Københavns Universitet
Matematisk Institut

September 1985.

INTRODUKTION TIL DATALOGI A FOR MATEMATIKERE

1.HENSIGTERNE MED KURSET.

Kurset er tiltænkt studerende på matematik-fagkombinationer, som har behov for et afrundet støttefagsmodul i datalogi. Det er normeret til halvdelen af en fuldtidsstudieindsats for efterårssemestrets vedkommende og en sjettedel fuldtidsstudium for forårssemestrets vedkommende, og det kan således naturligt følges samtidig med kurset Matematik 1. Der skal afleveres opgaver i løbet af kurset til godkendelse, og kurset afsluttes med en skriftlig prøve, hvori der gives karakter.

Kurset forudsætter, navnlig i opgavevalget, matematisk kunnen og interesse hos de studerende, som herved får mulighed for at anskue matematik fra nye synsvinkler og lære at bruge EDB som hjælpemiddel ved løsning af problemer af matematisk karakter. Men det tilstræbes også, at deltagerne får en bredere forståelse af vigtige datalogiske begreber og datalogiens betydning.

Efterårssemestret indeholder en almen indføring i databehandling, en grundig introduktion til programmering ved hjælp af programmeringssproget PASCAL, samt praktisk arbejde med trinvis udvikling af et program til løsning af lineære programmeringsproblemer. Ved dette praktiske arbejde benyttes Universitetets IBM 4341 anlæg.

For at sikre, at deltagerne i kurset sættes i stand til i praksis at udnytte de mange muligheder EDB giver, vil der i forårssemestret blive arbejdet med en række småprojekter, der delvis skal udføres på andre regnemaskineanlæg (RECKU og Mikrodatamater) og vil benytte andre programmeringssprog (f.eks. FORTRAN). Disse projekter kan f.eks. bestå i udnyttelse af grafiksystemer og databasesystemer eller programmering ved hjælp af programbiblioteker til numeriske beregninger.

2. UNDERVISNINGENS FORM.

I efterårssemestret er der ugentlig 2x2 timers forelæsninger: tirsdag kl. 12-14 og torsdag kl. 14-16 i aud. 4 på H.C.Ørsted institutet, samt 1x3 timer øvelser: mandag kl. 15-18, onsdag kl. 15-18 eller fredag kl.14-17, på hold med omkring 20 deltagere.

Første forelæsning finder sted tirsdag d. 3 september.

Tilmelding til øvelseshold sker ved forelæsningerne i første uge.

Forelæsningerne omfatter en gennemgang af dele af lærebogsmaterialet, først og fremmest PASCAL lærebogen og stof med baggrundsviden om EDB. Øvelserne består af arbejde med opgaver stillet i forbindelse med stoffet fra forelæsningerne. Hertil kommer praktisk arbejde med programmeringsopgaver dvs. programskrivning samt indtastning og afprøvning af programmer ved terminalerne. Problemer i forbindelse med programmeringsopgaverne kan diskuteres ved øvelserne. I begyndelsen skal der afleveres programmeringsopgaver hver uge, senere mindre hyppigt. De første fire opgaver, der skal være løst (og godkendt) inden efterårsferien, følger efter denne introduktion.

Undervisningen i forårssemestret omfatter kun øvelser til støtte for projektarbejdet (med uændrede øvelseshold).

UNDERVISNINGSMATERIALE.

(PG) Per Gjerløv.
VM/370 - en introduktion.
Teknisk Forlag.
1984. (Pris ca. kr. 90 - 10% til studenter.)

(WG) C.William Gear.
Programming in PASCAL.
Science Research Associates.
1983. (Pris ca. kr.250 - 10% til studenter.)

(PO) Politikens EDB bog.
Politikens Forlag.
1985. (Pris kr. 168 - 10% til studenter.)

(LP) Noter om lineær programmering.
Matematisk Institut.
1985.

Her giver (PG) den nødvendige viden til brug af Universitetets IBM 4341 anlæg. Bogen vil ikke blive gennemgået, men kendskab til de første 6 afsnit er en forudsætning for at kunne arbejde ved terminalerne. Det anbefales derfor at læse bogen igennem inden introduktionsdemonstrationen (se nedenfor).

Bogen (WG) indeholder en grundig indføring i PASCAL, og den vil blive gennemgået ved forelæsningerne.

Bogen (PO) skal benyttes som kilde til baggrundsviden om EDB. Den forudsættes læst ved eksamen, men vil ikke blive 'gennemgået'.I forbindelse med forelæsningernes omtale af datalogiske grundbegreber, vil der blive henvist til den, og den vil blive suppleret med smånoter, der direkte tager sigte på forårssemestrets

projekter, og det anbefales de studerende at læse bogen igennem i løbet af efterårssemestret.

Noterne (LP) er en introduktion til emnet LINEÆR PROGRAMMERING, til brug for løsning af programmeringsopgaverne i sidste halvdel af efterårssemestret.

Derudover er der adgang til håndbøger ('manualer'), med vejledning og referenceoplysninger til brugen af IBM 4341-maskinen og dens programmer. Disse håndbøger er anbragt i terminalrummet, lokale A109, og:

```
+-----------------------------------+
!          DISSE MANUALER           !
!      SKAL UNDER ALLE FORHOLD      !
!     FORBLIVE I TERMINALRUMMET     !
+-----------------------------------+
```

3. FORELØBIG PLAN FOR EFTERÅRETS UNDERVISNING.

Planen gælder gennemgangen ved forelæsningerne. Emnerne er stort set kapiteloverskrifter fra det benyttede undervisningsmateriale. F.eks. betyder (WG2) kapitel 2 i PASCAL-lærebogen (WG).

Der holdes ikke sædvanlige øvelser i forbindelse med første uges forelæsninger. I stedet afholdes en introduktionsdemonstration af terminalerne for mindre grupper. Afsnit 4 i denne introduktion er materiale til brug ved demonstrationen. Tilmelding til introduktionsdemonstrationen foretages ved forelæsningerne i første uge.

UGE 36. Overblik over emnet.
Anvendelse af universitetets IBM 4341.(PG).

UGE 37. Introduktion til PASCAL.(WG0).
Data.(WG1).

UGE 38. Programstrukturer.(WG2&3).

UGE 39. "Arrays".(WG4).
Introduktion til lineær programmering.(LP).

UGE 40. Konstruktion og afprøvning af programmer.(WG5).
Gennemgang af LP-programmeringsopgaven.(LP).

UGE 41. Underprogrammer.(WG7).

EFTERÅRSFERIE.

UGE 43. Inddata/uddata.(WG6).
CPU, maskinkode og assembler.

UGE 44. Mere om data.(WG8).
Operativsystemer.

UGE 45. Mere om programstruktur.(WG9).
(Datakommunikation.)

UGE 46. Mere om underprogrammer.(WG10).
FORTRAN.

UGE 47. Yderligere datatyper i PASCAL.(WG11).
NAG, MATLAB og REDUCE.

UGE 48. Filer og TEXT-filer.(WG12).
EDB-grafik (og databaser.)

UGE 49. Lidt om algoritmer.(LP).
Tekstbehandling (SCRIPT).
Administrative og tekniske anvendelser.

4. OM BRUGEN AF TERMINALER TIL IBM 4341.

(Materiale til introduktionsdemonstration af terminalerne.)

Det praktiske arbejde - indtastning og afprøvning af PASCAL programmer - i forbindelse med Datalogi A undervisningen, vil i efteråret 1985 foregå ved terminaler til Universitetets IBM 4341 anlæg. Disse terminaler er opstillet i lokale A109 på H.C.Ørsted Institutet.

BRUGERIDENTIFIKATION. Til hver studerende på Datalogi A er der oprettet en 'virtuel maskine' under IBM 4341 maskinen, med en brugeridentifikation af formen MATDATXY, hvor XY er et tocifret tal (00,01,...,99).

PASORD. For at kontrollere hvem der kan få kontakt med en virtuel maskine er der til hver virtuel maskine knyttet et 'løsen' (PASSWORD), som på opfordring skal oplyses ved LOGON. Ved introduktionsdemonstrationen udleveres brugeridentifikationer og pasord. Det er muligt at ændre sit løsen (se PG afsnit 9). Vent med at ændre løsen til du har opnået fortrolighed med terminalerne, og sørg i givet fald for at vælge et løsen du kan huske. Det er muligt (men besværligt) at finde glemte løsener. Sørg endvidere for, at kun du selv kender dit pasord.

Øvelse. Få kontakt med din virtuelle maskine.

BRUG AF EDITOREN. Filer kan oprettes og ændres ved hjælp af editorprogrammet XEDIT (se PG, afsnit 5). Kommandoen:

X NAVN TESTDATA A

vil, hvis der findes en fil med navn: NAVN TESTDATA A , gøre denne tilgængelig for rettelser ('editering'), og ellers oprette en ny (tom) fil med navn: NAVN TESTDATA A , og gøre denne tilgængelig for tilføjelser og ændringer.

Når filen har fået det ønskede udseende gemmes den ved kommandoen FILE skrevet på editorens kommandolinie, eller simpelthen ved at trykke på PF3 tasten.

Øvelse. Prøv at oprette en fil: MXY_0A TESTDATA A (hvor XY er de to tegn der afslutter din brugeridentifikation) og indsæt to linier, hver bestående af et helt tal, i denne fil. Husk at gemme filen.

HVIS TASTATURET LÅSER. Det vil i starten ofte ske, at tastaturet 'låser' (hvilket sker fordi man har forsøgt at indtaste et 'forbudt' sted på skærmen). Dette bliver tilkendegivet ved et kryds i feltet forneden på skærmen. Situationen kan normaliseres ved at trykke på 'RESET'- tasten tilvenstre for mellemrumstasten (eventuelt skal også 'ALT'-tasten samtidig holdes nede).

Læg iøvrigt mærke til 'status'-feltet nederst på skærmen (en grund til at maskinen virker 'død' kunne være, at den simpelthen venter på en indtastning).

B-DISKEN. Hver brugeridentifikation er automatisk tilknyttet en privat såkaldt A-disk til opbevaring af brugerens egne program- og datafiler. Til opbevaring af filer til fælles brug for samtlige

studerende på Datalogi A er der oprettet en såkaldt B-disk, som indeholder filer med opgaver, små PASCAL prøveprogrammer, hjælpeprogrammer, holdfortegnelser, meddelelser og lignende.

Indholdet af B-disken kan ses ved brug af kommandoen:

FILELIST * * B,

og filerne på B-disken kan læses på samme måde som filer på A-disken, og modificeres ved hjælp af editoren. En modificeret fil fra B-disken kan dog kun gemmes på den private A-disk (se nedenfor).

Øvelse. Prøv at kigge på indholdet af B-disken.

En af filerne på B-disken: NYHEDER MEMO B, er beregnet til korte (vigtige) meddelelser til samtlige Datalogi A studerende.

Øvelse. Prøv at give kommandoen: TYPE NYHEDER MEMO B

KØRSEL AF PASCAL PROGRAMMER. Den typiske databehandlingsopgave består i, at nogle givne 'data' ønskes bearbejdet efter visse præcise 'forskrifter' for at give nogle 'resultater'. Her kan 'forskrifterne' f.eks. være specificeret i et PASCAL program som er placeret i en fil, medens de givne 'data', ligeledes placeret i en fil benyttes i programmet, som under sin kørsel udskriver 'resultaterne' og placerer dem i en fil.

På B-disken er indlagt et hjælpeprogram (kommandoprogram) RUN til simpel kørsel af PASCAL programmer. RUN vil forsøge at oversætte PASCAL kildeprogrammet, og hvis dette lykkes, udføre det oversatte program.

For eksempel kan PASCAL kildeprogrammet som findes i filen: SUM PASCAL B, og som læser sine data fra filen: MXY_OD TESTDATA A oversættes og udføres med kommandoen:

RUN SUM MXY_OD

og efter kørslen vil programmets resultater findes i filen: MXY_OD TESTOUT A. (Hvis der allerede findes en fil med navn: MXY_OD TESTOUT A, vil indholdet i denne blive erstattet med resultaterne fra programkørslen.) Derudover vil RUN sørge for at vise inddata og uddata filerne på skærmen.

Øvelse. Kig på PASCAL prøveprogrammet SUM på B-disken og benyt RUN til at 'køre' det. Husk at skrive inddata filen først. Prøv at skrive andre inddata filer, der f.eks. kan navngives: MXY_OB TESTDATA A , MXY_OC TESTDATA A osv., og gentag kørslen af SUM med disse nye inddata filer.

Et PASCAL program der ikke benytter inddata (disse er så indlagt i selve programmet) kan også oversættes og udføres med RUN. F.eks. kan PASCAL kildeprogrammet SQUARES fra B-disken oversættes og udføres med kommandoen:

RUN SQUARES

Til hjælp ved løsning af programmeringsopgaverne i Datalogi A findes et afprøvningsprogram TEST på B-disken. Dette afprøvningsprogram vil først forsøge at oversætte løsningen ~~til en~~ løsningsforsøget

programmeringsopgave. Hvis oversættelsen lykkes vil TEST køre løsningen med alle de prøveinddatafiler der hører til opgaven og derefter køre en korrekt løsning til opgaven med de samme inddatafiler for tilsidst at sammenligne uddatafilerne fra løsningsforsøget med uddatafilerne fra den rigtige løsning. De to sæt af uddatafiler skal have helt det samme indhold (pånær blanktegn og linieskift) for at kunne godkendes. Der er naturligvis en række situationer hvor afprøvningen ikke kan fortsættes og disse vil blive tilkendegivet på skærmen. Se afsnit 5 i denne Introduktion for en nærmere beskrivelse af TEST's virkning.

TEST vil ud fra løsningens filnavn finde frem til de tilhørende inddatafiler og den korrekte opgaveløsning. F.eks. vil TEST efter kommandoen:

TEST MXY_0

forsøge at oversætte PASCAL kildeprogrammet fra filen MXY_0 PASCAL A, og derefter køre det med alle inddatafiler med navn: MXY_0Z TESTDATA A , (hvor Z er et eller andet bogstav), der findes på A-disken, for tilsidst at gøre det samme med det korrekte program: OPG_0 PASCAL og derefter sammenligne de to sæt uddatafiler. NB. Det korrekte program findes kun i oversat form, nemlig i filen: OPG_0 TEXT , på B-disken, så man kan ikke snydekigge.

> Øvelse. Prøv at ændre PASCAL programmet SUM på B-disken, til et program med navn MXY_0 , der indlæser to hele tal og udskriver produktet af det første og det andet tal. Også programnavnet skal ændres og den tekst som udskrives (og filnavnet som programmet gemmes under), og kommentarerne i programmet skal kommentere det ændrede program (vildledende kommentarer er værre end ingen kommentarer).
>
> Den nemmeste fremgangsmåde er nok at starte editorprogrammet med kommandoen: X SUM PASCAL B, lave de ønskede rettelser, og derefter gemme det rettede program på A-disken med kommandoen: FILE MXY_0 PASCAL A (på editorens kommandolinie).
>
> Benyt derefter TEST til at 'køre' det nye program med de forskellige inddatafiler MXY_0Z TESTDATA A , som tidligere blev skrevet.

AFBRYDELSE AF EN KØRSEL. Det kan være nødvendigt at kunne standse den virtuelle maskine under en kørsel af et program f. eks. på grund af fejl i programmet. Det vil senere blive forklaret hvorledes dette nemmest kan gøres. Her nøjes vi med at prøve en tilsyneladende brutal måde til at afbryde en kørsel. Denne måde består i at sætte Normal/Test kontakten på panelet til højre for skærmen i 'Test' positionen og skifte tilbage til 'Normal' igen. Derefter vil 'logo'-et komme på skærmen og man kan logge på igen på normal måde, og for at komme til den sædvanlige sitation må man taste: IPL CMS (samt en tom linie).

> Øvelse. Prøv den voldsomme måde til afbrydelse.

PROFILE EXEC. Den virtuelle maskine er fra sin 'fødsel' forsynet med et vigtigt hjælpeprogram ved navn PROFILE EXEC, som er anbragt på maskinens A-disk. Dette program består af en række kommandoer (se PG, afsnit 10), som udføres hver gang der etableres kontakt med

maskinen (det er blandt andet dette program, der udføres i tidsrummet fra indtastningen af løsenet til det første 'R;' kommer).

Det er vigtigt at programmet PROFILE EXEC ikke ødelægges, da det er med til at definere den 'virtuelle' maskine (f. eks. adgang til systemprogrammer, betydningen af PF-tasterne m.m.).

UDSKRIVNING AF FILER PÅ PRINTER. En fil kan udskrives på en af de printere, der findes i terminalrummet ved hjælp af kommandoerne: PHCO1 eller PHCO2. For eksempel vil kommandoen:

PHCO2 SUM PASCAL B

bevirke at filen: SUM PASCAL B, (som findes på B-disken) udskrives på printer nummer 2 (som normalt er forsynet med A4-papir på højkant), medens PHCO1 anvendt analogt bevirker udskrift på printer nummer 1. Disse printerkommandoer kan også anvendes i kommandoområdet i en filliste (ud for den fil der ønskes udskrevet).

RESERVATION AF TERMINALER. Der vil normalt være adgang til terminalerne i lokale A109 i tidsrummet 8.00-17.00, og der kan reserveres terminaltid i den fremlagte reservationsbog. Hver bruger har ret til at foretage en reservation af en terminal i en eller to halve timer ad gangen, dog må der af en enkelt bruger ikke foretages reservation af mere end sammenlagt en times varighed. Reservationen gøres ved at anføre 'brugerid' i feltet ud for terminalen og det ønskede klokkeslet (på den ønskede dag).

5. EFTERÅRETS PROGRAMMERINGSOPGAVER.

En vigtig del af arbejdet i forbindelse med undervisningen i Datalogi A består i udfærdigelse af PASCAL programmer og afprøvning af disse programmer med forskellige sæt af inddata. I begyndelsen vil opgaverne bestå i at modificere (forbedre) eksisterende programmer og afprøve de nye programmer, senere vil opgaverne være selvstændig udarbejdelse/afprøvning af PASCAL programmer.

A.Udformningen af programmerne. Besvarelsen af opgave n (n = 1..7) skal ske i en fil med navn og type

MXY_n PASCAL

hvor XY er de to sidste tegn i brugeridentifikationen MATDATXY.

Den første linie i programmet skal indeholde en kommentar med opgavenummer, samt besvarerens navn og holdnummer. Ved kommentarer bruges "parenteserne"

'/*' og '*/'

i stedet for standard PASCAL's '(*' og '*)', der giver problemer med 'æ', 'ø' og 'å'.

Derefter skal følge en kommentar med en kort beskrivelse af formålet med programmet, samt en kommentar med en nøjere specifikation af hvorledes programmet vekselvirker med sine omgivelser, dvs. filer (og senere for underprogrammers vedkommende også parametre og globale variable).

Specifikationen må nøje præcisere tilstanden af de filer, som programmet læser fra eller skriver i, både før og efter programmet er udført. Der skal altså stå de betingelser, som programmet antager er opfyldt, når det starter, og det er programmets ansvar, at der altid reageres fornuftigt, ~~(eventuelt med fejlmeddelelser)~~ når disse krav er opfyldt. Udseendet af filerne efter kørslen skal i princippet være beskrevet så nøje, at beskrivelsen alene kan danne grundlag for udfærdigelsen af andre programmer, der arbejder videre med filerne.

Selve programmet skal skrives, så det fremtræder overskueligt og velkommenteret. Det er svært at give præcise regler, men instruktorerne vil kommentere de færdige programmer og give anvisninger på, hvorledes deres opstilling/strukturering forbedres.

B.Afprøvning af programmerne. Afprøvningen af et program sker ved at køre dette med en række inddatafiler, der svarer til specifikationerne i programmet. Det er vigtigt at sørge for, at der er så mange forskellige filer at afprøve programmet med, at alle dele af dette bliver gennemløbet, altså ikke blot de "normale" tilfælde, men også tilfælde, hvor programmet skal kunne reagere med en fejlmeddelelse.

Inddatafilerne til afprøvning af program MXY_n PASCAL skal alle have filtype TESTDATA og filnavne

MXY_nZ

hvor tegnet Z er et bogstav. Når MATDAT99 har lavet besvarelsen

M99_1 PASCAL

af opgave 1, kan man f.eks. tænke sig at han/hun laver 7 datafiler for at dække alle muligheder, som programmet skal kunne klare, og kalder disse

M99_1A TESTDATA
M99_1B TESTDATA
M99_1C TESTDATA
M99_1D TESTDATA
M99_1E TESTDATA
M99_1F TESTDATA
M99_1G TESTDATA

Når program og testdata til opgave n er færdige, gives kommandoen

TEST n

der bevirker, at programmet oversættes og køres med alle TESTDATA filerne (eller kommandoen TEST afgives fra fillisten ud for PASCAL programmet). Testfilerne bliver også kørt med en korrekt besvarelse af opgaven (et facitprogram), og uddata fra de to programmer bliver sammenlignet. Denne sammenligning sker tegn for tegn, for tals vedkommende er det dog talværdierne der sammenlignes, hvor der kun ses bort fra eventuelle blanktegn og linieskift. Hvis programmerne har givet samme resultat skrives

Ingen fejl fundet

og uddata hørende til MXY_nZ TESTDATA findes i filen

MXY_nZ TESTOUT.

Hvis oversætteren finder fejl under oversættelsen af programmet MXY_n PASCAL, bliver man sendt ind i editoren med skærmen delt i to dele, hvor man i øverste del ser på en fil med fejlmeddelelser fra oversættelsen, medens man i nederste del ser på PASCAL programmmet. Man kan nu arbejde med begge filer, idet markørens placering i øvre eller nedre del afgør hvilken fil kommandoer gælder. I filen med fejlmeddelelser er linienummereringen fjernet for til højre at give plads til fejlmeddelelsernes linienumre, der refererer til PASCAL programmet. Når man har rettet fejlene i PASCAL programmet, må man først trykke på PF3, medens markøren er i nederste del af skærmen, hvorved rettelserne indføres endeligt i programmet, hvorefter men blot skriver qq i kommandolinien i en af de to halvdele af skærmen, der nu begge viser filen med fejlmeddelelser.

Hvis blot en af TESTDATA filerne giver et resultat, der ikke kan godkendes, dannes en fil kaldet

RAPPORT MEMO .

Denne indeholder for hver ikke-godkendt TESTDATA fil først en kopi af denne fil og dernæst kopier af uddata fra kørslen (filtype TESTOUT) og uddata fra facitprogrammet (filtype FACITOUT).

