met vragen over GAUSS terecht kunt. U kunt zich abonneren met een Email-bericht aan `majordomo@eco.utexas.edu` met als enige tekst de regel

```
subscribe
```

Meer internet-informatie over GAUSS-applicaties en andere software voor econometrie vindt u op de internetsite van het Econometrisch Instituut:

```
http://www.eur.nl/few/ei/links
```

A.5. Andere versies van gauss

Deze introductie beschrijft de *studenten-* of *light* versie van GAUSS `3.2.34`. Deze versie is geheel gelijk aan de standaard-versie, afgezien van een beperking op de maximale grootte van matrices tot `64 Kb` of ±9000 elementen.

Er bestaat ook een Unix versie van GAUSS 3.2. Voor de liefhebbers is er ook nog een DOS-versie 3.2.16 beschikbaar, die goed is gedocumenteerd.

A.6. Conversie van gauss-plaatjes naar Windows-formaten

GAUSS-plaatjes kunnen makkelijk in Windows-applicaties worden ingelezen met het programma `PlayW` dat tegenwoordig met GAUSS wordt meegeleverd. Een handleiding vindt u op de WWW-server van de FEW:

`http://www.few.eur.nl/few/services/ia/fase3/software/gaussplay.htm`