Uddatafilerne kan eventuelt være afsluttet med en fejlmeddelelse fra kørslen af PASCAL programmet, forårsaget af f.ex. division med nul. Sådanne fejlmeddelelser indeholder henvisning til et STATEMENT nummer i PASCAL programmet. Disse numre er desværre ikke linienumrene i kildeteksten men numre oversætteren selv tildeler sætningerne i programmet. Hvis der forekommer sådanne meddelelser dannes også en fil

PROGRAM LISTING

som indeholder PASCAL programmet med disse STATEMENT-numre.

C.Aflevering af programmerne. Når programmet kører tilfredstillende sendes det til en speciel rette maskine, knyttet til Datalogi A, med identifikation:

MATTEST

ved brug af den normale SENDFILE kommando. Dvs. at MATDAT99 sender filen M99_1 PASCAL A ved at taste

SF M99_1 PASCAL A MATTEST

eller ved at taste

SF / MATTEST

i kommandofeltet udfor M99_1 PASCAL A i fillisten.

Programmet køres af MATTEST-maskinen med en række testdatafiler, som det skal kunne klare. Hvis resultatet kan godkendes meddeles dette, hvorefter programmet videresendes til besvarerens instruktor. Denne får en udskrift af programmet, som besvareren senere får tilbage ved øvelserne med kvittering og kommentarer. Udskriften skal gemmes. Nøjagtige tidsfrister for aflevering aftales på hvert enkelt hold.

Hvis MATTEST ikke kan godkende besvarelsen, meddeles dette, og der sendes en fil til besvareren med en kort beskrivelse af hvilke typer testdata, der har givet anledning til fejl. Denne fil modtages på sædvanlig måde i 'READER'en (se PG afsnit 11). Det er herefter IKKE muligt at sende en ny besvarelse før tidligst dagen efter, så det er vigtigt at sørge for selv at have afprøvet programmet med en relevant samling TESTDATA filer.

Udover at fungere som rette maskine har MATTEST-maskinen en række bogholderfunktioner. Bl.a. fører MATTEST en liste over afleverede/godkendte opgaver, og ved at give kommandoen:

OPSTATUS

vil MATTEST få tilsendt en anmodning om at oplyse indholdet af denne opgave-status-liste (for afsenderen), og MATTEST vil derefter lave en statusrapport for afsenderen og sende denne til READER'en (i en fil: MATDATXY STATUS , hvor XY er identifikationen).

Matematisk Institut
Københavns Universitet

September 1984

Noter om lineær programmering.

Datalogi A(M).

1. Indledning.

Løsning af praktiske problemer ved hjælp af matematik og EDB foregår i flere faser:

A. Erkendelse af et problem.

B. Formulering af problemet ved hjælp af en matematisk model og vurdering af denne.

C. Analyse af den matematiske model blandt andet med henblik på brugbare algoritmer.

D. Valg af datamatisk løsning udfra hensyn til den forventede brug af det færdige programmel, samt hvad der er til rådighed af

maskiner
algoritmer
programmeringssprog
andet programmel
arbejdskraft.

E. Programmering, programafprøvning, programdokumentation og udfærdigelse af brugervejledning.

F. Kørsler.

I praksis holdes disse faser ikke skarpt adskilt, da man i hver fase naturligvis efter bedste evne bør tage hensyn til arbejdet i de senere faser. Behov for forbedret programmel vil ofte føre til at arbejdet genoptages fra en af de tidligere faser.

I dette kursus vil vi for at blive fortrolige med den ovenfor skitserede proces som "case" benytte udvikling af programmel til lineær programmering. Det er naturligvis navnlig faserne D og E, der interesserer os her.

2. Hvad er "lineær programmering" ?

Disciplinen "lineær programmering" stammer fra sidst i fyrrerne. Ordet "programmering" er ikke anvendt i sin nuværende betydning, men må nærmest oversættes ved "optimering". (Dengang "programmerede" man ikke datamaskiner, men "(maskin)kodede" dem).

Et lineært program er en optimeringsopgave, der går ud på at bestemme værdier af en række tal $x_1,\ldots,x_n$ for hvilke en lineær objektfunktion

$$f(x_1,\cdots,x_n) = c_1x_1 + \cdots + c_1x_n$$

antager sin største (eller mindste) værdi, når $x_1,\cdots,x_n$ opfylder bibetingelser af formen

$$\begin{aligned} & x_1,\cdots,x_n \geqq 0 \\ & a_{11}x_1 + \cdots + a_{1n}x_n \; R_1 \; b_1 \\ & \cdots \\ & a_{m1}x_1 + \cdots + a_{mn}x_n \; R_m \; b_1 \, , \end{aligned}$$

hvor relationerne $R_1,\cdots,R_m$, der gerne må være forskellige, er "$\leqq$", "$\geqq$" eller "=".

Som eksempel kan nævnes opgaven

$$\begin{cases} x_1 - 2x_2 + 3x_3 + x_4 = \min \\ x_1 + x_3 \leqq 2 \\ x_2 - x_4 \geqq 0 \\ x_1 + x_2 + x_3 + x_4 = 5 \, , \end{cases}$$

hvor det er underforstået, at tallene x_1,x_2,x_3 og x_4 ikke må antage negative værdier.

3. Hvornår har man behov for lineær programmering ?

Vi ser på tre typiske eksempler:

A. En landmand ved, at hans bestand af svin behøver 60, 84 og 72 enheder af tre forskellige slags protein om dagen. Han har mulighed for at skaffe det i to slags fodertilsætning X og Y, der indeholder henholdsvis 3,7,3 og 2,2,6 enheder af de tre slags protein pr.kg. Et kg. X koster 10 kr. og et kg. Y koster 4 kr.

Antag, at han bruger x kg. X og y kg. Y om dagen. Der skal da gælde

$$3x + 2y \geqq 60$$

$$7x + 2y \geqq 84$$

$$3x + 6y \geqq 72 .$$

Både x og y skal være større end eller lig nul, og landmanden slipper så billigt som muligt ved at sørge for, at

$$10x + 4y = \min .$$

B. Vi skal nu se på et transportproblem. Her er det ikke helt naturligt at formulere problemet som et lineært program, men omformningen til et "velkendt" problem kan alligevel betale sig.

En papirfabrikant har to papirmøller, der ugentlig producerer 350 t og 550 t. Han skal hver uge levere 300 t til trykkeri 1, 400 t til trykkeri 2 og 200 t til trykkeri 3. Transportomkostningerne pr. ton fra møllerne til trykkerierne er givet ved

		Trykkeri		
		1	2	3
Mølle	1	27	22	15
	2	18	16	12

Kaldes antal tons papir, der ugentlig skal transporteres fra mølle i til trykkeri j for

$$x_{ij} ,$$

skal åbenbart gælde

$$
\begin{aligned}
x_{11} + x_{12} + x_{13} &= 350 \\
x_{21} + x_{22} + x_{23} &= 550 \\
x_{11} + x_{21} &= 300 \\
x_{12} + x_{22} &= 400 \\
x_{13} + x_{23} &= 200 .
\end{aligned}
$$

Det gælder om at minimere de samlede transportomkostninger, dvs.

$$27x_{11} + 22x_{12} + 15x_{13} + 18x_{21} + 16x_{22} + 12x_{23} = \min .$$

Her er det igen underforstået, at x-erne ikke må være negative.

C. Antag at der er givet en funktion

$$f: [a,b] \to \mathbb{R} ,$$

samt yderligere f.eks. to funktioner

$$g_1, g_2: [a,b] \to \mathbb{R} .$$

Vi ønsker at bestemme to konstanter k_1, k_2, så funktionen

$$h(x) = k_1 g_1(x) + k_2 g_2(x)$$

approksimerer $f(x)$ "bedst muligt" i intervallet $[a,b]$, dvs., så størsteværdien i $[a,b]$ af afvigelsen

$$|h(x) - f(x)|$$

bliver mindst mulig.

Denne opgave kan ikke løses eksakt med lineær programmering, men en brugbar tilnærmelse fås ved at vælge et rimeligt "tæt" sæt punkter $c_1, \cdots, c_N$ i $[a,b]$, og bestemme k_1 og k_2 og z, så

$$|h(c_i) - f(c_i)| \leq z \quad \text{for} \quad i = 1, \cdots, n$$

og

$$z = \min .$$

Dette optimeringsproblem har ganske vist ikke umiddelbart den form som vi tidligere har betragtet, men k_1 og k_2 kan erstattes af x_1-y_1 og x_2-y_2, hvorefter vi forlanger, at

$$g_1(c_i)x_1-g_1(c_i)y_1 + g_2(c_i)x_2-g_2(c_i)y_2 - z \leqq f(c_i)$$
$$g_1(c_i)x_1-g_1(c_i)y_1 + g_2(c_i)x_2-g_2(c_i)y_2 + z \geqq f(c_i)$$

for $i = 1,\cdots,N$, samt

$z = \min$,

hvor x_1,y_1,x_2,y_2,z er ikke-negative.

(Den sidste omskrivning udnytter, at

$$|h(c) - f(c)| \leqq z$$

er ensbetydende med

$$h(c) - z \leqq f(c) \quad \wedge \quad h(c) + z \geqq f(c) \quad).$$

Opgaver: Formulér følgende problemer som lineære programmer:

A.) En guldsmed fremstiller to typer smykker, begge af guld og platin. Til den første type bruges 2 g guld og 3 g platin, til den anden 6 g guld og 1 g platin. Guldsmedens råvarelager er 10 kg guld og 8 kg platin. Typerne indbringer det samme i salgspris. Hvor mange smykker af hver type kan guldsmeden udføre med det foreliggende lager af guld og platin, når han vil have størst mulig fortjeneste? (Kald antallet af smykker af de to typer for henholdsvis x og y).

B.) Et land består af fem øer. I tabel 1 vises den årlige æg-produktion og efterspørgsel efter æg, målt i enheder af 10 ton, for de enkelte øer.

x_{AB} = antal tons fra A til B

TABEL 1.

Ø	Produktion	Efterspørgsel
A	40	48
B	28	36
C	80	71
D	53	37
E	110	116
	311	308

Omkostningerne, målt i kr. pr. 10 kg, ved at transportere æg mellem to forskellige øer sættes til de beløb, der er anført i tabel 2.

TABEL 2.

	A	B	C	D	E
A		5	6	8	1
B			4	3	6
C				7	5
D					7

Landbrugsministeriet skal tilrettelægge en fordelingsplan for æg, således at øer med overskudsproduktion skal levere til øer med underskudsproduktion, desuden skal transportomkostningerne minimaliseres.

C.) Antag, at en person pr. uge behøver mindst 8 enheder protein, 12 enheder kulhydrat og 9 enheder fedtstof. Af de nævnte stoffer indeholder et fødemiddel A henholdsvis 2, 6 og 1 enheder pr. kg., et fødemiddel B henholdsvis 1, 1 og 3 enheder pr. kg. og et fødemiddel C henholdsvis 3, 1 og 2 enheder pr. kg.

De tre fødemidler A, B og C koster pr. kg. henholdsvis 8 kr., 4 kr. og 5 kr. Find hvor mange kg. af hver af de tre fødemidler, man skal købe pr. uge, når de tilsammen skal dække ens behov for protein, kulhydrat og fedtstof for den lavest mulige pris.

Hvad bliver resultatet, hvis prisen på A nedsættes med 25%, medens priserne på B og C opretholdes ?

$8 \cdot A +$ min

$8 \times$ antal A + 4 antal B + 5 antal C

4. Omskrivning til normalform.

En ulighed

$$a_{i1}x_1 + \ldots + a_{in}x_n \leqq b_i$$

i et lineært program kan erstattes af ligningen

$$a_{i1}x_1 + \ldots + a_{in}x_n + y_i = b_i \ ,$$

hvor y_i er en såkaldt restvariabel, der ikke optræder andre steder. (Husk at variablene forudsættes at være $\geqq 0$). Ulighedstegnet "$\geqq$" kan behandles analogt.

Hvis problemet går ud på at maksimere en objektfunktion

$$c_1x_1 + \ldots + c_nx_n$$

kan problemet omformuleres til minimering af

$$(-c_1)x_1 + \ldots + (-c_n)x_n \ .$$

Lineære programmer kan derfor altid omskrives til normalformen

objektfunktion: $c_1x_1 + \ldots + c_nx_n = \min$

bibetingelser :
$$\begin{cases} a_{11}x_1 + \ldots + a_{1n}x_n = b_1 \\ \vdots \\ a_{m1}x_1 + \ldots + a_{mn}x_n = b_m \ , \end{cases}$$

hvor $a_i \ , \ldots , \ a_n \geqq 0$. Hvis koefficientmatricen

$$\underline{\underline{A}} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \ ,$$

har rang mindre end m, vil mindst én af rækkerne være en linearkombination af de øvrige. Der kan f.eks. findes $k_1,\ldots,k_{m-1} \in \mathbb{R}$, så

$$(a_{m1},\ldots,a_{mn}) = k_1(a_{11},\ldots,a_{1n}) +\ldots+ k_{m-1}(a_{m-1,1},\ldots,a_{m-1,n}) .$$

Hvis der i dette tilfælde gælder

$$b_m = k_1b_1 +\ldots+ k_{m-1}b_{m-1} ,$$

er den sidste bibetingelse overflødig, da den følger af de øvrige. Hvis derimod

$$b_m \neq k_1b_1 +\ldots+ k_{m-1}b_{m-1} ,$$

kan bibetingelserne overhovedet ikke opfyldes.

I det følgende antages, at LP-problemer er på normalform med $rg\underline{\underline{A}} = m$, da problemer med løsning altid kan bringes på denne form. Da $rg\underline{\underline{A}} \leq n$ gælder $m \leq n$.

5. Basisløsninger.

Hvis variablene $x_{m+1},\ldots,x_n$ sættes lig nul giver bibetingelserne ligningssystemet

$$\begin{aligned} a_{11}x_1 +\ldots+ a_{1m}x_m &= b_1 \\ &\vdots \\ a_{m1}x_1 +\ldots+ a_{mm}x_m &= b_m . \end{aligned}$$

Hvis koefficientmatricen her er regulær, og der altså findes netop én løsning $(\bar{x}_1,\ldots,\bar{x}_m)$, kaldes $(\bar{x}_1,\ldots,\bar{x}_m,0,\ldots,0)$ en <u>basisløsning</u> hørende til <u>basisvariable</u> $x_1,\ldots,x_m$. Ethvert af de ialt $\binom{n}{m}$ delsæt af m variable giver på lignende måde basisløsninger, såfremt den tilhørende koefficientmatrix er regulær. Vor antagelse om at bibetingelserne er på normalform medfører, at der altid findes mindst én basisløsning.

Koordinatsæt, som opfylder bibetingelserne uden kravet om at koordinaterne skal være ikke-negative, kaldes i almindelighed løsninger, og løsninger, hvis koordinater alle er positive eller nul, kaldes tilladte løsninger. Der gælder følgende hovedsætning:

> *Hvis der findes tilladte løsninger, så findes også tilladte basisløsninger, og objektfunktionens minimum* μ *i disse er enten globalt minimum, eller også antages enhver værdi mindre end* μ .

Denne sætning giver umiddelbart en algoritme til løsning af LP-problemer på normalform:

Man gennemløber samtlige delmængder bestående af m af de n variable. For hver delmængde undersøges om den kvadratiske matrix bestående af de tilhørende søjler i koefficientmatricen, er regulær. Hvis dette er tilfældet findes den tilhørende basisløsning, og hvis denne er tilladt, objektfunktionens værdi. Herved bestemmes objektfunktionens minimum μ i de tilladte basisløsninger, såfremt sådanne findes.

Hvis der findes et globalt minimum, må dette være μ . Man kan undersøge om μ er globalt minimum ved til det oprindelige LP-problem at tilføje bibetingelsen

$$c_1x_1 + \ldots + c_nx_n = \mu - 1$$

og undersøge om det udvidede problem har tilladte basisløsninger.

6. Bevis for hovedsætningen.

Vi har givet et LP-problem på normalform

$$\begin{cases} a_{11}x_1 + \ldots + a_{1n}x_n = b_1 \\ \cdot \qquad\qquad \cdot \qquad\quad \cdot \\ \cdot \qquad\qquad \cdot \qquad\quad \cdot \\ \cdot \qquad\qquad \cdot \qquad\quad \cdot \\ a_{m1}x_1 + \ldots + a_{mn}x_n = b_m \\ c_1x_1 + \ldots + c_nx_n = \min . \end{cases}$$

Ved et j-delproblem vil vi forstå et LP-problem, der fremkommer ved at sætte alle undtagen j variable lig 0 , under forudsætning af, at koefficientmatricen bevarer rangen m efter at søj-

lerne hørende til de nulstillede variable er udeladt. Basisløsningerne er altså netop løsningerne til m-delproblemerne.

Da problemet er på normalform, må matricen

$$\underline{\underline{M}} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \\ c_1 & \cdots & c_n \end{pmatrix}$$

have rang m eller m+1 , fordi matricen, der fremkommer ved at fjerne sidste række, har rang m . Hvis rangen er m , er den sidste række en linearkombination af de øvrige, og objektfunktionens værdi i løsninger er konstant, nemlig, den tilsvarende linearkombination af højresiderne $b_1,\ldots,b_m$. I dette tilfælde er hovedsætningen trivielt opfyldt, og vi kan derfor antage, at

$$\text{rg } \underline{\underline{M}} = m+1 \ .$$

Man overbeviser sig let om, at følgende hjælpesætning medfører, at det er tilstrækkeligt at vise hovedsætningen i tilfældet $n = m+1$:

Hjælpesætning. Enhver værdi af objektfunktionen i en tilladt løsning antages også i en tilladt løsning for et (m+1)-delproblem.

Bevis for hjælpesætningen: Vi antager, at t er funktionsværdi for objektfunktionen i en tilladt løsning $(x_1,\ldots,x_n)$, og antager yderligere, at der ikke findes nogen anden tilladt løsning med funktionsværdi t og et mindre antal positive koordinater. Vi antager endvidere, at variablene er nummererede, så $x_1,\ldots,x_j > 0$ og $x_{j+1},\ldots,x_n = 0$.

Der gælder altså

$$x_1 \begin{pmatrix} a_{11} \\ \vdots \\ a_{m1} \\ c_1 \end{pmatrix} + \ldots + x_j \begin{pmatrix} a_{1j} \\ \vdots \\ a_{mj} \\ c_j \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \\ t \end{pmatrix} .$$

Hvis søjlerne på venstresiden er lineært afhængige findes $(k_1,\ldots,k_j) \neq (0,\ldots,0)$, så

$$k_1 \begin{pmatrix} a_{11} \\ \vdots \\ a_{m1} \\ c_1 \end{pmatrix} + \ldots + k_j \begin{pmatrix} a_{1j} \\ \vdots \\ a_{mj} \\ c_j \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}.$$

For alle $s \in \mathbb{R}$ gælder da

$$(x_1 + k_1 s) \begin{pmatrix} a_{11} \\ \vdots \\ a_{m1} \\ c_1 \end{pmatrix} + \ldots + (x_j + k_j s) \begin{pmatrix} a_{1j} \\ \vdots \\ a_{mj} \\ c_j \end{pmatrix} = \begin{pmatrix} b_j \\ \vdots \\ b_m \\ t \end{pmatrix}.$$

Vælges s passende, bliver mindst ét af tallene $x_1 + k_1 s, \ldots, x_j + k_j s$ lig nul, medens de øvrige alle er større end eller lig nul. Vi får således en tilladt løsning med funktionsværdi t og færre end j positive koordinater i modstrid med antagelserne, hvilket viser, at søjlerne på venstresiden er lineært uafhængige. Da $\underline{\underline{M}}$ har rang $m+1$ kan disse søjler suppleres op til et sæt af $m+1$ lineært uafhængige søjler, der bestemmer et $(m+1)$-delproblem med den tilladte løsning $(x_1, \ldots, x_j, 0, \ldots, 0)$, i hvilken objektfunktionen antager værdien t. Hermed er hjælpesætningen vist.

Bevis for hovedsætningen i tilfældet $n = m+1$:

Vi benytter samme betegnelser som tidligere, og som nævnt i indledningen til beviset for hovedsætningen kan vi antage, at $\operatorname{rg} \underline{\underline{M}} = m+1$, dvs. at $\underline{\underline{M}}$ er regulær. Endvidere kan vi antage, at $\det \underline{\underline{M}} > 0$, da dette kan opnås ved eventuelt at gange en bibetingelse med -1.

Ved hjælp af Cramers sætning ses nu, at objektfunktionen antager værdien t for

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \frac{1}{\det \underline{\underline{M}}} \begin{pmatrix} t c_1 q_1 + c_2 q_2 + \ldots + c_n q_n \\ \vdots \\ c_1 q_1 + c_2 q_2 + \ldots + t q_n \end{pmatrix} = \frac{1}{\det \underline{\underline{M}}} \begin{pmatrix} p_1 + q_1 t \\ \vdots \\ p_n + q_n t \end{pmatrix},$$

hvori q_i er komplementet til c_i i $\underline{\underline{M}}$, og p_i-erne er uafhængige af t.

Betingelsen for at værdien t antages i en tilladt løsning er at $p_i + q_i t \geqq 0$ for $i = 1,\ldots,n$, dvs. at

$$\begin{cases} t \geqq -\dfrac{p_i}{q_i} & \text{for alle } q_i > 0 \\ p_i \geqq 0 & \text{for alle } q_i = 0 \\ t \leqq -\dfrac{p_i}{q_i} & \text{for alle } q_i < 0 \end{cases} .$$

Vi kan antage, at $p_i \geqq 0$ for alle $q_i = 0$, da der ellers ingen tilladte løsninger findes.

Lad $A = \{i \mid q_i > 0\}$ og $B = \{i \mid q_i < 0\}$. Mængderne A og B kan ikke begge være tomme, da

$$c_1 q_1 + \ldots + c_n q_n = \det \underline{\underline{M}} \neq 0 .$$

For $i \in A \cup B$ gælder, at objektfunktionen antager værdien $-\dfrac{p_i}{q_i}$ i et punkt med $x_i = p_i + \left(-\dfrac{p_i}{q_i}\right) q_i = 0$. Punktet er derfor løsning til det LP-problem, der fremkommer ved at fjerne x_i og de tilhørende koefficienter. Da koefficientmatricen for dette problem er regulær, fordi determinanten er q_i, er der tale om et m-del-problem, og løsningen er derfor en basisløsning.

Vi definerer

$$\alpha = \max_{i \in A} \left(-\frac{p_i}{q_i}\right), \qquad \text{hvis } A \neq \emptyset ,$$

$$\beta = \min_{i \in B} \left(-\frac{p_i}{q_i}\right), \qquad \text{hvis } B \neq \emptyset .$$

Det er nu let at kontrollere, at hovedsætningen er opfyldt i hvert af følgende tre mulige tilfælde:

<u>$A \neq \emptyset$ og $B \neq \emptyset$</u>. Hvis $\alpha > \beta$ findes ingen tilladte løsninger. Hvis $\alpha \leqq \beta$ er objektfunktionens værdimængde $[\alpha,\beta]$, og både mindste- og størsteværdi antages i en tilladt basisløsning.

<u>$A = \emptyset$ og $B \neq \emptyset$</u>. Objektfunktionens værdimængde er $]-\infty,\beta]$, og størsteværdien antages i en tilladt basisløsning.

<u>$A \neq \emptyset$ og $B = \emptyset$</u>. Objektfunktionens værdimængde er $[\alpha,\infty[$, og mindsteværdien antages i en tilladt basisløsning.

Matematisk Institut
Københavns Universitet

December 1985

Afslutning af LP-noterne

Datalogi A(M).

7. Vurdering af køretider.

Antag at vi vil løse et LP-problem med n variable og m bibetingelser ved hjælp af metoden i afsnit 5. Vi må da normalt indføre m restvariable, hvorved der fremkommer et system på normalform med $m+n$ variable og m ligninger. Det fører til løsning af m ligninger med m ubekendte ialt $\binom{n+m}{m}$ gange.

Tiden der går med at løse m ligninger med m ubekendte er proportional med m^3 (groft sagt indeholder Gauss eleminationsalgoritmer 3 løkker inden i hinanden). Den samlede køretid for LP-problemet bliver derfor at størrelsesordenen,

$$k \cdot m^3 \cdot \binom{n+m}{m} ,$$

hvor k er en proportionalitetsfaktor, der afhænger af den benyttede datamat.
Her er set bort fra den tid, der går med indlæsning og anden "administration", idet denne "kun" er proportional med $n \times m$ og derfor ikke spiller nogen rolle for store problemer.

For $n = m$ får man nemt den meget forsigtige vurdering

$$\binom{n+m}{m} > 2^m .$$

Man må altså i almindelighed forvente, at <u>køretiden vokser mindst eksponentielt med problemstørrelsen</u>.

Den benyttede algoritme er derfor <u>uanvendelig for store problemer</u>.

Ved vurdering af køretiden anvendes ofte O-notationen. Således siges tiden for løsning af m ligninger med m ubekendte ved Gauss-elimination at være af størrelsesorden $O(m^3)$. Med gode algoritmer er tiden for sortering af n poster af størrelsesorden $O(n \log n)$.

8. Simplexalgoritmen.

Vi tænker os givet et LP-problem på normalform som i afsnit 4, og forudsætter yderligere, at højresiderne $b_1,\ldots,b_m$ er ≥ 0.

Simplexalgoritmen tager sit udgangspunkt i en tilladt basisløsning. Hvis en sådan ikke kendes, kan den skaffes ved minimering af

$$t_1 + \ldots + t_m$$

med bibetingelserne

$$\begin{aligned} a_{11}x_1 + \ldots + a_{1n}x_n + t_1 &= b_1 \\ &\vdots \\ a_{m1}x_1 + \ldots + a_{mn}x_n + t_m &= b_m . \end{aligned}$$

Her kan man nemlig starte simplexalgoritmen med den oplagte tilladte basisløsning med $t_1 = b_1,\ldots,t_m = b_m$. Såfremt det oprindelige problem overhovedet har tilladte basisløsninger, har det nye problem minimumsværdien 0 hørende til tilladte basisløsninger for det oprindelige problem.

Idéen er at ændre én basisvariabel ad gangen, så der fremkommer basisløsninger hørende til stadig mindre værdier af objektfunktionen. Princippet i valg af ny basis er følgende:

Antag at basisvariablene er $x_1,\ldots,x_m$. Bibetingelserne kan på matrixform skrives som

$$\underline{\underline{A}}\ \underline{\underline{X}} = \underline{\underline{B}}$$

eller mere udførligt som

$$(\underline{\underline{A}}_1 \vdots \underline{\underline{A}}_2)\underline{\underline{X}} = \underline{\underline{B}}$$

hvor $\underline{\underline{A}}_1$ er den regulære matrix bestående af de første m søjler i $\underline{\underline{A}}$, og $\underline{\underline{A}}_2$ er delmatricen bestående af de øvrige søjler.

Multipliceres fra venstre med $\underline{\underline{A}}_1^{-1}$ fås

$$(\underline{\underline{E}} \vdots \underline{\underline{A}}_1^{-1}\underline{\underline{A}}_2)\underline{\underline{X}} = \underline{\underline{A}}_1^{-1}\underline{\underline{B}} ,$$

hvor $\underline{\underline{E}}$ er m×m-enhedsmatricen. Da de oprindelige bibetingelser kan fås heraf ved multiplikation fra venstre med $\underline{\underline{A}}_1$, kan bibetingelserne altså bringes på formen

$$
\begin{aligned}
x_1 + \quad & + a'_{1,m+1}x_{m+1} + \ldots + a'_{1n}x_n = b'_1 \\
& \vdots \qquad\qquad\qquad\qquad \vdots \\
x_m + \; & a'_{m,m+1}x_{m+1} + \ldots + a'_{mn}x_n = b'_m
\end{aligned}
$$

Trækkes c_1 gange første ligning, c_2 gange anden ligning, osv. op til c_n gange n'te ligning fra ligningen

$$c_1x_1 + \ldots + c_mx_m + c_{m+1}x_{m+1} + \ldots + c_nx_n = f(x_1,\ldots,x_n)$$

for objektfunktionen, får denne ligning formen

$$c'_{m+1}x_{m+1} + \ldots + c'_nx_n = f(x_1,\ldots,x_n) - c_1b'_1 - \ldots - c_mb'_m \ .$$

Hvis $c'_{m+1},\ldots,c'_n$ alle er ≥ 0 fremgår umiddelbart heraf, at objektfunktionen antager sin mindsteværdi i basisløsningen hørende til basisvariable $x_1,\ldots,x_m$.

Ellers vælges den indgående basisvariabel x_i , så det tilhørende c'_i bliver et mindst muligt negativt tal. For hvert $x_i > 0$ bestemmer ligningerne værdier af de gamle basisvariable, og den udgående basisvariabel x_j vælges som den første, der antager værdien 0 , når x_j får lov at vokse fra 0 .

Hvis basisløsningen $x_1,\ldots,x_m$ er udartet, dvs. at tallene $b'_1,\ldots,b'_m$ ikke alle er større end 0 , bliver objektfunktionen ikke nødvendigvis formindsket ved basisskiftet, og metoden er her mere kompliceret.

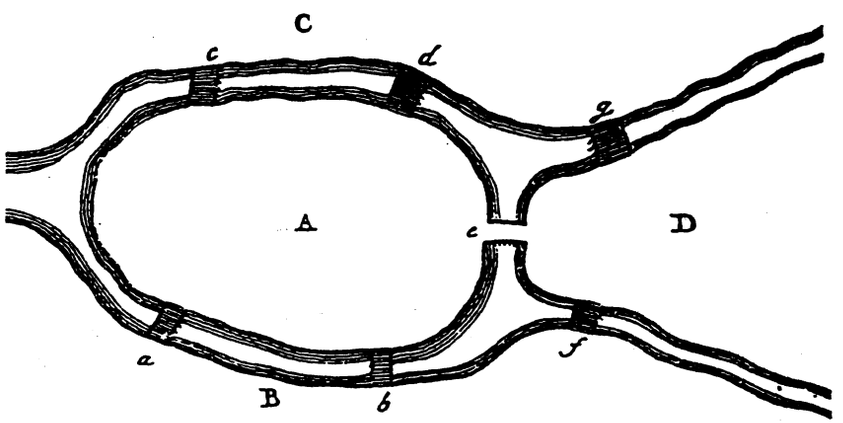
Det har i praksis vist sig, at man normalt kun kommer til at gennemløbe relativt få basisløsninger, så metoden kan anvendes for store problemer.

Populær artikel om kombinatorisk optimering
skrevet til ugebladet "Ingeniøren".

HVAD ER LET OG HVAD ER SVÆRT?

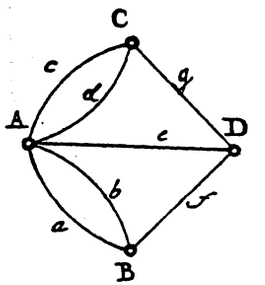
OM TEGNING AF FIGURER I ÉN STREG, HANDELSREJSENDE, POSTBUDE, REGNEMASKINERS MANGLENDE EFFEKTIVITET, EN RUSSISK SENSATION, OG DATALOGIENS VIGTIGSTE UBESVAREDE SPØRGSMÅL.

TROEN PÅ, AT MAN VED HJÆLP AF STORE REGNEMASKINER KAN LØSE ALLE SMÅ PROBLEMER, HOLDER NÆPPE STIK. GENNEM DE SIDSTE TI ÅRS FORSKNING ER DER OPNÅET NY INDSIGT I HVAD DET ER DER GØR AT NOGLE PROBLEMTYPER KAN LØSES LET OG HURTIGT, MENS ANDRE ER BESVÆRLIGE. MEN FORSKNINGEN HAR OGSÅ REJST NYE SPØRGSMÅL AF FUNDAMENTAL KARAKTER PÅ GRÆNSEN MELLEM MATEMATIK OG DATALOGI.



TEKST: BJARNE TOFT

Den berømte matematiker Euler løste i 1736 et "morskabsproblem" om at gå en rundtur i byen Königsberg, således at hver af byens 7 broer blev passéret netop én gang. Euler løste ikke blot det konkrete problem, men alle mulige problemer af samme type. I figuren ovenfor er gengivet et kort fra Eulers artikel over broernes beliggenhed. Euler repræsenterede situationen ved hjælp af en graf:



Grafen har 4 punkter svarende til de to bredder af floden og de to øer, samt 7 kanter svarende til de 7 broer. Problemet er nu: Kan vi tegne grafen i én streg uden at løfte blyanten fra papiret, således at hver kant tegnes præcis én gang og således at vi slutter i samme punkt som vi begyndte? En sådan streg eller rundtur kaldes en Euler-tur.

Eulers løsning var, at en graf har en Euler-tur når og kun når alle punkter har lige valens, dvs. fra alle punkter udgår et lige antal kanter. Grafen over

broerne har altså ingen Euler-tur, da der er et punkt med ulige valens (der er faktisk fire). Vi må betragte Eulers løsning som en god matematisk sætning, idet den giver en let-forklarlig grund på, at der ikke er en Euler-tur. Men set fra en praktisk synsvinkel er løsningen utilfredsstillende, da den ikke giver nogen opskrift på at finde Euler-turen (når der er én). En sådan opskrift, eller algoritme, skal bestå af regler så præcise, at algoritmen kan udføres helt automatisk, evt. af en maskine. En god algoritme for Euler-problemet, dvs. en algoritme, som ikke blot undersøger alle muligheder, blev først givet i 1873.

Ny forskning har sandsynliggjort, at problemtyper med gode matematiske løsninger er præcis de samme som dem, hvor man kan konstruere gode algoritmer. Det typiske har ellers tit været, at de to løsningstyper kom hver for sig, måske med mange års mellemrum, ligesom det skete med Eulers problem.

Den handelsrejsende

En problemtype, som overfladisk minder om Eulers, er problemet om den handelsrejsende: Givet n byer og afstandene mellem dem, samt en længde L. Er der en rundtur, som passérer hver by præcis én gang, og som højst har samlet længde L? Dette problem har været intensivt studeret, men der kendes ikke nogen god matematisk løsning:

Er svaret "nej, der er ingen rundtur af længde højst L", har man normalt ingen let-forklarlig grund hertil, bortset fra den meget langsommelige, at man kan undersøge alle mulige ture og se, at de alle har længde over L. Man kender heller ingen god algoritme til at finde en korteste tur for den handelsrejsende.

Den nye forskning har medført, at man ikke længere tror på, at der er en god matematisk løsning eller en god algoritme for den handelsrejsende, så al eftersøgning må på forhånd betragtes som forgæves.

Gode algoritmer og gode sætninger.

Den nyere udvikling startede i 1965 med at den canadiske matematiker Edmonds foreslog en præcisering af hvornår en algoritme for en problemtype kan kaldes god: antal skridt algoritmen skal gennemløbe for at løse et problem af typen må højst være et polynomium i størrelsen af problemet, altså højst f.eks. n^2 eller n^5, hvor n er størrelsen. Det er en grov præcisering, da den er ligeglad med om antal skridt er n^2 eller n^{17}. Men det der udelukkes er, at antal skridt kan vokse eksponentielt som f.eks. 2^n, eller endnu værre som $n!$, der er tallet $n\cdot(n-1)\cdot(n-2)\cdots3\cdot2\cdot1$. Tal som 2^n og $n!$ angiver ofte antallet af samtlige muligheder. Antal mulige rundture for den handelsrejsende gennem n byer er således $(n-1)!$. At sige, at vi har en god algoritme for en problemtype betyder derfor, at vi har en langt hurtigere metode end den langsommelige som gennemser alle muligheder.

Moderne regnemaskiner kan udføre et meget stort antal skridt pr. sekund, helt op til 100 millioner, så i lyset heraf kan forskellen på n^2 og 2^n forekomme uden større betydning. Men tabellen viser, at det er den langtfra: Selv for små værdier af n må en super-regnemaskine give op hvis 2^n eller (n-1)! skridt skal udføres.

Det var også Edmonds, som først formodede, at gode algoritmer og gode sætninger følges ad. Han præciserede begrebet "god sætning" således: Hvad enten svaret er "ja" eller "nej" skal der eksistere en let-forklarlig grund. For den handelsrejsende eksisterer der en let-forklarlig grund når svaret er "ja, der er en tur af længde højst L" (man kan blot fremvise turen). Men der kendes som sagt ingen let-forklarlig grund når svaret er "nej".

Edmonds underbyggede sin formodning ved at konstruere gode algoritmer for problemer, hvor man allerede fra tidligere kendte gode sætninger. De nye algoritmer, der herved opstod, har vist sig at være af betydning i forbindelse med løsning af flere problemer med praktiske anvendelser, f.eks. postbud-problemet (find i en graf en korteste rundtur, som passérer alle kanter i grafen mindst én gang hver) og visse typer af skemalægning og projekt-planlægning.

Datalogiens vigtigste ubesvarede spørgsmål.

Det endelige gennembrud for Edmonds' ideer kom i 1972, da canadieren Cook knyttede forbindelsen til teoretisk datalogi. Ved hjælp af en tænkt model for regnemaskiner gav Cook en helt præcis betydning til klasserne P, NP og co-NP. Klassen P er de problemtyper, hvor der er gode algoritmer. Klassen NP er de problemtyper, hvor svaret "ja" har en let-forklarlig begrundelse, og co-NP er de tilsvarende med "nej". Klassen P er en del af NP og af co-NP , og Edmonds' formodning lyder nu helt præcis: Er de fælles problemtyper for NP og co-NP netop P ? Cook stillede det mere vidtgående spørgsmål om det faktisk er sådan at P = NP = co-NP ? Svaret kendes ikke, men P = NP ville være for godt til at være sandt! Det ville indebære, at der eksisterer gode algoritmer og gode sætninger for næsten alle problem-typer, f.eks. ville der være en god algoritme for den handelsrejsende blot i kraft af den meget simple bemærkning at svaret "ja" har en let-forklarlig begrundelse. Alle er derfor overbevist om at P er forskellig fra NP . Spørgsmålet om det virkelig forholder sig sådan er idag den teoretiske datalogi's vigtigste ubesvarede spørgsmål.

Cooks hovedresultat fra 1972 var, at der blandt NP-problemtyperne er én, som er vanskeligere end alle de andre NP-problemtyper: kan man finde en god algoritme for blot denne ene problemtype vil man automatisk have fundet gode algoritmer for alle andre NP-problemtyper (med P = NP til følge). En sådan vanskeligste problemtype

i NP kaldes NP-komplet. Den amerikanske matematiker Karp viste snart efter, at der ikke blot er én, men mange NP-komplette problemtyper, inklusive en lang række kendte problemer, bl.a. den handelsrejsende. En god algoritme for den handelsrejsende medfører altså også at P = NP , og bliver dermed utænkelig.

Listen over NP-komplette problemtyper er nu meget lang. Siden 1972 er der udkommet mere end 1000 videnskabelige artikler om emnet, og man har nu et meget detaljeret billede af hvad der har gode løsninger og hvad ikke.

En russisk sensation.

Det, som har fremkaldt den største opmærksomhed, har været udviklingen indenfor lineær programmering (forkortet LP). LP er den matematiske disciplin, som benyttes mest ved løsning af praktiske problemer på regnemaskine (bortset fra bogholderi, søgning og sortering). LP er interessant fordi der eksisterer en god sætning for LP, mens den sædvanlige algoritme, simplex-metoden, ikke er nogen god algoritme. LP var således det bedste bud på et modeksempel til Edmonds' formodning. Men i 1979 offentliggjorde en russisk matematiker Khachian en god algoritme for LP. Algoritmen kaldes nu ellipsoide-metoden. I visse tilfælde fungerer den meget bedre end simplex-metoden, men i gennemsnit er den dårligere, så det er endnu et spørgsmål om den får praktisk betydning.

Ellipsoide-metoden vakte stor opsigt, da den blev opdaget i Vesten i 1980, med amerikanske avisoverskrifter af typen: "Et russisk geni ryster computer-verdenen!". Grunden til opsigten blandt forskere var først og fremmest forbindelsen til Edmonds' formodning. Men Khachian havde ikke vist, at den handelsrejsende er et P-problem, som nogle aviser skrev. Det ville have været en langt større sensation, thi det ville have betydet at P = NP .

P contra NP.

Ingen synes at have nogen idé om, hvordan man skal løse problemet om P er forskellig fra NP. Enkelte forskere har peget på muligheden af, at spørgsmålet måske slet ikke kan besvares med de grundlæggende forudsætninger vi benytter os af i normal matematik. De fleste er dog ikke så pessimistiske, men ingen venter at svaret kommer snart.

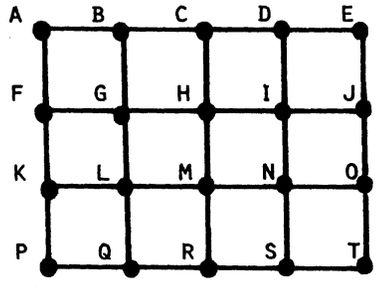
Et bevis for at P = NP er således også stadig en mulighed. Det er en ønskedrøm, som ville have stor økonomisk betydning, idet det ville betyde, at regne-maskiners effektivitet i flere sammenhænge med et slag ville forøges voldsomt.

Antal skridt	N = 10	N = 20	N = 30	N = 50	N = 60
N^2	0,000001 sek	0,000004 sek	0,000009 sek	0,000025 sek	0,000036 sek
N^5	0,001 sek	0,032 sek	0,24 sek	3,1 sek	7,8 sek
2^N	0,00001 sek	0,01 sek	10,7 sek	130 døgn	366 år
(N-1)!	0,0036 sek	38,6 år	$2{,}8 \cdot 10^{15}$ år	$1{,}9 \cdot 10^{47}$ år	$4{,}4 \cdot 10^{64}$ år

SELV EN SUPERREGNEMASKINE, SOM KAN KLARE 100.000.000 SKRIDT PR SEKUND, DUR IKKE, HVIS DER SKAL UDFØRES 2^N SKRIDT. REGNETIDEN VIL VÆRE 130 DØGN FOR N = 50 OG 366 ÅR FOR N = 60. HVIS DER SKAL UDFØRES (N-1)! SKRIDT, SOM ER ANTALLET AF MULIGE RUTER FOR EN HANDELSREJSENDE, DER SKAL BESØGE N BYER, VIL REGNETIDEN FOR N = 20 VÆRE OVER 38 ÅR OG FOR N = 30 OVER 2000 BILLIONER ÅR.

OPGAVE

TEGNINGEN FORESTILLER GADERNE I EN BY. HVER AF DE 31 GADER ER 100 M LANG. ET POSTBUD SKAL PÅ EN RUNDTUR GENNEMKØRE ALLE GADER MINDST ÉN GANG. FIND EN KORTEST MULIG RUNDTUR FOR POSTBUDET !



KOMMENTAR

DET NÆRLIGGENDE SVAR, AT EN KORTEST MULIG RUNDTUR ER 3100 M, ER IKKE KORREKT, DA DER IKKE EKSISTERER EN RUNDTUR, SOM GENNEMKØRER ALLE GADER PRÆCIS ÉN GANG. DET ER NØDVENDIGT AT GENNEMKØRE NOGLE AF GADERNE TO GANGE.

LØSNING

EDMONDS' LØSNINGSMETODE ER AT PUNKTERNE MED ULIGE VALENS (B,C,D,F,J,K,O,Q,R,S) ORDNES I PAR, SÅLEDES AT SUMMEN AF ALLE AFSTANDE MELLEM TO PUNKTER DER DANNER PAR ER MINDST MULIG (ÉN MULIGHED ER DE 5 PAR (B,C), (D,J), (F,K), (Q,R), (O,S) MED DE 5 AFSTANDE 100 M, 200 M, 100 M, 100 M, 200 M, IALT 700 M). EN KORTEST MULIG RUNDTUR BENYTTER NU DE 700 M TO GANGE OG RESTEN ÉN GANG, D.V.S. EN KORTEST MULIG RUNDTUR BLIVER PÅ 3800 M (F.EKS. ABCDEJIHGFKLMNOTSTOJEDINSRQRMHCBGLQPKFA). ETHVERT POSTBUD-PROBLEM LØSES LET MED DENNE TEKNIK.

Københavns Universitet
Naturvidenskabelig embedseksamen, maj 1985

DATALOGI A for matematikere

Sættet består af ialt 6 sider, med numrene 1,...,6 (siderne 4 og 5 indeholder begge et eksemplar af Opgave 5, hvoraf det ene, med påførte rettelser kan vedlægges besvarelsen).

Opgave 0 skal besvares uden brug af hjælpemidler i løbet af den første time hvorefter besvarelsen afleveres. Ved besvarelsen af de øvrige opgaver kan alle sædvanlige hjælpemidler benyttes.

Opgave 0. *Opgaver af denne karakter vil ikke forekomme mere.*

1a) Giv f.eks. med udgangspunkt i en blokskitse en beskrivelse af en datamaskines principielle opbygning (CPU, lager og inddata/uddatamedier).

1b) Giv en kort beskrivelse af en CPU's hoveddele og deres funktion.

1c) Nævn forskellige typer af inddata- og uddatamedier og giv eksempler på deres anvendelse.

2) Nævn hvilke kontrolstrukturer der indgår i 'struktureret' programmering, og giv ud fra disse en kort karakterisering af nogle programmeringssprog.

3) Angiv de vigtigste funktioner for et operativsystem (multiprogrammering, tidsdeling, filhåndtering m.v.) - gerne belyst med eksempler fra egne erfaringer.

Opgave 1.

Afgør for hvert af nedenstående udtryk om det er et gyldigt PASCAL-udtryk, og angiv i bekræftende fald dets type og værdi.

a) 9 * 6 / 14 * 2
b) chr(succ('4')-1)
c) round(sqrt(2))
d) (1 < 5 <= 8)
e) succ(sqr(10E1))
f) ('a'='b')=('c'<'d')
g) ('3')0) or (4<2)
h) ('a'<='b')or('9'<'8')and('1'<='0')

Opgave 2.

Nedenstående PASCAL program vil ved kørsel udskrive 4 linier hver indeholdende 2 heltal. Angiv hvilke tal der udskrives.

```
program opg2;
   var x:integer;

   procedure p(var a:integer; b:integer);
      begin
         a:=a+1;
         b:=b+1;
         writeln(a:3,b:3)
      end;

   procedure q(var c:integer; d:integer);
      begin
         c:=d-1;
         c:=c-1;
         if c>0 then q(c,c);
         p(c,d);
         writeln(c:3,d:3)
      end;
begin
   q(x,3)
end.
```

Opgave 3

En naturlig repræsentation af 5x5 matricer med reelle elementer i et PASCAL program er ved hjælp af datatypen MATRIX fastlagt ved følgende erklæring:

```
type MATRIX = array (.1..5,1..5.) of real
```

Skriv et PASCAL program, der indlæser to sådanne matricer A og B og derefter beregner og udskriver deres matrixprodukt C=AB. Det (i,j)-te element i C er givet ved:

$$c_{ij} = \sum_{k=1}^{5} a_{ik}\, b_{kj} \; .$$

Beregningen af matrixproduktet skal foretages af en procedure PRODUKT med følgende specifikation:

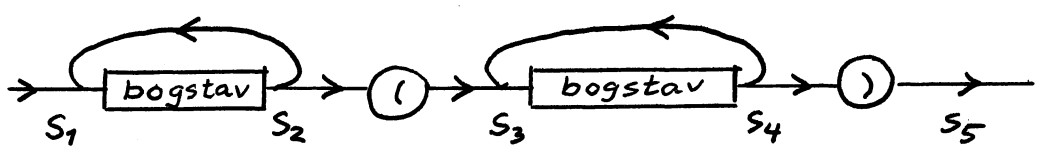
```
procedure PRODUKT(A,B:MATRIX; var C:MATRIX)
```

Opgave 4.

Nedenstående syntaksdiagram definerer udtryk af formen

funktionsnavn(argument)

hvor både funktionsnavn og argument er strenge af lutter bogstaver.



a) Hvorledes skal syntaksdiagrammet ændres, så det tillader at argumentet må være enten en streng af bogstaver eller en streng af cifre?

b) Nedenstående PASCAL program læser en linie og kontrollerer om den er i overensstemmelse med det oprindelige syntaksdiagram, idet blanktegn kun tillades i slutningen af linien. Hvorledes skal programmet ændres for at svare til det nye syntaksdiagram?

```
program opg4;

type state=(s1,s2,s3,s4,s5,error);

var s:state;
    c:char;

function bogstav:boolean;
begin
    bogstav:= ('a'<=c) and (c<='z') or ('A'<=c) and (c<='Z')
end;

begin
    s:=s1;
    while (s<>error) and not eoln do
    begin
       read(c);
       case s of
       s1:if bogstav then s:=s2 else s:=error;
       s2:if c='(' then s:=s3 else if not bogstav then s:=error;
       s3:if bogstav then s:=s4 else s:=error;
       s4:if c=')' then s:=s5 else if not bogstav then s:=error;
       s5:if c<>' ' then s:=error;
       end;
    end;
    if s=s5
    then writeln('Syntaks iorden')
    else writeln('Syntaks ikke iorden');
end.
```

Opgave 5.

Det efterfølgende PASCAL program indeholder en række syntaksfejl. Fejlmeddelelserne fra et oversættelsesforsøg følger.

AFLEVER OPGAVETEKSTEN MED RETTELSE AF SYNTAKSFEJL.

```
program opg5; /*Programmet finder mindste fælles multiplum
                af en række heltal som indtastes på en linie.*/

var n,resultat:integer;

function maxdivisor(n,m:integer):integer;
   begin
       while (n<>m) and (n>0) and (m>0) do
           if n<m then m:=m mod n else n:=n mod m;
       if n>0 then maxdivisor:=n else maxdivisor:=m;
   end;

begin
    resultat:=1;
    repeat
          read(n);
          resultat:=resultat*n/maxdivisor(resultat,n);
    until eoln;
    writeln('Mindste fælles multiplum er ',resultat);
end.
```

```
       while n<>m and n>0 and m>0
                        Å149    Å64,149
 ERROR 149: Not a valid comparison operation
 ERROR  64: Expression's type is incorrect
 ERROR 149: Not a valid comparison operation
          if n<m then m:=m mod n else n:=n mod m;
           Å64,96
 ERROR  64: Expression's type is incorrect
 ERROR  96: "DO" expected
begin
    Å8
 ERROR   8: semicolon ";" expected
          resultat:=resultat*n/maxdivisor(resultat,n);
                                                 Å125
 ERROR 125: Real to integer conversion not valid
   writeln('Mindste fælles multiplum er ';resultat);
                                        Å21      Å5,87,5
 ERROR  21: ")" expected
 ERROR   5: Symbol invalid or out of context
 ERROR  87: ":=" expected
 ERROR   5: Symbol invalid or out of context
11 ERRORS DETECTED
```

Opgave 6. *Opsplitningen i emner vil ikke forekomme mere.*

Besvar netop det af følgende 5 spørgsmål, der drejer sig om det programmel du brugte i den første gruppeopgave i forårssemestret.

Principperne skal beskrives, medens detaljerne ikke behøver at være korrekte.

ICU) Forklar, hvorledes man i ICU laver et "tomt" kort og derefter udskriver teksten

GOTISK SKRIFT

i stort format med gotiske bogstaver.

GRAPHPAK) Forklar, hvorledes man ved hjælp af APL GRAPHPAK kan lave en grafisk fremstilling af funktionen

$$f(x) = x*\sin(x)$$

for x tilhørende intervallet fra -10 til 10.

SQL) Forklar, hvorledes man ved hjælp af SQL kan opdatere en tabel, hvor der for hver bog i et bibliotek står oplysning om forfatter, titel, forlag og trykkeår.

REDUCE) Forklar, hvorledes man med REDUCE kan udregne et udtryk for p(20,x), når der for alle x og i>0 gælder

```
p(0,x) = 1
p(i+1,x) = x*(p(0,x)+ ... +p(i,x))
```

MATLAB) Forklar, hvorledes man med MATLAB kan udregne

$$\sum_{i=1}^{50} (E+i*A)^{i}$$

hvor E er 5x5-enhedsmatricen og

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

Københavns Universitet
Naturvidenskabelig embedseksamen, august 1985

DATALOGI A for matematikere

Sættet består af ialt 6 sider, med numrene 1,...,6 (siderne 4 og 5 indeholder begge et eksemplar af Opgave 5, hvoraf det ene, med påførte rettelser kan vedlægges besvarelsen).

Opgave 0 skal besvares uden brug af hjælpemidler i løbet af den første time hvorefter besvarelsen afleveres. Ved besvarelsen af de øvrige opgaver kan alle sædvanlige hjælpemidler benyttes.

Opgave 0. *Opgaver af denne karakter vil ikke forekomme mere.*

a) Forklar udtrykkene 1.-4. generations maskiner.

b) Fortæl om hjælpemidler til programudvikling, f.eks. rutediagrammer (flowcharts), beslutningstabeller (decision tables), strukturdiagrammer (structure charts) og pseudokode.

c) Karakteriser programmeringssproget FORTRAN og giv eksempler på afvigelser fra PASCAL.

d) Fortæl om datakommunikation, bl. a. teleprocessing systemer, distribueret databehandling, forskellige former for netværk, samt protokoller.

Opgave 1.

Afgør for hvert af nedenstående udtryk om det er et gyldigt PASCAL-udtryk, og angiv i bekræftende fald dets type og værdi.

```
a)  10 div 10 div 10        b)  if 'a'<'b' then 2 else 3
c)  odd(10/2)               d)  (','<':') and (ord(',')>ord(':'))
e)  sqr(sqrt(4))            f)  (false=false) = false
g)  succ(pred(true))        h)  chr(round(3.7+ord('3')))
```

Opgave 2.

Hvad bliver der udskrevet ved en kørsel af nedenstående PASCAL program ?

```
program opg2;
   procedure F(p,q:integer);
   var r:integer;
   begin
      writeln( p div q);
      r:=p mod q;
      if r>0 then F(q,r);
   end;
begin
   F(25,7)
end.
```

Opgave 3

Repræsentation af 10-dimensionale reelle vektorer og kvadratiske matricer kan i PASCAL ske ved hjælp af typerne VEKTOR og MATRIX fastlagt ved erklæringen:

```
type VEKTOR = array (.1..10.) of real;

     MATRIX = array (.1..10.) of VEKTOR
```

Skriv et PASCAL program, der indlæser en 10x10-matrix A og en 10-dimensional vektor X og udskriver værdien af den til matricen hørende kvadratiske form:

$$X^t A X = \sum_{i=1}^{10} \sum_{j=1}^{10} a_{ij} x_i x_j$$

Beregningen skal foretages af et underprogram med følgende specifikation:

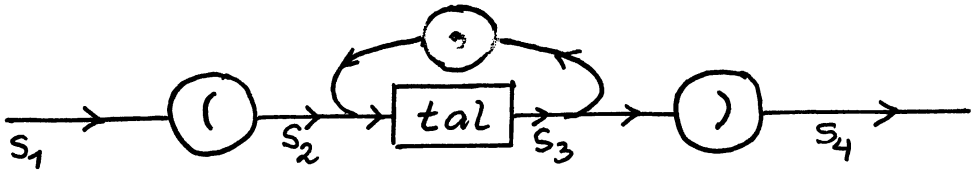
```
function KFORM(A:MATRIX; X:VEKTOR):real
```

Opgave 4.

Nedenstående PASCAL-procedure indlæser en linie fra en fil, der er åbnet udenfor proceduren, og kontrollerer, om den er af formen

(tal , tal , ... , tal)

dvs. om den svarer til syntaksdiagrammet



hvor tal betyder heltal uden fortegn.

a) Angiv en modifikation af syntaksdiagrammet, så fortegnet '-' tillades foran tal .

b) Angiv hvorledes proceduren skal ændres, så den svarer til det ændrede syntaksdiagram, og yderligere indlæser tal , tal , ... i element 1, element 2 , ... af et globalt array a erklæret ved:

var a:array(.1..100.) of integer

```
procedure readarray;

type state=(s1,s2,s3,s4,error);
var s:state;
    c:char;
    t:integer;
begin
    s:=s1; t:=1                                   and t<=100 do
    while not eoln do
        case s of
            s1: if input↑='('
                then begin read(c); s:=s2 end
                else s:=error;
            s2: if ('0'<=input↑) and (input↑<='9')          or (input↑='-')
                then begin read(t); s:=s3 end
                else s:=error;                    read(a[t]);
            s3: if input↑=','                     t:=t+1;
                then begin read(c); s:=s2 end
                else if input↑=')'
                     then begin read(c); s:=s4 end
                     else s:=error;
            s4: begin read(c); s:=error end;
            error: read(c);
        end;
    if s=s4
    then writeln('Syntaks iorden')
    else writeln('Syntaks ikke iorden');
end;
```

Opgave 5.

Det efterfølgende PASCAL program indeholder en række syntaksfejl. Fejlmeddelelserne fra et oversættelsesforsøg følger.

AFLEVER OPGAVETEKSTEN MED RETTELSE AF SYNTAKSFEJL.

```
program opg5; /*Programmet indlæser en række heltal
                og udskriver dem sorteret*/;
   var n,k,i:integer;
       a    :array(.1..100.) of integer;

begin
   n:=0;
   while not eoln do
      begin
         n:=n+1;
         read(k);
         i:=n;
         while i>1 begin
            if k<a(.i.)
            then begin
                    a(.i.):=a(.i-1.);
                    i:=i-1;
                 end;
            else begin
                    a(.i.):=k;
                    j:=0;
                 end;
         if i=1 then a(.i.):=k;
      end;
   for i:=1 to n do writeln(a(.i.));
end.
```

```
                og udskriver dem sorteret*/;
                                          Å5
 ERROR   5: Symbol invalid or out of context
   n:=0;
      Å106
 ERROR 106: Undeclared identifier
         while i>1 begin
                      Å96
 ERROR  96: "DO" expected
            else begin
               Å161
 ERROR 161: ELSE clause without associated IF statement
                    j:=0;
                    Å106
 ERROR 106: Undeclared identifier
end.
   Å3
 ERROR   3: "END" expected
6 ERRORS DETECTED
```

Opgave 5.

Det efterfølgende PASCAL program indeholder en række syntaksfejl. Fejlmeddelelserne fra et oversættelsesforsøg følger.

AFLEVER OPGAVETEKSTEN MED RETTELSE AF SYNTAKSFEJL.

```
program opg5; /*Programmet indlæser en række heltal
                og udskriver dem sorteret*/;
   var n,k,i:integer;
       a    :array(.1..100.) of integer;

begin
   n:=0;
   while not eoln do
      begin
         n:=n+1;
         read(k);
         i:=n;
         while i>1 begin
            if k<a(.i.)
            then begin
                    a(.i.):=a(.i-1.);
                    i:=i-1;
                 end;
            else begin
                    a(.i.):=k;
                    j:=0;
                 end;
         if i=1 then a(.i.):=k;
      end;
   for i:=1 to n do writeln(a(.i.));
end.
```

```
                og udskriver dem sorteret*/;
                                          Å5
 ERROR   5: Symbol invalid or out of context
   n:=0;
      Å106
 ERROR 106: Undeclared identifier
         while i>1 begin
                      Å96
 ERROR  96: "DO" expected
             else begin
                Å161
 ERROR 161: ELSE clause without associated IF statement
                    j:=0;
                    Å106
 ERROR 106: Undeclared identifier
end.
   Å3
 ERROR   3: "END" expected
6 ERRORS DETECTED
```

Opgave 6.

Den anden opgave i forårssemestret gik bl.a. ud på at implementere PASCAL programmet fra opgave 5 i efterårssemestret på PICCOLO'maskinerne. Skriv en kort brugervejledning for programmet beregnet på personer, der er fortrolige med operativsystemet CP/M.

Til støtte følger nedenfor en del af den oprindelige opgavetekst:

Opgaven gik ud på at skrive et PASCAL program, der indlæser et LP-problem på den i LP-noterne beskrevne normalform, og udskriver en basisløsning, for hvilken objektfunktionen er mindst mulig, samt minimumsværdien.

Programmet skal begynde med at læse en linie med to heltal

$$m \quad n$$

hvor m er antallet af bibetingelser (ligninger) og n er antallet af variable. Det skal kontrolleres at værdierne er rimelige. Derefter indlæses først koefficienterne i objektfunktionen

$$c_1 \quad c_2 \quad \ldots \quad c_n$$

og dernæst koefficienterne og højresider

$$\begin{array}{cccc} a_{11} & a_{12} & \ldots & a_{1n} \quad b_1 \\ \vdots & \vdots & & \vdots \quad \vdots \\ \vdots & \vdots & & \vdots \quad \vdots \\ \vdots & \vdots & & \vdots \quad \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \quad b_m \end{array}$$

i de m ligninger.

Københavns Universitet
Matematisk Institut

oktober 1985

DATALOGI A for matematikere

CPU - MASKINKODE - ASSEMBLER

Det er ved mange anvendelser af EDB nyttigt at have en vis forståelse af hvordan en datamaskine er opbygget og hvad der foregår i den, når den udfører et program. Især hvor flere programmer (eller underprogrammer), måske skrevet i forskellige programmeringssprog, skal arbejde sammen ved løsningen af en større opgave og herunder skal udveksle oplysninger, er det vigtigt at vide hvorledes dette håndteres af maskinen.

En datamaskine har følgende bestanddele:

- en centralenhed (Central Processing Unit - CPU i slang),
- et primærlager (internt lager, hurtiglager),
- et sekundærlager (eksternt lager),
- forskellige ydre enheder.

De ydre enheder knytter forbindelse mellem maskinen og omverdenen, og kan f.eks. være terminaler, printere eller plottere. Sekundærlageret bruges til langtidsopbevaring af data og programmer, primærlageret til opbevaring af data og programmer, der aktuelt skal benyttes af maskinen, mens CPU'en er den del af maskinen, der udfører de ordrer som er beskrevet i det program maskinen arbejder på. CPU'en har direkte adgang til lagerpladserne i primærlageret, den kan 'læse' indholdet på en specificeret plads og den kan 'skrive' et bestemt indhold et specificeret sted. Sekundærlageret er ikke på samme måde tilgængeligt for CPU'en: for at benytte information placeret i sekundærlageret, skal denne først flyttes til primærlageret (hvilket ofte foregår uden CPU'ens direkte medvirken), hvorefter CPU'en kan læse den. Tilsvarende skal information, der skal gemmes i det sekundære lager først skrives i primærlageret og derefter flyttes til sekundærlageret.

Maskinens primærlager er opbygget af et (stort) antal elektroniske kredsløb - flip/flops, der kan være i netop een af to tilstande, f.eks. have et af to mulige spændingsniveauer på en bestemt ledning. Information repræsenteres i lageret ved at specificere tilstandene for bestemte af disse flip/flops, og man kan tænke på den som mønstre af 0'er og 1'taller (bit). Sekundærlageret er typisk

opbygget af magnetiserbart materiale (bånd eller plader), hvor information repræsenteres ved retningen af magnetiseringen på udvalgte positioner, og igen kan man tænke på den som mønstre af 0'er og 1'er.

Til specifikation af de operationer som CPU'en skal foretage benyttes et specielt sprog: maskinkode. Dette maskinkodesprog er fastlagt ved konstruktionen af CPU'en, og definerer sammen med den fysiske udformning af CPU'en det repertoire af operationer - maskinordrer - som CPU'en kan udføre. En maskinordre skal, for at kunne 'forstås' og udføres af CPU'en, repræsenteres i maskinen, altså i princippet som 0'er og 1'er. Denne repræsentation af CPU'ens handlinger er imidlertid vanskelig at benytte for mennesker, og der er derfor for hver type CPU udviklet symbolske versioner af maskinkode - såkaldte ASSEMBLERsprog - som muliggør en for mennesker lettere (dog ikke let) læselig specifikation af CPU ordrerne. Programmer skrevet i ASSEMBLERsprog kan ikke direkte udføres af maskinen, men skal ligesom f.eks. PASCAL programmer 'oversættes' til maskinkode før de kan udføres. Denne oversættelse foretages af et specielt program - en ASSEMBLER, men oversættelsen har en ganske anden karakter end oversættelsen af et PASCAL program, idet der i almindelighed er en en-til-en korrespondance mellem ordrerne i ASSEMBLERprogrammet og maskinkodeordrerne i det oversatte program.

Hensigten med det følgende er (kort) at give en nærmere beskrivelse af en CPU's virkemåde og med IBM 4341 maskinen (det såkaldte System/370) som eksempel omtale nogle maskinkodeordrer og deres ASSEMBLERsprogsversioner. De omtalte principper for såvel CPU-virkemåde som for ASSEMBLERsprog gælder dog for de fleste (alle) datamaskiner, og der henvises til Politikens EDB bog for yderligere baggrundsviden.

For at konkretisere gives et simpelt modelprogram skrevet i ASSEMBLER. Dette ASSEMBLERprogram kan kaldes som et underprogram, f.eks. af et PASCALprogram. Fremgangsmåden for dette bliver nærmere beskrevet i noten om OPERATIVSYSTEMER.

CPU'en.

Centralenheden (eller CPU'en) består af en overvågningsdel - kontrolenheden (Control Unit - CU), en beregningsdel - den aritmetisk-logiske enhed (Arithmetic/Logical Unit - ALU) og en række registre , og CPU'en er i direkte forbindelse med det primære lager.

Primærlageret i IBM 4341 er organiseret i celler på 8 bits, de såkaldte bytes, og hver celle har en adresse (et nummer) som benyttes når CPU'en skal referere til den. Ønsker CPU'en f.eks. at læse indholdet af en celle sker dette på følgende måde: først anbringer CPU'en adressen på den pågældende celle i et specielt adresseregister , hvorefter CPU'en beder om at læse indholdet af den celle, som adresseregisteret 'peger' på.

Til adressering af cellerne i det primære lager benyttes adresserne fra 0 til 16777215 (som er (2 opløftet til 24) minus 1), hvilket svarer til, at en primærlageradresse kan angives ved 24 bit. Maskinen kan derfor addresse et primærlager på 16M (megabytes). I virkeligheden er primærlageret 'kun' på 8 megabytes, men IBM 4341 maskinen er (i lighed med en række andre moderne datamaskiner)

udstyret med en speciel facilitet: virtuelt lager , som betyder, at maskinen 'simulerer' et større adresseområde end den reelt råder over: de adresser som et program kan benytte sig af, 'oversættes' ved udførelsen af programmet til de faktisk eksisterende maskinadresser og dele ("pages") af det tilsyneladende primærlager kan læses ind og ud fra sekundærlageret, hvorved mange brugere samtidig kan anvende maskinen.

CPU'ens virkemåde kan beskrives som en 'uendelig' løkke med følgende indhold:

- læs den næste maskinordre,
- fortolk maskinordren dvs. find ud af hvad der skal gøres,
- udfør maskinordren.

Udførelsen af denne løkke styres af kontrolenheden, som har et register ordretælleren (se PO side 270), eller programtælleren , hvis indhold er adressen på den næste instruktion som CPU'en skal udføre. Normalt vil maskinordrerne være placeret sekventielt i lageret og blive udført i den rækkefølge som de står i, og kontrolenheden vil i dette tilfælde sørge for, at programtælleren kommer til at pege på den næste maskinordre, inden der startes forfra i løkken. I andre tilfælde kan den maskinordre, der skal udføres, være et hop til et helt andet sted i programmet (svarende til GOTO sætninger i PASCAL), og maskinordrens virkning er i dette tilfælde, at programtællerens indhold ændres, så den ordre, der indlæses ved næste gennemløb af løkken, netop er den ønskede.

Derudover kan den normale programafvikling midlertidigt blive suspenderet via såkaldte interrupts. Ofte vil en CPU skulle løse en række forskellige opgaver parallelt, og da disse kan have lange døde venteperioder, f.eks. indtil en der bruger editoren trykker ENTER, eller indtil en påbegyndt transmission mellem primær- og sekundærlageret er tilendebragt, er det praktisk at lade CPU'en arbejde videre med andre aktive opgaver, indtil det ved hjælp af et interrupt bliver meddelt, at et ventende program kan fortsætte.

Den anden vigtige del af CPU'en, den aritmetisk/logiske enhed, er involveret i selve udførelsen af de fleste maskinordrer, og ALU'en benytter hertil en række regneregistre. Visse registre, de såkaldte generelle registre , er beregnet til arbejde med hele tal (herunder forskellige udregninger i forbindelse med beregning af adresser på data og maskinordrer), mens andre bruges til regning med flydende tal. Der findes 16 generelle registre, nummereret 0,1,...,15 , hver af længde 32 bits, samt 4 registre til regning med 'flydende' tal, hver af længde 64 bits, nummereret 0,2,4 og 6. Vi skal ikke komme nærmere ind på brugen af registrene for flydende beregninger.

Endelig findes et register, det såkaldte flagregister , hvor forskellige oplysninger om forløbet af f.eks. aritmetiske operationer placeres: om resultatet er 0, om det er negativt, om en operation har forårsaget overløb (at f.eks. en addition har givet et resultat, der er for stort til at kunne repræsenteres i et register). Afhængigt af indholdet af dette register kan programmet så vælge en blandt flere fortsættelser.

NOGLE UDVALGTE MASKINORDRER.

Nedenfor er omtalt nogle maskinordrer eller rettere deres ASSEMBLERsprogsversioner, hovedsagelig de maskinordrer, der benyttes i ASSEMBLER modelprogrammet, med en kort beskrivelse af deres virkning.

Den typiske maskinordre er opbygget af to dele: en operationsdel på 8 bit (en byte), der specificerer hvad der skal udføres, og en adressedel , der specificerer adresser på de data som operationen skal arbejde med.

I ASSEMBLERsprog angives en maskinordres operationsdel ved en kode, der ofte er en forkortelse for maskinordrens virkning. Ved specifikation af adressedelen, der i selve maskinordren er tal (skrevet binært), benyttes forskellige konventioner.

Den simpleste type adresseangivelse er for data, der findes i de generelle registre. Her angives data ved numrene på de registre, som indeholder data, f.eks.

```
AR  4,5
```

(AR står for Add Register), som bevirker, at indholdet af register 5 adderes til indholdet af register 4. Efter udførelsen af denne maskinordre indeholder register 4 summen af de (gamle) indhold af register 4 og register 5, mens register 5 er uændret. En maskinordre af denne type, i det såkaldte RR-format, fylder altså ialt 2 bytes, nemlig 1 byte til operationsdelen og 1 byte (4 bit til hver registerangivelse) til adressedelen.

Addressering af data, der befinder sig i primærlageret er lidt mere kompliceret. Typisk er der brug for at referere til data i et register R1 samt til data i en af primærlagerets lagerceller. Primærlageradressen er i princippet opbygget på følgende måde: indholdet af et register benyttes som basisadresse B, indholdet af et andet register benyttes som index for adressen I, medens en tredie del af adressespecifikationen er en displacement D (på 12 bit, svarende til tallene 0,1,...,4095), og den 'effektive' adresse (dvs. den adresse som maskinen bruger ved datareferencen) er så summen: B+I+D af disse tre størrelser. Angivelse af register 0 som indexregister eller basisregister betyder dog, at den pågældende adressedel bortfalder. I modelprogrammet benyttes muligheden for indicering ikke samtidig med et basisregister, men det er kun fordi programmet er så simpelt. Maskinordrer i det såkaldte RX-format, fylder altså 32 bit (4 bytes), nemlig 1 byte til operationsdelen, 12 bit til angivels af de 3 registre (4 bit til hvert nummer på et register), et dataregister, et basisregister B og et indexregister I, og sluttelig 12 bit til angivelse af displacement D. Det generelle udseende af en RX-format maskinordre er således:

```
OPKODE R1,D(I,B)
```

hvor OPKODE specificerer den operation der skal udføres, R1 er nummeret på dataregisteret, D er talværdien for displacement (i bytes) mens I og B er numrene på henholdsvis index- og basisregister. Benyttes basismuligheden ikke, udelades blot B-delen. F.eks. bevirker maskinordren:

```
ST   2,24(13)
```

(ST står for STore), at indholdet af register 2 placeres i lagercellen hvis adresse er summen af 24 og indholdet af register 13. Tilsvarende bevirker maskinordren:

```
A    6,0(2)
```

(A står for Add), at indholdet af lagercellen med adresse lig indholdet af register 2 (plus 0) adderes til indholdet af register 6, og maskinordren:

```
L    6,40(13)
```

(L står for Load), bevirker at indholdet af lagercellen hvis adresse er 40 plus indholdet af register 13, placeres i register 6.

En tredie vigtig type af maskinordrer er hopordrerne, eller branching ordrer. Disse hopordrer kan være ubetingede eller deres udførelse kan være afhængig af en betingelse, og adressen på det sted hvortil der skal hoppes kan enten angives som indholdet af et register f.eks.:

```
BR   14
```

(BR står for BRanch), som bevirker, at der foretages et (ubetinget) hop til den i register 14 indeholdte adresse, eller den kan angives på samme måde som ovenfor i formen D(I,B) f.eks.:

```
BNZ  LOOP
```

(BNZ står for Branch if Not Zero), (hvor LOOP af ASSEMBLER'en oversættes til en adresse i D(I,B) format, se nedenfor), som bevirker, at der hoppes til LOOP hvis resultatet af den foregående aritmetiske operation er forskelligt fra nul.

En stor del af maskinens arbejdskraft bruges til 'flytning' af data: ofte skal data befinde sig i et af de generelle registre før CPU'en kan udføre de 'interessante' operationer med disse data. Mange delopgaver har derfor følgende struktur:

- flyt data fra lageret til registre,
- udfør operationerne på registerindholdet,
- flyt registerindholdet tilbage til lageret.

Selvom det nok er nemmest at tænke på dataflytning, er der i virkeligheden tale om kopiering, hvor det altså kun er det sted der flyttes til der ændres, mens det sted der 'flyttes' fra er uændret efter operationen.

Et ASSEMBLERprogram indeholder udover selve maskinkodeordrerne (i deres ASSEMBLERversioner) en række 'direktiver', der benyttes ved oversættelsen af programmet til maskinkode. Disse kan være programnavnet samt start og slut på ASSEMBLERprogrammet. Vigtigere er måske direktiver i forbindelse med anvendelse af symbolske navne i assemblerprogrammet. Det er muligt at benytte navne på bl.a. adresser og konstanter, og for at ASSEMBLER'en kan oversætte disse

symbolske navne til maskinadresser, der ofte involverer et basisregister, skal nummeret på det register der skal bruges som basisregister angives. Direktivet

```
USING  *,15
```

bevirker netop, at register 15 benyttes som basisregister i den pågældende del af maskinkoden.

UNDERPROGRAMMER.

Et program (f.eks. et PASCALprogram), der kalder et underprogram reserverer plads til at gemme det aktuelle indhold af alle registre (mens underprogrammet kører). Addressen på starten af dette gemmeområde findes ved underprogramkaldet i register 13. Det kaldte underprogram må så for at sikre en fornuftig tilbagevenden, gemme alle registres indhold i dette område, og inden der returneres retablere registrene til den tilstand de havde før kaldet. Rækkefølgen af registrene i dette gemmeområde er følgende: 12 bytes fra starten placeres indholdet af register 14, og derefter (hvert register fylder 4 bytes) registrene 15,0,1,...,12. Videre indeholder register 14 adressen på det sted hvorfra underprogrammet blev kaldt, og register 15 indeholder startadressen på det kaldte underprogram medens register 1 indeholder adressen på starten af en eventuel liste med parametre.

Fremgangsmåden ved et underprogramkald indeholder altså følgende aktiviteter fra underprogrammets side (se også PASCAL-manualerne):

- gemme indholdet af de registre som underprogrammet modificerer i kaldeprogrammets gemmeområde,
- gemme startadressen på dette gemmeområde i lokalt gemmeområde (eventuelt et register),
- udføre sine pligter (arbejde på de værdier der er overført til underprogrammet via parametre),
- hente startadressen på det kaldende programs gemmeområde,
- retablere registrene,
- hoppe tilbage.

MODELPROGRAMMET.

Nedenstående ASSEMBLERprogram har følgende funktion. Ved kald af programmet venter det at få overført tre parameterværdier:

- et array af INTEGER's,
- længden af dette array,
- en parameter hvori programmets resultat kan gives tilbage.

Programmet beregner summen af array'ets elementer og leverer denne sum til det kaldende program i den tredie parameter.

Register 15 indeholder startadressen på underprogrammet, og ASSEMBLER-direktivet: USING *,15 , bevirker, at dette register benyttes som basisregister.

Register 1 indeholder adressen på det kaldende programs parameterområde. De første 4 bytes i dette parameterområde

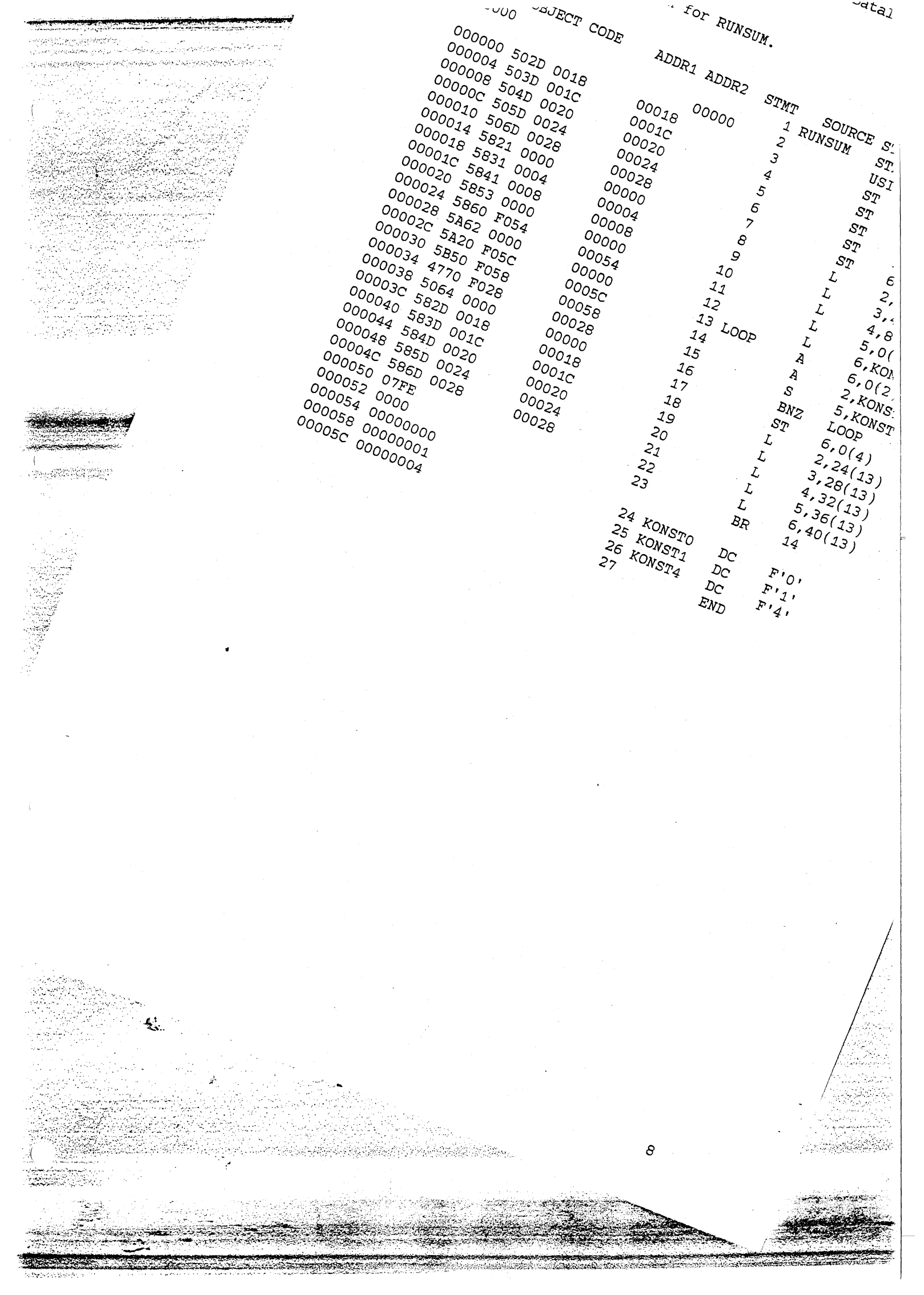
indeholder adressen på starten af array'et (hvorfra elementerne er placeret fortløbende), de næste 4 bytes indeholder adressen på den næste parameter og de følgende 4 bytes indeholder adressen på resultatparameteren.

Programmet starter med at gemme registrene 2,...,6 i kaldeprogrammets gemmeområde. Derefter initialiseres diverse registre, og selve summationen foregår i en løkke (LOOP ordren og de følgende 3 ordrer). Sluttelig gemmes resulatet og de benyttede registre retableres inden der vendes tilbage til det kaldende program.

ASSEMBLERprogrammet RUNSUM (findes også på B-disken):

```
RUNSUM   START 0                  ** program navn og hoved
         USING *,15               ** brug reg. 15 som basisreg.
         ST    2,24(13)           ** gem reg. 2
         ST    3,28(13)           ** gem reg. 3
         ST    4,32(13)           ** gem reg. 4
         ST    5,36(13)           ** gem reg. 5
         ST    6,40(13)           ** gem reg. 6
         L     2,0(1)             ** array startadr. til reg. 2
         L     3,4(1)             ** længdeadr. til reg. 3
         L     4,8(1)             ** resultatadr. til reg. 4
         L     5,0(3)             ** længde til reg. 5
         L     6,KONST0           ** nulstil reg. #6
LOOP     A     6,0(2)             ** adder arrayelem. til reg. #6
         A     2,KONST4           ** adr. på næste elem. til reg. 2
         S     5,KONST1           ** træk 1 fra reg. 5
         BNZ   LOOP               ** hop til LOOP hvis <>0
         ST    6,0(4)             ** reg. 6 til 3. parameter
         L     2,24(13)           ** retabler reg. 2
         L     3,28(13)           ** retabler reg. 3
         L     4,32(13)           ** retabler reg. 4
         L     5,36(13)           ** retabler reg. 5
         L     6,40(13)           ** retabler reg. 6
         BR    14                 ** hop tilbage
KONST0   DC    F'0'               ** definition af 0
KONST1   DC    F'1'               ** definition af 1
KONST4   DC    F'4'               ** definition af 4
         END
```

Til illustration af den 'ægte' maskinkode som ASSEMBLER'en frembringer ud fra dette ASSEMBLERprogram er nedenfor medtaget et udsnit af den LISTING fil, som ASSEMBLER'en producerer samtidig med maskinkode filen (filtype TEXT). Den første kolonne indeholder maskinkodens relative adresser, altså adresser beregnet ud fra programmets start, mens de to næste kolonner indeholder den frembragte maskinkode. Både adresser og maskinkode er angivet hexadecimalt, dvs. i 16-talsystemet, hvor talværdierne 0,1,...,15 angives ved de hexadecimale 'cifre': 0,1,...,9,A,B,...,F . Ved hjælp af to hexadecimale cifre kan så hvert tal mellem 0 og 255, altså hvert muligt indhold af en byte, skrives. F.eks. er operationskoden for en ST(ore)-maskinordre det hexadecimale tal 50 (=80 i 10-talsystemet) svarende til bitmønsteret: 01010000 .


for RUNSUM.
OBJECT CODE
ADDR1 ADDR2 STMT SOURCE ST
1 RUNSUM
000000 502D 0018
000004 503D 001C
000008 504D 0020
00000C 505D 0024
000010 506D 0028
000014 5821 0000
000018 5831 0004
00001C 5841 0008
000020 5853 0000
000024 5860 F054
000028 5A62 0000
00002C 5A20 F05C
000030 5B50 F058
000034 4770 F028
000038 5064 0000
00003C 582D 0018
000040 583D 001C
000044 584D 0020
000048 585D 0024
00004C 586D 0028
000050 07FE
000052 0000
000054 00000000
000058 00000001
00005C 00000004
00018
0001C
00020
00024
00028
00000
00004
00008
00000
00054
00000
0005C
00058
00028
00000
00018
0001C
00020
00024
00028
00000
13 LOOP
BNZ
BR
24 KONST0
25 KONST1
26 KONST4
27
DC
DC
DC
END
F'0'
F'1'
F'4'
14
6,0(4)
2,24(13)
3,28(13)
4,32(13)
5,36(13)
6,40(13)
LOOP
2,KONST
5,KONST

Københavns Universitet
Matematisk Institut

oktober 1985

DATALOGI A for matematikere

OPERATIVSYSTEMER.

Der henvises til afsnittet om basisprogrammel i (PO) side 273-283, hvor man finder en generel gennemgang af operativsystemets opgaver. Nedenfor følger nogle bemærkninger om de operativsystemer, som benyttes til forårets opgaver på universitetets IBM 4341, RECKU's SPERRY 1100 og DIKU's piccolo'er.

IBM 4341.

Som nævnt i (PG) er VM operativsystemet todelt, idet der er et ydre operativsystem CP, som tager sig af tildeling af ressourcer (og kommunikation mellem brugere), og en indre del CMS, der nærmest svarer til f.eks. CP/M eller MS-DOS på mikrodatamater med kun en bruger. I princippet kan den enkelte bruger udskifte CMS-delen med et andet operativsystem, f.eks. således at maskinen fungerer som om den kører under CP/M.

Større programmer vil ofte bestå af et hovedprogram og en række underprogrammer, som ikke behøver at være programmeret i samme sprog. Når de skal bringes til at køre som et samlet program sker det i tre faser: 1. oversættelse, 2. indlæsning og 3. start.

1.) Først oversættes hovedprogrammet og underprogrammerne hver for sig.

ASSEMBLERprogrammer gives filtype ASSEMBLE. Er filnavnet PROG sker oversættelsen ved kommandoen

```
ASSEMBLE PROG .
```

FORTRANprogrammer gives filtype FORTRAN. Er filnavnet PROG sker oversættelsen ved kommandoen

```
FORTVS PROG .
```

PASCALprogrammer gives filtype PASCAL. Er filnavnet PROG sker oversættelsen ved kommandoen

PASCALVS PROG .

Resultatet af oversættelserne er ikke køreklare programmer, men filer af type TEXT. Disse indeholder næsten færdige programmer med to afvigelser:

Programmerne skal kunne anbringes hvorsomhelst i primærlageret (de skal være relokerbare) og filerne indeholder derfor oplysninger om steder i programmet med adresser, der må modificeres, når det er besluttet, hvor i primærlageret programmet skal placeres.

Programmerne skal kunne samarbejde med andre programmer ved at de gensidigt indeholder adresser på programpunkter i hinanden. Dette effektueres ved at TEXT- filerne indeholder symbolske navne på visse af sine programpunkter. F.eks. bliver etiketten på START direktivet i et assemblerprogram til et sådant eksternt navn og navnene på FORTRAN og PASCAL programmer fungerer analogt. Andre programmer kan referere til disse programpunkter via de symbolske navne, idet TEXT-filerne også indeholder oplysninger om steder, hvor de endelige adresser knyttet til symbolske navne skal indsættes som såkaldte adressekonstanter.

2.) Indlæsning i primærlageret og sammenkædning af f.eks. et hovedprogram MAIN og to underprogrammer SUB1 og SUB2, der alle findes som TEXT-filer, sker med kommandoen

LOAD MAIN SUB1 SUB2 .

3.) Når det færdige program skal køres skrives

START navn

hvor navn er det symbolske navn på det programpunkt, hvor man vil starte. Navnet kan som regel, men ikke altid udelades fra START-kommandoen. Hvis det kan udelades, kan man også få LOAD-kommandoen til at starte programmet som angivet i (PG) side 39.

I mange tilfælde ønsker man at benytte underprogrammer fra programbiblioteker i stedet for underprogrammer man selv har skrevet. TEXT-filerne hørende til sådanne biblioteker er normalt samlet i særlige filer af type TXTLIB. Man kan angive at eksterne referencer skal søges opfyldt i sådanne biblioteker med en GLOBAL-kommando som angivet i (PG) side 38.

Når et hovedprogram skrevet i PASCAL skal kalde underprogrammer i f.eks. FORTRAN eller ASSEMBLER erklæres disse med en procedurekrop, der kun består af ordet FORTRAN. Dette definerer, at der er tale om et eksternt underprogram, der skal kaldes på den til operativsystemet hørende standardmåde (i modsætning til eksterne underprogrammer skrevet i PASCAL, se IBM's manualer). Parametrene skal være af type VAR eller CONST, som ikke er standard-PASCAL, men kan benyttes i PASCALVS som vist i eksemplet nedenfor.

I GLOBAL-kommandoen må medtages TEXT-biblioteket

PASCALVS

for at de nødvendige biblioteksfunktioner etc. kan blive medtaget.

Antag f.eks. at ASSEMBLERprogrammet RUNSUM fra afsnittet om assemblerprogrammering ligger i filen

```
RUNSUM ASSEMBLE A
```

Afprøvning kan ske ved at vi i TEST PASCAL A indtaster følgende pascalprogram:

```
program sumtest;
const n=10;
type n_array=array(.1..n.) of integer;
var a:n_array;
    i,resultat:integer;
procedure sum(const a:n_array;const n:integer;var resultat:integer);
          FORTRAN;
begin
     for i:=1 to n do a(.i.):=i;
     sum(a,n,resultat);
     writeln(resultat);
end.
```

De to programmer oversættes med kommandoerne

```
ASSEMBLE RUNSUM
PASCALVS TEST
```

Programmerne indlæses i primærlageret med kommandoerne

```
GLOBAL TXTLIB PASCALVS
LOAD TEST SUM .
```

Inden kørslen startes kan man fortælle, at OUTPUT-filen skal være terminalen med en FILEDEF-kommando

```
FILEDEF OUTPUT TERMINAL
```

Kørslen kan nu startes med

```
START SUMTEST .
```

(Under oversættelserne opstår to filer med programlister,

SUM LISTING A og TEST LISTING A ,

og under "loadningen" opstår en fil

```
LOAD MAP
```

der angiver maskinadresserne, som knyttes til de symbolske eksterne navne).

Efter LOAD-kommandoen kan man gemme det færdige maskinprogram i en fil RUNSUM MODULE ved kommandoen

```
GENMOD RUNSUM .
```

Dette program kan herefter bringes til udførelse med kommandoen

RUNSUM .

Metoden kan kun anbefales til programmer, der anvendes meget, da et sådant færdigt program med alle hjælpeunderprogrammer fylder godt op.

SPERRY 1100.

Operativsystemet som benyttes hedder OS1100. Vi vil her kun beskæftige os med de dele af det, som benyttes til simple batch-kørsler afsendt fra IBM4341. Iøvrigt henvises til RECKU's publikationer, hvoraf en del er tilgængelige i terminalrummet.

GENERELT OM KOMMANDOER.

En kommando består af følgende:

1. Tegnet "@" , som skal være det første tegn på linien. (Dette tegn findes ikke i EBCDIC, men transmissionsprogrammerne, som sender filer til RECKU, "oversætter" skråstregen, som går fra oven nedad mod højre, til dette tegn, som kaldes masterspace).

2. Et kommandoord , som definerer den operation, som ønskes udført (den processor man ønsker aktiveret).

3. En option-liste bestående af et komma efterfulgt af et eller flere bogstaver. Optionlisten specificerer overfor systemet, hvordan den pågældende operation ønskes udført. Option-listen kan udelades.

4. En parameterliste bestående af et eller flere blanktegn efterfulgt af en række parametre, som adskilles med komma. Parametrene specificerer overfor den pågældende processor, hvad den skal behandle, og vil som oftest være et filnavn. I visse tilfælde opdeles et parameterfelt i underparametre som adskilles med "/".

ET UDVALG AF KOMMANDOER.

@RUN runid,konto/brugerid,projekt,tid,sider

Kommandoen bruges til at starte en kørsel.

runid Identifikaton af kørslen. Hvis denne begynder med HCO vil uddata blive sendt med post til ekspeditionskontoret på H.C.Ørsted Institutet.

konto/brugerid Identifikation af brugeren. Tildeles af RECKU.

projekt Projektidentifikation. Ved afsendelse af job fra IBM4341 anbefales det, at brugeridentifikationen på denne anvendes som projektidentifikation, idet man derved sikrer, at der under alle omstændigheder kommer uddata tilbage fra kørslen.

tid	maksimal antal sekunder/minutter som kørslen får lov at bruge. Ønskes specificeret et antal sekunder skal tallet indledes med et "S". (Den benyttede tidsenhed er fiktiv, idet forbrug af forskellige resourcer indgår).
sider	Det maksimale antal sider som kørslen må udskrive.

@PASSWD løsen

Kommandoen opgiver brugerens løsen til systemet.

@FTN,I programnavn

Oversættelse af et FORTRANprogram, som følger umiddelbart efter frem til den næste kommando til operativsystemet, som identificeres ved at første tegn er "@". Det oversatte program svarer til TEXT-filerne i IBM-maskinen. Det får navnet, som angives i kommandoen.

@MAP,I ,programnavn

Kommandoen bevirker, at der laves et køreklart program svarende til filerne af type MODULE på IBM-maskinen. Hovedprogrammet er det i kommandoen angivne program. Standard hjælperutiner til f.eks. ind- og udlæsning vil automatisk blive medtaget, medens andre biblioteker må specificeres ved hjælp af linier af formen

LIB biblioteksnavn

som skal følge efter kommandoen inden den næste kommando til operativsystemet.

@XQT programnavn

Kommandoen starter udførelsen af et færdigt program. Bemærk, at man ikke angiver filtyper, som i CMS-systemet. Det samme programnavn kan benyttes til det oprindelige kildeprogram, til det oversatte program inden sammenkædningen med underprogrammer, og til det færdige program.

P*A.SCAL

Kommando til kald af RECKU's "load and go" PASCAL-system (som er lavet på DIKU og tidligere brugt til DATO undervisning). Til systemet hører bl.a. kommandoen @PAS, der fortæller, at der følger et PASCALprogram, kommandoen @GO, der starter kørsel med det oversatte PASCALprogram og indleder data til programmet, samt kommandoen @ENDX, der afslutter brugen af systemet. Et eksempel følger nedenfor.

AFSENDELSE OG MODTAGNING AF BATCHKØRSLER.

Når forårets opgaver nødvendiggør kørsel på RECKU, vil kursusdeltagerne få sendt en fil

RUNSTART SPERRY

som indeholder en RUN-kommando og en PASSWD-kommando. Når kommandoen

PUNSP filnavn filtype

anvendes på en fil (evt. fra fillisten), vil denne blive afsendt til RECKU forsynet med RUN-kommandoen og PASSWD-kommandoen som de to første linier. Uddata fra kørslen kommer tilbage til afsenderens "reader", og modtages ved at man udfor filen i reader-listen taster

RECEIVE / filnavn filtype

hvor filnavn og filtype vælges efter behag.

Vi kan f.eks. tænke os at filen RUN SPERRY A har følgende udseende:

```
@P*A.SCAL
@PAS
program demo;
var i:integer;
begin
    while not eof do begin
        readln(i);
        writeln(i,sqrt(i):10:5);
      end;
end.
@GO
   1
   2
   3
@ENDX
```

Vi forudsætter at der eksisterer en fil RUNSTART SPERRY A med gyldigt kontonummer, brugerid og løsen. Man afsender da ovennævnte fil med kommandoen

PUNSP RUN SPERRY

og modtager senere en fil af følgende udseende (bortset fra visse indledende og afsluttende øvelser).

```
@P*A.SCAL
Pascal 1100A/9R1 started at: Tuesday, 1985 October 22, 13:04:32


@PAS
No errors were found and no warnings given
INPUT and OUTPUT are ASCII files
Size (lines/code/procedures/stack): 8/43/0/3072, time used: 107 ms
```

```
@GO
1    1.00000
2    1.41421
3    1.73205
User program terminated normally, resources used: 5 ms / 0 pages
Pascal 1100A/9R1 terminated: Tuesday, 1985 October 22, 13:04:32
```

PICCOLOERNE.

Her henvises til materiale om CP/M og COMPAS systemet, som også bruges ved DAT 0. Dette vil blive udleveret til foråret.

Københavns Universitet
Matematisk Institut

november 1985

DATALOGI A for matematikere

FORTRAN

Formålet med disse noter er at sætte læseren istand til at forstå og modificere FORTRAN programmer. Der henvises iøvrigt til omtalen af FORTRAN i (PG) og (WG). Mange flere detailler kan ses i FORTRAN manualerne i terminalrummet.

Opstilling af FORTRAN programmer.

Linier med 'C' i position 1 opfattes som kommentarer og ignoreres.

Et FORTRAN program består af en række sætninger (statements). Hver sætning fylder normalt 1 linie. Position 1-5 på hver linie er reserveret til sætningsnumre (labels, der ikke skal erklæres som i PASCAL). Position 6 skal være tom medmindre man med et ikke blankt tegn angiver, at linien er en fortsættelseslinie. Position 7-72 benyttes til det egentlige sætningsindhold, medens position 73-80 eventuelt kan benyttes til linienummerering.

Datatyper og erklæringer i FORTRAN.

De grundlæggende datatyper er

INTEGER	Heltal
REAL	Enkelt nøjagtighed flydende tal
DOUBLE PRECISION	Dobbelt nøjagtighed flydende tal
COMPLEX	Enkelt nøjagtighed komplekst tal
LOGICAL	Svarer til BOOLEAN
CHARACTER	Svarer til CHAR
CHARACTER*n	Svarer til PACKED ARRAY(.1..n.) OF CHAR;

Der er desuden på mange maskiner mulighed for at benytte f.eks.

INTEGER*2	2 bytes heltal
INTEGER*4	4 bytes heltal ligesom INTEGER
REAL*4	4 bytes flydende tal ligesom REAL
REAL*8	8 bytes flydende tal ligesom DOUBLE PRECISION
REAL*16	16 bytes flydende tal

Variabelnavne må højst være seks tegn. Variable erklæres med sætninger som

```
DOUBLE PRECISION A,B(5),C(0:7),D(5,0:7)
```

svarende til

```
var A: real;
    B: array(.1..5.) of real;
    C: array(.0..7.) of real;
    D: array(.1..5,0..7.) of real;
```

i PASCAL.

Bemærk, at matricer i FORTRAN lagres søjlevis, medens de i PASCAL lagres rækkevis. Dette har betydning, når FORTRAN underprogrammer kaldes fra PASCAL programmer.

Ved opskrivning af array-elementer benyttes blot almindelige parenteser '(' og ')' i stedet for PASCAL's '(.' og '.)'.

Brugerdefinerede skalartyper, records, pointers og sets eksisterer ikke i FORTRAN.

Variabelnavne kan benyttes uden at de er erklærede, idet de da implicit antages at være af type INTEGER, hvis navnet begynder med I,J,K,L,M,N og ellers REAL. Denne regel kan dog modificeres med en sætning som

```
IMPLICIT DOUBLE PRECISION (A-H),(O-Z)
```

hvis man f.eks.kun vil regne med dobbelt nøjagtighed flydende tal. Implicit erklærede variabelnavne kan knyttes til arrays ved DIMENSION sætninger som

```
DIMENSION H(100,10),K(100)
```

Konstanter i FORTRAN.

De logiske konstanter benævnes .TRUE. og .FALSE.

Konstanter erklæres med PARAMETER-sætninger som f.eks.

```
PARAMETER ( NMAX=6 , MMAX=6 )
```

Man kan med DATA-sætninger sørge for, at variable fra starten har fastlagte værdier. En DATA-sætning indeholder variabellister og tilhørende konstantlister som f.eks.

```
DATA A/1.2/,B/5*1.0/,(C(I),I=0,3)/10.0,20.0,30.0,40.0/
```

hvor A, B og C tænkes erklæret som tidligere. I de fem elementer i B indsættes 1.0, og i de fire første elementer i C indsættes 10, 20, 30 og 40.

Udtryk i FORTRAN.

Hierarkiet af operatorer er følgende:

1. (Funktionsapplikation)
2. ** (Potensopløftning)
3. * og /
4. + og -
5. .GT. .GE. .LT. .LE. .EQ. .NE. (> >= < <= = <>)
6. .NOT.
7. .AND.
8. .OR.

Lighedstegnet '=' benyttes som ':=' i PASCAL. Venstresiden må ikke være et helt array. Der er ikke strengt typecheck som i PASCAL, så man kan f.eks. flytte en REAL (der trunkeres) over i en INTEGER.

Kontrolstrukturer i FORTRAN.

Løkker konstrueres normalt med DO-sætninger, der svarer til FOR sætninger i PASCAL:

```
DO  stn  i = e1 , e2 , ( , e3 )
```

hvor

stn	nummeret på sidste sætning i løkken
i	løkkevariablen (kan være af type REAL)
e1	udtryk, der evalueres til startværdien
e2	udtryk, der evalueres til slutværdien
e3	udtryk, der evalueres til skridtlængden (kan udelades og sættes da til 1).

Hvis skridtlængden er positiv/negativ gennemløbes løkken indtil løkkevariablen er større/mindre end slutværdien.

GOTO fungerer som i PASCAL.

Der er desuden mulighed for at anvende "computed goto", der nærmest er en simpel CASE-konstruktion. Sætningen

```
GOTO (100,200,300),N
```

får programmet til at hoppe til sætning 100, 200 eller 300 afhængigt af om N er 1, 2 eller 3. Ellers fortsættes med næste sætning.

Betingelser udtrykkes oftest ved hjælp af IF-sætninger af formen

```
IF ( udtryk med logisk værdi ) sætning
```

hvor sætningen til højre udføres, hvis den logiske værdi er sand.

Skal f.eks. MAXB sættes lig det største af tallene

B(N1) , B(N1+1) , ... , B(N2)

kan det f.eks. ske således:

```
      MAXB = B(N1)
      DO 101 I = N1+1 , N2
      IF ( B(I) .GT. MAXB ) MAXB = B(I)
  101 CONTINUE
```

hvor CONTINUE-sætningen intet udfører, men blot afslutter løkken.

Konstruktioner som REPEAT...UNTIL og WHILE...DO i PASCAL må klares med GOTO-er, men der er (fra FORTRAN 77) mulighed for IF...THEN og IF...THEN...ELSE konstruktioner ved hjælp af følgende 4 sætningstyper:

```
IF ( udtryk med logisk værdi ) THEN

ELSE

ELSEIF ( udtryk med logisk værdi ) THEN

ENDIF
```

Disse sætningstyper kan f.eks. benyttes som følger (hvor C tænkes af type CHARACTER*5):

```
IF ( C .EQ. 'RED  ' ) THEN
   WRITE (6,*) 'Color is red'
ELSE IF ( C .EQ.'BLUE ' ) THEN
   WRITE (6,*) 'Color is blue'
ELSE
   WRITE (6,*) 'Color is not set'
END IF
```

Ind- og udlæsning i FORTRAN.

Filer har numre i stedet for navne. Standard INPUT og OUTPUT filerne (som i PASCAL) har traditionelt numrene 5 og 6.

Ind- og udlæsning er notorisk den vanskeligste del af FORTRAN, fordi der er så mange muligheder for kompakt programmering. De oftest benyttede sætninger har formen

```
READ ( filnr , format ) variabelliste
WRITE( filnr , format ) variabelliste
```

hvor variabellisterne er som i DATA-sætninger, og format enten er * (formatfrit input/output omtrent som i PASCAL) eller nummeret på en FORMAT-sætning, der gennemløbes samtidig med variabellisten og styrer ind/udlæsningen. Idet w er et tal, som angiver antallet af benyttede positioner, er de vigtigste FORMAT-koder:

Iw	heltal
Fw.d	flydende tal med d cifre efter punktum
Aw	CHARACTER*w
wH---	tekstkonstant (svarer til '---')
wX	overspringelse
/	linieskift

Koder som '3I5' svarer til 'I5,I5,I5'.

READ og WRITE sætninger efterlader altid filerne således at næste operation begynder på en ny linie. I uddatafiler fortolkes første tegn som styretegn til printeren efter følgende regler:

blank	normalt linieskift
0	overspringelse af linie
1	ny side
+	den nye linie skrives oveni den foregående.

Hvis f.eks. B er erklæret ved

```
      REAL  B(200)
```

kan indholdet af B udskrives i 20 linier med 10 tal i hver linie ved hjælp af

```
      WRITE (6,999) B
  999 FORMAT(1H ,10F8.5)
```

idet 'udløb' af et FORMAT bevirker linieskift, hvorefter FORMAT'et bruges igen indtil variabellisten er udtømt.

Underprogrammer i FORTRAN.

I et FORTRAN program kan man efter variabelerklæringerne definere simple sætningsfunktioner som

```
      F(X,Y) = SIN( 2*X+Y )
```

og bruge dem i resten af programmet. Funktionsværdien har samme type som "variablen" F.

Ellers må underprogrammer kompileres hver for sig således at de giver anledning til hvert sit selvstændige relokerbare program (TEXT-filer på IBM4341).

Der er to slags underprogrammer svarende til procedurer og funktioner i PASCAL.

Subrutiner indledes med en SUBROUTINE-sætning, som specificerer de variabelnavne, der benyttes som parametre. Alle parametre fungerer som var parametre i PASCAL. Parametrenes type specificeres som de øvrige variable i underprogrammet. Dog behøver indeksgrænser for arrays ikke at være konstante, men kan overføres som parametre (eller som * hvis værdien er underordnet).

F.eks. kan en subrutine, der beregner sporet af en NxN-matrix af dobbelt nøjagtighed flydende tal konstrueres således:

```
      SUBROUTINE TRACE(A,N,S)
      DOUBLE PRECISION A(N,N),S
      S = 0.0
      DO 1 I = 1,N
    1 S = S+A(I,I)
      RETURN
      END
```

Et hovedprogram, der indlæser en 10x10-matrix rækkevis og kalder TRACE, kan se således ud:

```
      PROGRAM TEST
      DOUBLE PRECISION B(10,10),X
      READ (5,*) (( B(I,J),J=1,10),I=1,10)
      CALL TRACE(B,10,X)
      WRITE (6,*) X
      STOP
    1 FORMAT(' Sporet er ',F10.10)
      END
```

Sporunderprogrammet kan konstrueres som funktionsunderprogram på følgende måde:

```
      PROGRAM TEST
      DOUBLE PRECISION FUNCTION TRACE(A,N)
      TRACE = 0.0
      DO 100 I = 1,N
  100 TRACE = TRACE+A(I,I)
      RETURN
      END
```

Her kan kaldet i hovedprogrammet skrives som

```
      X = TRACE(B,10)
```

Ved kald af underprogrammer er det kun begyndelsesadresserne på parametrene som overføres. Det kontrolleres (normalt) ikke om parametrene har korrekt type.

Variable tilhørende et underprogram oprettes ikke ved hjælp af en stak, når underprogrammet kaldes. Der er derfor ikke mulighed for rekursive kald i FORTRAN.

1. Purpose

H01ADF solves the linear programming problem via the Revised Simplex method.

IMPORTANT: before using this routine, read the appropriate implementation document to check the interpretation of italicised terms and other implementation-dependent details.

2. Specification (FORTRAN)

```
      SUBROUTINE H01ADF(A,MMM,M,N,INEQ,RHS,MIN,MAXIT,MN,MM,M1,M2,CO,
     1                  IN,II,NBV,D,B,ANS,OPT,NUMIT,IPAR,IFAIL)
C     INTEGER MMM,M,N,INEQ(M),MAXIT,MN,MM,M1,M2,CO(M),II,NBV(MN),D(MM),
C    1        B(M),NUMIT,IPAR,IFAIL
C     real A(MMM,N),RHS(M),MIN,IN(II,M1),ANS(M2),OPT
```

3. Description

H01ADF solves the linear programming problem:-

$$\text{minimise} \quad Z = \sum_{j=1}^{n} c_j x_j$$

subject to the constraints

$$\sum_{j=1}^{n} a_{ij} x_j \; R_i \; b_i$$

and $x_j \geq 0$

where R_i is either "≥", "=" or "≤"

$i = 1,2,\ldots,m$ and $j = 1,2,\ldots,n$

via the Revised Simplex method.

The routine automatically sets up all slack and any artificial variables required to solve the problem. Slack variables are given integer identifiers in the range n+1 to n+m inclusive and artificial variables are given integer identifiers in the range n+m+1 to n+m+m as required. An initial basis is established comprised of slack and/or artifical variables.
Degeneracy is resolved using the perturbed problem technique described in [1], Chapter 14. The Revised Simplex technique used is outlined in [1], Chapter 13.

H01ADF

4. References

[1] GARVIN, W.W.
Introduction to Linear Programming.
McGraw Hill, New York, 1960.

[2] MULLER-MERBACH, H.
On Round-off Errors in Linear Programming.
Springer-Verlag, Berlin, 1970.

5. Parameters

A - *real* array of DIMENSION (MMM,p) where p ≥ N.
Before entry, A(I,J) (I = 1,2,...,M; J = 1,2,...,N) must be set to the coefficients of the constraints a_{ij} (i = 1,2,...,m; j = 1,2,...,n) and A(M1,J) (J = 1,2,...,N) must be set to the coefficients of the objective function c_j (j = 1,2,...,n).
Unchanged on exit provided all RHS(I) (I = 1,2,...,M) were >0 on entry to the subroutine (see Section 11).

MMM - INTEGER.
On entry, MMM specifies the first dimension of the array A as declared in the calling (sub)program, MMM ≥ M1.
Unchanged on exit.

M - INTEGER.
On entry, M specifies the number of constraints, m.
Unchanged on exit.

N - INTEGER.
On entry, N specifies the number of variables, n.
Unchanged on exit.

INEQ - INTEGER array of DIMENSION at least (M).
Before entry, INEQ(I) (I = 1,2,...,M) must be set to indicate the type of constraint i(i = 1,2,...,m) as follows:-

INEQ(I) = -1 a "≤" constraint
INEQ(I) = 0 an "=" constraint
INEQ(I) = 1 a "≥" constraint

Unchanged on exit provided all RHS(I) (I = 1,2,...,M) were >0 on entry to the subroutine (see Section 11).

RHS - *real* array of DIMENSION at least (M).
Before entry, RHS(I) (I = 1,2,...,M) must be set to b_i (i = 1,2,...,m) respectively. Unchanged on exit provided that all its elements were positive on entry to the subroutine (see Section 11).

5. **Parameters** (contd)

MIN - *real*.
On entry, MIN specifies the tolerance factor which is used to test quantities which are affected by rounding errors (see Section 10). Unchanged on exit.

MAXIT - INTEGER.
On entry, MAXIT specifies the maximum number of iterations that the subroutine is allowed to perform. For most problems MAXIT set to 2 × M is sufficient. Unchanged on exit.

MN - INTEGER.
On entry, MN specifies M + N. Unchanged on exit.

MM - INTEGER.
On entry, MM specifies 2M. Unchanged on exit.

M1 - INTEGER.
On entry, M1 specifies M + 1. Unchanged on exit.

M2 - INTEGER.
On entry, M2 specifies M + 2. Unchanged on exit.

CO - INTEGER array of DIMENSION at least (M) used as workspace.
Undefined on exit.

IN - *real* array of DIMENSION (II,q) where q ≥ M1, used as workspace.
On exit, IN(I,J) (I = 1,2,...,M;J = 1,2,...,N) contain the inverse of the final basis and IN(M1,J) (J = 1,2,...,M) contain the last generated set of simplex multipliers, the remainder of the array being undefined.

II - INTEGER.
On entry, II specifies the first dimension of the array IN as declared in the calling (sub)program, II ≥ M2.
Unchanged on exit.

NBV - INTEGER array of DIMENSION at least (MN), used as workspace.
Undefined on exit.

D - INTEGER array of DIMENSION at least (MM), used as workspace.
Undefined on exit.

H01ADF

5. Parameters (contd)

B - INTEGER array of DIMENSION at least (M).
On exit, if IPAR = 0 and IFAIL = 0, the first M elements of B contain the subscripts of the variables that are in the final basis. For other exit values of IPAR and IFAIL, the contents of B are undefined.

ANS - *real* array of DIMENSION at least (M2), used as workspace.
On exit, it contains the values of the variables that are in the last generated basis, viz:-

$$X_{B(I)} = ANS(I) \quad (I = 1,2,\ldots,M)$$

ANS(M1) and ANS(M2) are undefined on exit.

OPT - *real*.
On exit, if IPAR = 0 and IFAIL = 0, OPT contains the minimum. For other combinations of values of IPAR and IFAIL, OPT is undefined on exit.

NUMIT - INTEGER.
On exit, NUMIT contains the number of iterations carried out by the subroutine.

IPAR - INTEGER.
On exit, if IFAIL = 0, IPAR is set as follows:-

IPAR = 0	an optimal solution has been found
IPAR = 1	there is no feasible solution to the problem
IPAR = 2	the constraint region for the problem is unbounded.

For other values of IFAIL, IPAR is undefined on exit.

IFAIL - INTEGER.
Before entry, IFAIL must be assigned a value. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0. Unless the routine detects an error (see Section 6), IFAIL contains 0 on exit.

6. Error Indicators

Errors detected by the routine:-

IFAIL = 1	On entry to the routine the array INEQ has one, or more, elements incorrectly assigned.
IFAIL = 2	On entry to the routine MAXIT < 1.

6. Error Indicators (contd)

IFAIL = 3 During computation MAXIT has been exceeded.

IFAIL = 4 On entry to the routine one, or more, of the following conditions has been detected:-
M < 1,N < 1,M2 ≠ M+2,M1 ≠ M+1,M2 > II,M1 > MMM,MM ≠ 2×M.

7. Auxiliary Routines

This routine calls NAG Library routines P01AAF and X02ACF.

8. Timing

The computation time is dependent upon the constraint region.

9. Storage

There are no internally declared arrays.

10. Accuracy

An algorithm is incorporated in the subroutine which controls the propagation of rounding errors between iterations. The algorithm is dependent upon the value of MIN. For most problems, MIN = 10^3 × EPS is sufficient, where EPS is the smallest, positive number such that 1.0 + EPS > 1.0 , on the computer. (See NAG Library routine X02AAF.) It may prove necessary to run a program using the routine a number of times, with different values of MIN, to determine the effects of round-off errors on the final solution. The algorithm as applied when computing new elements of the inverse matrix is outlined in [2].

11. Further Comments

A condition of use of the Revised Simplex algorithm is that on entry the b_i (i = 1,2,...,m) are ≥ 0. If on entry to the routine RHS(K) (1 ≤ K ≤ M) is found to be negative then RHS(K),INEQ(K) and A(K,I) (I = 1,2,...,N) all have their signs reversed. On exit from the subroutine the elements are not returned to their original state.

12. Keywords

Linear Programming.
Revised Simplex Method.

H01ADF

13. Example

Solve the problem:-

$$\text{minimise:-} \quad -0.75x_1 + 20x_2 - 0.5x_3 + 6x_4$$

$$\text{subject to:-} \quad 0.5x_1 - 12x_2 - 0.5x_3 + 3x_4 \leq 0$$

$$0.25x_1 - 8x_2 - x_3 + 9x_4 \leq 0$$

$$x_3 \leq 1$$

$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0$$

We assume that the maximum number of iterations (MAX) is less than 200 and MIN = 10^{-5}.

Program

This single precision example program may require amendment
i) for use in a DOUBLE PRECISION implementation
ii) for use in either precision in certain implementations.
The results produced may differ slightly.

```
C     H01ADF EXAMPLE PROGRAM TEXT
C     NAG COPYRIGHT 1975
C     MARK 4.5 REVISED
      REAL IN(5,4), MIN, OPT, X(4,4), RHS(3), ANS(5), TITLE(7)
      INTEGER B(3), C(3), D(6), NIN, NOUT, M, N, M1, M2, MM, MAX,
     * I, J, IFAIL, NUMIT, IPAR, K, MN, NBV(7), INEQ(3)
      DATA NIN /5/, NOUT /6/
      READ (NIN,99999) TITLE
      WRITE (NOUT,99998) (TITLE(I),I=1,6)
      M = 3
      N = 4
      M1 = 4
      M2 = 5
      MM = 6
      MN = 7
      MAX = 200
      MIN = 0.00001
      READ (NIN,99997) ((X(I,J),J=1,4),INEQ(I),RHS(I),I=1,3),
     * (X(4,J),J=1,4)
```

13. Example

Program (contd)

```
      IFAIL = 1
      CALL H01ADF(X, 4, M, N, INEQ, RHS, MIN, MAX, MN, MM, M1, M2,
     * CO, IN, 5, NBV, D, B, ANS, OPT, NUMIT, IPAR, IFAIL)
      IF (IFAIL.EQ.0) GO TO 20
      WRITE (NOUT,99991) IFAIL
      STOP
   20 K = IPAR + 1
      GO TO (80, 40, 60), K
   40 WRITE (NOUT,99996)
      STOP
   60 WRITE (NOUT,99995)
      STOP
   80 WRITE (NOUT,99994) NUMIT
      WRITE (NOUT,99993) (B(I),ANS(I),I=1,M)
      WRITE (NOUT,99992) OPT
      STOP
99999 FORMAT (6A4, 1A3)
99998 FORMAT (4(1X/), 1H , 5A4, 1A3, 7HRESULTS/1X)
99997 FORMAT (4F8.2, I3, F8.2)
99996 FORMAT (1HO/21H NO FEASIBLE SOLUTION)
99995 FORMAT (1HO/19H UNBOUNDED SOLUTION)
99994 FORMAT (1HO/24H NUMBER OF ITERATIONS = , I3)
99993 FORMAT (1HO, (/2H X, I3, 3H = , F15.4))
99992 FORMAT (1HO, 14H OPTIMUM VALUE, F17.4)
99991 FORMAT (1HO/16H SOFT FAIL ERROR, I3)
      END
```

Data

```
HO1ADF EXAMPLE PROGRAM DATA
    0.50  -12.00   -0.50    3.00 -1    0.00
    0.25   -8.00   -1.00    9.00 -1    0.00
    0.00    0.00    1.00    0.00 -1    1.00
   -0.75   20.00   -0.50    6.00
```

H01ADF

13. Example (contd)

Results

```
H01ADF EXAMPLE PROGRAM RESULTS


NUMBER OF ITERATIONS =   5

X  3 =          1.0000

X  1 =          1.0000

X  6 =          0.7500

OPTIMUM VALUE          -1.2500
```

Københavns Universitet
Matematisk Institut

November 1985.

DATALOGI A for matematikere

LIDT OM MATLAB.

Når EDB skal anvendes ved løsning af mere komplicerede opgaver er det bekvemt at kunne råde over et passende "bibliotek" af hjælpeprogrammer. Et stort antal rutiner beregnede til løsning af lineære problemer er samlet i programbibliotekerne LINPACK og EISPACK. Systemet MATLAB giver brugeren adgang til at udnytte visse af disse interaktivt. MATLAB er skrevet i FORTRAN.

MATLAB PÅ IBM 4341.

På IBM 4341 aktiveres systemet med kommandoen

```
MATLAB
```

Herefter kan der på skærmen føres en dialog med systemet i form af at der indtastes kommandoer (efterfulgt af tryk på ENTER). Når MATLAB har reageret, kan der indtastes nye kommandoer. Systemet forlades med kommandoen

```
RETURN
```

(eller EXIT eller ved at trykke PF3).
Kommandoer til systemet kan være definitionskommandoer, beregningskommandoer eller systemkommandoer. Blandt de sidste kan nævnes kommandoen

```
HELP
```

der bevirker, at der på skærmen vises en menu over de parametre, der kan anvendes i kommandoen

```
HELP parameter
```

Der henvises iøvrigt til MATLAB-manualen (fra RECKU), som findes i terminalrummet, samt den efterfølgende listning af HELP systemets forklaringer:

The HELP document

INTRO Welcome to MATLAB.

Here are a few sample statements:

```
A = <1 2; 3 4>
b = <5 6>'
x = A\b
<V,D> = eig(A),  norm(A-V*D/V)
help \ , help eig
```

HELP HELP gives assistance.
HELP HELP obviously prints this message.
To see all the HELP messages, list the file
HELP MATLAB

< < > Brackets used in forming vectors and matrices.
<6.9 9.64 SQRT(-1)> is a vector with three elements separated by blanks. <6.9, 9.64, sqrt(-1)> is the same thing. <1+I 2-I 3> and <1 +I 2 -I 3> are not the same. The first has three elements, the second has five.
<11 12 13; 21 22 23> is a 2 by 3 matrix . The semicolon ends the first row.

Vectors and matrices can be used inside < > brackets.
<A B; C> is allowed if the number of rows of A equals the number of rows of B and the number of columns of A plus the number of columns of B equals the number of columns of C . This rule generalizes in a hopefully obvious way to allow fairly complicated constructions.

A = < > stores an empty matrix in A , thereby removing it from the list of current variables.

For the use of < and > on the left of the = in multiple assignment statements, see LU, EIG, SVD and so on.

In WHILE and IF clauses, <> means less than or greater than, i.e. not equal, < means less than, > means greater than, <= means less than or equal, >= means greater than or equal.

For the use of > and < to delineate macros, see MACRO.

> See < . Also see MACRO.

(( ) Used to indicate precedence in arithmetic expressions in the usual way. Used to enclose arguments of functions in the usual way. Used to enclose subscripts of vectors and matrices in a manner somewhat more general than the usual way. If X and V are vectors, then X(V) is

<X(V(1)), X(V(2)), ..., X(V(N))> . The components of V are rounded to nearest integers and used as subscripts. An error occurs if any such subscript is less than 1 or greater than the dimension of X . Some examples:
X(3) is the third element of X .
X(<1 2 3>) is the first three elements of X . So is
X(<SQRT(2), SQRT(3), 4*ATAN(1)>) .
If X has N components, X(N:-1:1) reverses them.
The same indirect subscripting is used in matrices. If V has M components and W has N components, then A(V,W) is the M by N matrix formed from the elements of A whose subscripts are the elements of V and W . For example...
A(<1,5>,:) = A(<5,1>,:) interchanges rows 1 and 5 of A .

) See (.

= Used in assignment statements and to mean equality in WHILE and IF clauses.

. Decimal point. 314/100, 3.14 and .314E1 are all the same.

Element-by-element multiplicative operations are obtained using .* , ./ , or .\ . For example, C = A ./ B is the matrix with elements c(i,j) = a(i,j)/b(i,j) .

Kronecker tensor products and quotients are obtained with .*. , ./. and .\. . See KRON.

Two or more points at the end of the line indicate continuation. The total line length limit is 1024 characters.

, Used to separate matrix subscripts and function arguments. Used at the end of FOR, WHILE and IF clauses. Used to separate statements in multi-statement lines. In this situation, it may be replaced by semicolon to suppress printing.

; Used inside brackets to end rows.
Used after an expression or statement to suppress printing. See SEMI.

\ Backslash or matrix left division. A\B is roughly the same as INV(A)*B , except it is computed in a different way. If A is an N by N matrix and B is a column vector with N components, or a matrix with several such columns, then X = A\B is the solution to the equation A*X = B computed by Gaussian elimination. A warning message is printed if A is badly scaled or nearly singular.
A\EYE produces the inverse of A .

If A is an M by N matrix with M < or > N and B is a column vector with M components, or a matrix with several such columns, then X = A\B is the solution in the least squares sense to the under- or overdetermined system of equations A*X = B . The effective rank, K, of A is determined from the QR decomposition with pivoting. A

solution X is computed which has at most K nonzero components per column. If K < N this will usually not be the same solution as PINV(A)*B .
A\EYE produces a generalized inverse of A .

If A and B have the same dimensions, then A .\ B has elements a(i,j)\b(i,j) .

/ Slash or matrix right division. B/A is roughly the same as B*INV(A) . More precisely, B/A = (A'\B')' . See \ .

IF A and B have the same dimensions, then A ./ B has elements a(i,j)/b(i,j) .

Two or more slashes together on a line indicate a logical end of line. Any following text is ignored.

' Transpose. X' is the complex conjugate transpose of X .
Quote. 'ANY TEXT' is a vector whose components are the MATLAB internal codes for the characters. A quote within the text is indicated by two quotes. See DISP and FILE .

+ Addition. X + Y . X and Y must have the same dimensions.

- Subtraction. X - Y . X and Y must have the same dimensions.

* Matrix multiplication, X*Y . Any scalar (1 by 1 matrix) may multiply anything. Otherwise, the number of columns of X must equal the number of rows of Y .

Element-by-element multiplication is obtained with X .* Y .

The Kronecker tensor product is denoted by X .*. Y .

Powers. X**p is X to the p power. p must be a scalar. If X is a matrix, see FUN .

: Colon. Used in subscripts, FOR iterations and possibly elsewhere.
J:K is the same as <J, J+1, ..., K>
J:K is empty if J > K .
J:I:K is the same as <J, J+I, J+2I, ..., K>
J:I:K is empty if I > 0 and J > K or if I < 0 and J < K .
The colon notation can be used to pick out selected rows, columns and elements of vectors and matrices.
A(:) is all the elements of A, regarded as a single column.
A(:,J) is the J-th column of A
A(J:K) is A(J),A(J+1),...,A(K)
A(:,J:K) is A(:,J),A(:,J+1),...,A(:,K) and so on.
For the use of the colon in the FOR statement, See FOR .

ABS ABS(X) is the absolute value, or complex modulus, of the elements of X .

ANS Variable created automatically when expressions are not assigned to anything else.

ATAN ATAN(X) is the arctangent of X . See FUN .

BASE BASE(X,B) is a vector containing the base B representation of X . This is often used in conjunction with DISPLAY. DISPLAY(X,B) is the same as DISPLAY(BASE(X,B)). For example, DISP(4*ATAN(1),16) prints the hexadecimal representation of pi.

BACK BACK(B,U,P) is a vector (matrix) of the same dimension as B. P is an optional permutation vector, and U permuted with P is assumed to be upper triangular with non-zero elements on the diagonal. If P is omitted no permutation is assumed. The vector returned is the solution X to the equation U*X = B. See FORW.

CHAR CHAR(K) requests an input line containing a single character to replace MATLAB character number K in the following table. For example, CHAR(45) replaces backslash. CHAR(-K) replaces the alternate character number K.

K	character	alternate	name
0 - 9	0 - 9	0 - 9	digits
10 - 35	A - Z	a - z	letters
36			blank
37	(	(	lparen
38	)	)	rparen
39	;	;	semi
40	:	ø	colon
41	+	+	plus
42	-	-	minus
43	*	*	star
44	/	/	slash
45	\	Å	backslash
46	=	=	equal
47	.	.	dot
48	,	,	comma
49	'	"	quote
50	<	#	less
51	>	$	great

CHOL Cholesky factorization. CHOL(X) uses only the diagonal and upper triangle of X . The lower triangular is assumed to be the (complex conjugate) transpose of the upper. If X is positive definite, then R = CHOL(X) produces an upper triangular R so that R'*R = X . If X is not positive definite, an error message is printed.

CHOP Truncate arithmetic. CHOP(P) causes P places to be chopped off after each arithmetic operation in subsequent computations. This means P hexadecimal digits on some computers and P octal digits on others. CHOP(0) restores full precision.

CLEAR Erases all variables, except EPS, FLOP, EYE and RAND. X = <> erases only variable X . So does CLEAR X .

COND Condition number in 2-norm. COND(X) is the ratio of the

largest singular value of X to the smallest.

CONJG CONJG(X) is the complex conjugate of X .

COS COS(X) is the cosine of X . See FUN .

DET DET(X) is the determinant of the square matrix X .

DIAG If V is a row or column vector with N components, DIAG(V,K) is a square matrix of order N+ABS(K) with the elements of V on the K-th diagonal. K = 0 is the main diagonal, K > 0 is above the main diagonal and K < 0 is below the main diagonal. DIAG(V) simply puts V on the main diagonal.
eg. DIAG(-M:M) + DIAG(ONES(2*M,1),1) + DIAG(ONES(2*M,1),-1)
produces a tridiagonal matrix of order 2*M+1 .
IF X is a matrix, DIAG(X,K) is a column vector formed from the elements of the K-th diagonal of X .
DIAG(X) is the main diagonal of X .
DIAG(DIAG(X)) is a diagonal matrix .

DIARY DIARY('file') causes a copy of all subsequent terminal input and most of the resulting output to be written on the file. DIARY(0) turns it off. See FILE.

DISP DISPLAY(X) prints X in a compact format. If all the elements of X are integers between 0 and 51, then X is interpreted as MATLAB text and printed accordingly. Otherwise, + , - and blank are printed for positive, negative and zero elements. Imaginary parts are ignored.
DISP(X,B) is the same as DISP(BASE(X,B)).

EIG Eigenvalues and eigenvectors.
EIG(X) is a vector containing the eigenvalues of a square matrix X .
<V,D> = EIG(X) produces a diagonal matrix D of eigenvalues and a full matrix V whose columns are the corresponding eigenvectors so that X*V = V*D .

ELSE Used with IF .

END Terminates the scope of FOR, WHILE and IF statements. Without END's, FOR and WHILE repeat all statements up to the end of the line. Each END is paired with the closest previous unpaired FOR or WHILE and serves to terminate its scope. The line
FOR I=1:N, FOR J=1:N, A(I,J)=1/(I+J-1); A
would cause A to be printed N**2 times, once for each new element. On the other hand, the line
FOR I=1:N, FOR J=1:N, A(I,J)=1/(I+J-1); END, END, A
will lead to only the final printing of A .
Similar considerations apply to WHILE.
EXIT terminates execution of loops or of MATLAB itself.

EPS Floating point relative accuracy. A permanent variable whose value is initially the distance from 1.0 to the next largest floating point number. The value is changed by CHOP, and other values may be assigned. EPS is used as a

default tolerance by PINV and RANK.

EXEC EXEC('file',k,n) obtains subsequent MATLAB input from an external file. The printing of input is controlled by the optional parameter k . The optional parameter n is used to start input at line number n of the file . n = 1 is default.
If k = 0 , there is no echo, prompt or pause. This is the default if the exec command is followed by a semicolon.
If k = 1 , the input is echoed.
If k = 2 , the MATLAB prompt <> is printed.
If k = 3 , there will be echos and prompts, but no pauses. This is the the default if the exec command is not followed by a semicolon.
If k = 4 , MATLAB pauses before each prompt and waits for a null line to continue.
If k = 7 , there will be echos, prompts and pauses. This is useful for demonstrations on video terminals.
EXEC(0) causes subsequent input to be obtained from the terminal. An end-of-file has the same effect.
EXEC's may be nested, i.e. the text in the file may contain EXEC of another file. EXEC's may also be driven by FOR and WHILE loops.

EXIT Causes termination of a FOR or WHILE loop.
If not in a loop, terminates execution of MATLAB.

EXP EXP(X) is the exponential of X , e to the X . See FUN .

EYE Identity matrix. EYE(N) is the N by N identity matrix. EYE(M,N) is an M by N matrix with 1's on the diagonal and zeros elsewhere. EYE(A) is the same size as A . EYE with no arguments is an identity matrix of whatever order is appropriate in the context. For example, A + 3*EYE adds 3 to each diagonal element of A .

FILE The EXEC, SAVE, LOAD, PRINT and DIARY functions access files. The 'file' parameter takes different forms for different operating systems. On most systems, 'file' may be a string of up to 32 characters in quotes. For example, SAVE('AA') or EXEC('demoexec matlab') . The string will be used as the name of a file in the local operating system.
On all systems, 'file' may be a positive integer k less than 100 which will be used as a FORTRAN logical unit number. Some systems then automatically access a file with a name like FORT.k or FORk.DAT. Other systems require a file with a name like FT0kF001 to be assigned to unit k before MATLAB is executed. Check your local installation for details.
At CMS a file named WORKnn MATLAB A1 is generated for unit numbers not allocated to something else.

FLOPS Count of floating point operations.
FLOPS is a permanently defined row vector with two elements. FLOPS(1) is the number of floating point operations counted during the previous statement. FLOPS(2) is a cumulative total. FLOPS can be used in the same way

as any other vector. FLOPS(2) = 0 resets the cumulative total. In addition, FLOPS(1) will be printed whenever a statement is terminated by an extra comma. For example,
X = INV(A);,
or
COND(A), (as the last statement on the line).
HELP FLPS gives more details.

FLPS More detail on FLOPS.
It is not feasible to count absolutely all floating point operations, but most of the important ones are counted. Each multiply and add in a real vector operation such as a dot product or a 'saxpy' counts one flop. Each multiply and add in a complex vector operation counts two flops. Other additions, subtractions and multiplications count one flop each if the result is real and two flops if it is not. Real divisions count one and complex divisions count two. Elementary functions count one if real and two if complex. Some examples. If A and B are real N by N matrices, then
A + B counts N**2 flops,
A*B counts N**3 flops,
A**100 counts 99*N**3 flops,
LU(A) counts roughly (1/3)*N**3 flops.

FOR Repeat statements a specific number of times.
FOR variable = expr, statement, ..., statement, END
The END at the end of a line may be omitted. The comma before the END may also be omitted. The columns of the expression are stored one at a time in the variable and then the following statements, up to the END, are executed. The expression is often of the form X:Y, in which case its columns are simply scalars. Some examples (assume N has already been assigned a value).
FOR I = 1:N, FOR J = 1:N, A(I,J) = 1/(I+J-1);
FOR J = 2:N-1, A(J,J) = J; END; A
FOR S = 1.0: -0.1: 0.0, ... steps S with increments of -0.1 .
FOR E = EYE(N), ... sets E to the unit N-vectors.
FOR V = A, ... has the same effect as
FOR J = 1:N, V = A(:,J); ... except J is also set here.

FORW FORW(B,L,P) performes a forward substitution of the permuted lower triangular matrix L . Together with LU and BACK it is possible to solve linear equations A*X = B cheaper than using the \ operator, because there is no check on the matrix being singular or not. X = A\B is roughly the same as <L,U,P>=LU(A);X=BACK(FORW(B,L,P),U) .
FORW(B,L,<1:N>) is the same as FORW(B,L) if L is an N by N matrix.

FUN For matrix arguments X , the functions SIN, COS, ATAN, SQRT, LOG, EXP and X**p are computed using eigenvalues D and eigenvectors V . If <V,D> = EIG(X) then f(X) = V*f(D)/V . This method may give inaccurate results if V is badly conditioned. Some idea of the accuracy can be obtained by comparing X**1 with X .
For vector arguments, the function is applied to each component.

GET Get a variable from a file, where it is stored by PUT in a previous version of MATLAB.

HESS Hessenberg form. The Hessenberg form of a matrix is zero below the first subdiagonal. If the matrix is symmetric or Hermitian, the form is tridiagonal. <P,H> = HESS(A) produces a unitary matrix P and a Hessenberg matrix H so that A = P*H*P'. By itself, HESS(A) returns H.

HILB Inverse Hilbert matrix. HILB(N) is the inverse of the N by N matrix with elements 1/(i+j-1), which is a famous example of a badly conditioned matrix. The result is exact for N less than about 15, depending upon the computer.

IF Conditionally execute statements. Simple form...
IF expression rop expression, statements
where rop is =, <, >, <=, >=, or <> (not equal) . The statements are executed once if the indicated comparison between the real parts of the first components of the two expressions is true, otherwise the statements are skipped.
Example.
```
IF ABS(I-J) = 1, A(I,J) = -1;
```
More complicated forms use END in the same way it is used with FOR and WHILE and use ELSE as an abbreviation for END, IF expression not rop expression . Example
```
FOR I = 1:N, FOR J = 1:N, ...
   IF I = J, A(I,J) = 2; ELSE IF ABS(I-J) = 1, A(I,J) = -1; ...
   ELSE A(I,J) = 0;
```
An easier way to accomplish the same thing is
```
A = 2*EYE(N);
FOR I = 1:N-1, A(I,I+1) = -1; A(I+1,I) = -1;
```

IMAG IMAG(X) is the imaginary part of X .

INV INV(X) is the inverse of the square matrix X . A warning message is printed if X is badly scaled or nearly singular.

KRON KRON(X,Y) is the Kronecker tensor product of X and Y . It is also denoted by X .*. Y . The result is a large matrix formed by taking all possible products between the elements of X and those of Y . For example, if X is 2 by 3, then X .*. Y is

```
< x(1,1)*Y  x(1,2)*Y  x(1,3)*Y
  x(2,1)*Y  x(2,2)*Y  x(2,3)*Y >
```

The five-point discrete Laplacian for an n-by-n grid can be generated by

```
T = diag(ones(n-1,1),1);  T = T + T';  I = EYE(T);
A = T.*.I + I.*.T - 4*EYE;
```

Just in case they might be useful, MATLAB includes constructions called Kronecker tensor quotients, denoted by X ./. Y and X .\. Y . They are obtained by replacing the elementwise multiplications in X .*. Y with divisions.

LINES An internal count is kept of the number of lines of output since the last input. Whenever this count approaches a limit, the user is asked whether or not to suppress printing until the next input. Initially the limit is 25. LINES(N) resets the limit to N .

LOAD LOAD('file') retrieves all the variables from the file . See FILE and SAVE for more details. To prepare your own file for LOADing, change the READs to WRITEs in the code given under SAVE . Files SAVEed from an old version of MATLAB must be LOADed by the function OLDLOAD.

LOG LOG(X) is the natural logarithm of X . See FUN . Complex results are produced if X is not positive, or has nonpositive eigenvalues.

LONG Determine output format. All computations are done in complex arithmetic and double precision if it is available. SHORT and LONG merely switch between different output formats.

SHORT	Scaled fixed point format with about 5 digits.
LONG	Scaled fixed point format with about 15 digits.
SHORT E	Floating point format with about 5 digits.
LONG E	Floating point format with about 15 digits.
LONG Z	System dependent format, often hexadecimal.

LU Factors from Gaussian elimination. <L,U> = LU(X) stores a upper triangular matrix in U and a 'psychologically lower triangular matrix', i.e. a product of lower triangular and permutation matrices, in L , so that X = L*U . By itself, LU(X) returns the output from CGEFA . <L,U,P> = LU(X) also stores the permutation vector from the factorization in P. To be used in FORW when using the LU-factors .

MACRO The macro facility involves text and inward pointing angle brackets. If STRING is the source text for any MATLAB expression or statement, then

```
t = 'STRING';
```

encodes the text as a vector of integers and stores that vector in t . DISP(t) will print the text and

```
>t<
```

causes the text to be interpreted, either as a statement or as a factor in an expression. For example

```
t = '1/(i+j-1)';
disp(t)
for i = 1:n, for j = 1:n, a(i,j) = >t<;
```

generates the Hilbert matrix of order n.
Another example showing indexed text,

```
S = <'x = 3          '
     'y = 4          '
     'z = sqrt(x*x+y*y)'>
for k = 1:3, >S(k,:)<
```

It is necessary that the strings making up the "rows" of the "matrix" S have the same lengths.

MAGIC Magic square. MAGIC(N) is an N by N matrix constructed from the integers 1 through N**2 with equal row and column sums.

MAX Maximum value. MAX(X) gives the maximum value of X. MAX(ABS(X)) gives the maximum numerical value of X. MAX(X,'INDEX') gives the indices of the first occurence of the maximum value of X as a 1 by 2 matrix. P=MAX(X,'INDEX') then X(P(1,1),P(1,2)) is equal to MAX(X). You also get the indices by <N,M>=MAX(A). For a complex matrix the real part and the imaginary part of the matrix are treated seperately, if the maximum modulus is wanted, use MAX(ABS(A)).

ME Poor you. Don't give up that early - MATLAB is on.

MIN Gives the minimum value as MAX gives the maximum value.

NORM For matrices..
NORM(X) is the largest singular value of X .
NORM(X,1) is the 1-norm of X .
NORM(X,2) is the same as NORM(X) .
NORM(X,'INF') is the infinity norm of X .
NORM(X,'FRO') is the F-norm, i.e. SQRT(SUM(DIAG(X'*X))) .
For vectors..
NORM(V,P) = (SUM(V(I)**P))**(1/P) .
NORM(V) = NORM(V,2) .
NORM(V,'INF') = MAX(ABS(V(I))) .

OLDL The old version of LOAD. Use it only if you have stored something by the old version of MATLAB.

ONES All ones. ONES(N) is an N by N matrix of ones. ONES(M,N) is an M by N matrix of ones . ONES(A) is the same size as A and all ones .

ORTH Orthogonalization. Q = ORTH(X) is a matrix with orthonormal columns, i.e. Q'*Q = EYE, which span the same space as the columns of X .

PINV Pseudoinverse. X = PINV(A) produces a matrix X of the same dimensions as A' so that A*X*A = A , X*A*X = X and AX and XA are Hermitian . The computation is based on SVD(A) and any singular values less than a tolerance are treated as zero. The default tolerance is NORM(SIZE(A),'inf')*NORM(A)*EPS. This tolerance may be overridden with X = PINV(A,tol). See RANK.

PLOT PLOT(X,Y) produces a plot of the elements of Y against those of X . PLOT(Y) is the same as PLOT(1:n,Y) where n is the number of elements in Y . PLOT(X,Y,P) or PLOT(X,Y,p1,...,pk) passes the optional parameter vector P or scalars p1 through pk to the plot routine. The default plot routine is a crude printer-plot. It is hoped that an interface to local graphics equipment can be provided.
An interesting example is

```
t = 0:50;
PLOT( t.*cos(t), t.*sin(t) )
```

POLY Characteristic polynomial.
If A is an N by N matrix, POLY(A) is a column vector with N+1 elements which are the coefficients of the

characteristic polynomial, DET(lambda*EYE - A) .
If V is a vector, POLY(V) is a vector whose elements are the coefficients of the polynomial whose roots are the elements of V . For vectors, ROOTS and POLY are inverse functions of each other, up to ordering, scaling, and roundoff error.
ROOTS(POLY(1:20)) generates Wilkinson's famous example.

PRINT PRINT('file',X) prints X on the file using the current format determined by SHORT, LONG Z, etc. See FILE.

PROD PROD(X) is the product of all the elements of X .

QR Orthogonal-triangular decomposition.
<Q,R> = QR(X) produces an upper triangular matrix R of the same dimension as X and a unitary matrix Q so that X = Q*R .
<Q,R,E> = QR(X) produces a permutation matrix E , an upper triangular R with decreasing diagonal elements and a unitary Q so that X*E = Q*R .
By itself, QR(X) returns the output of CQRDC . TRIU(QR(X)) is R .

RAND Random numbers and matrices. RAND(N) is an N by N matrix with random entries. RAND(M,N) is an M by N matrix with random entries. RAND(A) is the same size as A . RAND with no arguments is a scalar whose value changes each time it is referenced.
Ordinarily, random numbers are uniformly distributed in the interval (0.0,1.0) . RAND('NORMAL') switches to a normal distribution with mean 0.0 and variance 1.0 . RAND('UNIFORM') switches back to the uniform distribution. RAND('SEED') returns the current value of the seed for the generator. RAND('SEED',n) sets the seed to n . RAND('SEED',0) resets the seed to 0, its value when MATLAB is first entered.

RANK Rank. K = RANK(X) is the number of singular values of X that are larger than NORM(SIZE(X),'inf')*NORM(X)*EPS.
K = RANK(X,tol) is the number of singular values of X that are larger than tol .

RCOND RCOND(X) is an estimate for the reciprocal of the condition of X in the 1-norm obtained by the LINPACK condition estimator. If X is well conditioned, RCOND(X) is near 1.0 . If X is badly conditioned, RCOND(X) is near 0.0 .
<R, Z> = RCOND(A) sets R to RCOND(A) and also produces a vector Z so that

NORM(A*Z,1) = R*NORM(A,1)*NORM(Z,1)

So, if RCOND(A) is small, then Z is an approximate null vector.

RAT An experimental function which attempts to remove the roundoff error from results that should be "simple" rational numbers.
RAT(X) approximates each element of X by a continued fraction of the form

$$a/b = d1 + 1/(d2 + 1/(d3 + ... + 1/dk))$$

with k <= len, integer di and abs(di) <= max . The default values of the parameters are len = 5 and max = 100. RAT(len,max) changes the default values. Increasing either len or max increases the number of possible fractions. <A,B> = RAT(X) produces integer matrices A and B so that

```
A ./ B  =  RAT(X)
```

Some examples:

```
long
T = hilb(6), X = inv(T)
<A,B> = rat(X)
H = A ./ B, S = inv(H)

short e
d = 1:8,  e = ones(d),  A = abs(d'*e - e'*d)
X = inv(A)
rat(X)
display(ans)
```

REAL REAL(X) is the real part of X .

RETURN From the terminal, causes return to the operating system or other program which invoked MATLAB. From inside an EXEC, causes return to the invoking EXEC, or to the terminal.

RREF RREF(A) is the reduced row echelon form of the rectangular matrix. RREF(A,B) is the same as RREF(<A,B>) .

ROOTS Find polynomial roots. ROOTS(C) computes the roots of the polynomial whose coefficients are the elements of the vector C . If C has N+1 components, the polynomial is C(1)*X**N + ... + C(N)*X + C(N+1) . See POLY.

ROUND ROUND(X) rounds the elements of X to the nearest integers.

SAVE SAVE('file') stores all the current variables in a file. SAVE('file',X) saves only X . See FILE . The variables may be retrieved later by LOAD('file') or by your own program using the following code for each matrix. The lines involving XIMAG may be eliminated if everything is known to be real.

```
      attach lunit to 'file'
      REAL or DOUBLE PRECISION XREAL(MMAX,NMAX)
      REAL or DOUBLE PRECISION XIMAG(MMAX,NMAX)
      READ(lunit,101) ID,M,N,IMG
      DO 10 J = 1, N
         READ(lunit,102) (XREAL(I,J), I=1,M)
         IF (IMG .NE. 0) READ(lunit,102) (XIMAG(I,J),I=1,M)
   10 CONTINUE
```

The formats used are system dependent. The following are typical. See SUBROUTINE SAVLOD in your local implementation of MATLAB.

```
101 FORMAT(4A1,3I4)
102 FORMAT(4Z18)
102 FORMAT(4O20)
102 FORMAT(4D25.18)
```

SCHUR Schur decomposition. <U,T> = SCHUR(X) produces an upper triangular matrix T , with the eigenvalues of X on the diagonal, and a unitary matrix U so that X = U*T*U' and U'*U = EYE . By itself, SCHUR(X) returns T .

SHORT See LONG .

SEMI Semicolons at the end of lines will cause, rather than suppress, printing. A second SEMI restores the initial interpretation.

SIN SIN(X) is the sine of X . See FUN .

SIZE If X is an M by N matrix, then SIZE(X) is <M, N> .
Can also be used with a multiple assignment,
<M, N> = SIZE(X) .

SQRT SQRT(X) is the square root of X . See FUN . Complex results are produced if X is not positive, or has nonpositive eigenvalues.

STOP Use EXIT instead.

SUM SUM(X) is the sum of all the elements of X . SUM(DIAG(X)) is the trace of X .

SVD Singular value decomposition. <U,S,V> = SVD(X) produces a diagonal matrix S , of the same dimension as X and with nonnegative diagonal elements in decreasing order, and unitary matrices U and V so that X = U*S*V' .
By itself, SVD(X) returns a vector containing the singular values.
<U,S,V> = SVD(X,0) produces the "economy size" decomposition. If X is m by n with m > n, then only the first n columns of U are computed and S is n by n .

SYST Issue a CMS command. All CP and CMS commands may be issued by this function. SYST('CP QUERY NAMES') will display the user-id's of the users currently logged on to the system. Unfortunately not all commands can be invoked , that includes EXEC's.

TRIL Lower triangle. TRIL(X) is the lower triangular part of X. TRIL(X,K) is the elements on and below the K-th diagonal of X. K = 0 is the main diagonal, K > 0 is above the main diagonal and K < 0 is below the main diagonal.

TRIU Upper triangle. TRIU(X) is the upper triangular part of X.

TRIU(X,K) is the elements on and above the K-th diagonal of X. K = 0 is the main diagonal, K > 0 is above the main diagonal and K < 0 is below the main diagonal.

USER Allows personal Fortran subroutines to be linked into MATLAB . The subroutine should have the heading

```
SUBROUTINE USER(A,M,N,S,T)
REAL or DOUBLE PRECISION A(M,N),S,T
```

The MATLAB statement Y = USER(X,s,t) results in a call to the subroutine with a copy of the matrix X stored in the argument A , its column and row dimensions in M and N , and the scalar parameters s and t stored in S and T . If s and t are omitted, they are set to 0.0 . After the return, A is stored in Y . The dimensions M and N may be reset within the subroutine. The statement Y = USER(K) results in a call with M = 1, N = 1 and A(1,1) = FLOAT(K) . After the subroutine has been written, it must be compiled and linked to the MATLAB object code within the local operating system . Consult Numerical Institute .

WHAT Lists commands and functions currently available.

WHILE Repeat statements an indefinite number of times.
WHILE expr rop expr, statement, ..., statement, END
where rop is =, <, >, <=, >=, or <> (not equal) . The END at the end of a line may be omitted. The comma before the END may also be omitted. The commas may be replaced by semicolons to avoid printing. The statements are repeatedly executed as long as the indicated comparison between the real parts of the first components of the two expressions is true. Example (assume a matrix A is already defined).

```
E = 0*A; F = E + EYE; N = 1;
WHILE NORM(E+F-E,1) > 0, E = E + F; F = A*F/N; N = N + 1;
E
```

WHO Lists current variables.

WHY Provides succinct answers to any questions.

Københavns Universitet
Matematisk Institut

November 1985.

DATALOGI A (M)

KORT VEJLEDNING I BRUG AF SCRIPT.

Tekstbehandlingsprogrammet SCRIPT på IBM 4341 anlægget er et meget omfattende tekstbehandlingssystem med et utal af muligheder, og en udnyttelse af alle disse muligheder kræver et grundigt kendskab til SCRIPT. Det er imidlertid, for enkle tekstbehandlingsformål, tilstrækkeligt at kende visse af de mange muligheder.

SCRIPT bruges på følgende måde: Den tekst , der ønskes formatteret skrives ved hjælp af XEDIT i en fil med filtype: SCRIPT, og undervejs forsynes teksten med styreord , der ved den senere behandling i programmet SCRIPT, benyttes til formatteringen.

Den basale opgave for SCRIPT er at ombryde teksten i tilpas lange linier og placere disse linier på passende sider. Denne opgave specificeres ved hjælp af styreordene, der dels kan definere det 'globale layout', dels kan fastlægge hvordan mindre dele af teksten skal opfattes henholdsvis behandles.

Disse styreord, der for det meste skrives i begyndelsen af en linie, består af et punktum efterfulgt af en to-bogstav kode (forkortelse) for den funktion der ønskes udført, eventuelt forsynet med parametre, og derefter følger selve teksten, enten på næste linie eller på samme linie men adskilt fra styreordet (og eventuelle parametre) ved et blanktegn.

Den med styreord forsynede tekst kan fås behandlet og derefter ses i formatteret version på skærmen med kommandoen:

```
SCRIPT filnavn.
```

Tilsvarende er det muligt at få en 'printet' version (når formatteringen er som ønsket). I terminalrummet A109 benyttes en af kommandoerne:

```
SCRPRT3 filnavn

SCRPRT2 filnavn
```

hvor SCRPRT3 bevirker udskrivning på printeren med A4-papir på højkant og SCRPRT2 på den anden.

STYREORD

Der findes i SCRIPT mere end 100 forskellige styreord, men det er muligt ved hjælp af en lille del heraf, at få brugbare resultater.

En gruppe af styreord vedrører størrelsen af den formatterede side (med til disse styreord hører en talangivelse):

```
.pl n    sidelængde= n linier
.ll n    linielængde= n karakterer
.pm n    venstremargin= n karakterer
```

hvor venstremargin er antallet af blanktegn i starten af hver linie inden selve tekstlinien.

En anden gruppe styreord bestemmer placeringen af teksten indenfor den fastlagte plads f.eks.:

```
.sk n    lav n tomme linier (.sk 0 starter på næste linie)
.cp n    start ny side, hvis færre end n linier tilbage
```

medens

```
.ce      centrering af tekst
.sx      adskillelse af tekst på linie
.us      understregning
```

direkte bestemmer formatteringen af den nærmest følgende tekst. Her vil '.ce' bevirke at teksten på resten af '.ce'-linien (eller den næste, hvis styreordet står alene) anbringes centreret på en ny linie, og '.us' bevirker at teksten på '.us'-linien (eller den næste, hvis styreordet står alene) bliver understreget. Styreordet '.sx' bevirker, at den nærmest følgende tekst adskilles i tre dele, der er specificeret ved et specielt adskillelsestegn f. eks. tegnet '/', hvor første del anbringes yderst til venstre, den midterste del benyttes som en udfyldningsstreng i midten, og den tredie del anbringes til højre i den formatterede linie (se linie 2 i eksemplet nedenfor).

Styreordene:

```
.in n    ryk tekst n pladser ind fra venstre margin
.ir n    ryk tekst n pladser ind fra højre margin
```

er i kraft indtil de annulleres ved styreord der rykker margin på plads igen (f.eks. vil '.ir -5' annullere virkningen af et foregående '.ir 5').

SCRIPT opfatter hver linie i den oprindelige tekst som en enhed, men det er muligt at få SCRIPT til at behandle en enkelt linie som bestående af flere enheder. Dette gøres ved hjælp af styreordskilletegnet ';', som anbragt i en linie der starter med et styreord får SCRIPT til at dele linien i to ved tegnet ';', der altså virker som en slags eoln-tegn. Endvidere vil tegnet ';' midt i en linie, hvis det forekommer umiddelbart foran et styreord, bevirke at linien af SCRIPT 'deles' ved ';' (se eksemplet).

Endvidere vil blanktegn i første position af en linie ved formatteringen bevirke at den foregående linie afsluttes og formatteringen går videre med teksten fra blanktegnslinien som næste linie. Dette kan være nyttigt hvor det ønskes at linier skal overføres 'som de er'. Sådanne linier må imidlertid så ikke indeholde styreord 'inde' i linien.

For en nærmere forklaring af de mange styreords betydning (og eventuelle varianter) henvises til SCRIPT-manualerne.

YDERLIGERE OM SCRIPT

SCRIPT kan imidlertid, udover simpel teksformattering, på mange flere nyttige måder hjælpe ved løsning af tekstbehandlingsopgaver, som f.eks. ved automatisk frembringelse af indholdsfortegnelse og stikordsregister samt med stavekontrol af tekster skrevet på engelsk.

Endvidere kan der i SCRIPT anvendes 'makroer', hvor en hel række SCRIPT-styreord er sat sammen til 'højniveau' styreord til udførelse af komplekse opgaver. Disse makroer kan man definere selv, eller man kan benytte nogle på forhånd definerede. For dette henvises til SCRIPT / Generalized Markup Language manualerne.

EKSEMPEL

Nedenfor er som eksempel anført hvorledes en del af denne vejledning ser ud inden formatteringen ved hjælp af SCRIPT.

```
.pl 72;.ll 68;.pm10
.sx /Københavns Universitet//December 1984./
Matematisk Institut
.sk 4;.ce DATALOGI A (M)
.sk 3;KORT VEJLEDNING I BRUG AF SCRIPT.
.sk;Tekstbehandlingsprogrammet SCRIPT på IBM 4341 anlægget
er et meget omfattende tekstbehandlingssystem med et utal af
muligheder, og en udnyttelse af alle disse muligheder kræver et
grundigt kendskab til SCRIPT.
Det er imidlertid, for enkle tekstbehandlingsformål,
tilstrækkeligt at kende visse af de mange muligheder.
.sk;SCRIPT bruges på følgende måde: Den
.us tekst;, der ønskes ;.us formatteret
skrives ved hjælp af XEDIT i en fil med filtype: SCRIPT, og
undervejs forsynes teksten med ;.us styreord
, der ved den senere behandling i programmet SCRIPT, benyttes til
.us formatteringen.
.sk;Den basale opgave for SCRIPT er at ;.us ombryde
teksten i tilpas lange linier og placere disse linier på
passende sider. Denne opgave specificeres ved hjælp af styreordene,
der dels kan definere det 'globale layout', dels kan fastlægge
hvordan mindre dele af teksten skal opfattes henholdsvis
behandles.
.sk;Disse styreord, der for det meste skrives i begyndelsen
af en linie, består af et punktum efterfulgt af en to-bogstav
kode (forkortelse) for den funktion der ønskes udført, eventuelt
forsynet med parametre, og derefter følger selve teksten, enten
på næste linie eller på samme linie men adskilt fra
styreordet (og eventuelle parametre) ved et blanktegn.
.sk;.sk;Den med styreord forsynede tekst kan fås behandlet og
derefter ses i formatteret version på skærmen med kommandoen:
.sk
      SCRIPT filnavn.
.sk;Tilsvarende er det muligt at få en 'printet' version
(når formatteringen er som ønsket). I terminalrummet A109
benyttes en af kommandoerne:
```

```
.sk
       SCRPRT3 filnavn
.sk
       SCRPRT2 filnavn
.sk;hvor SCRPRT3 bevirker udskrivning på printeren med A4-papir
på højkant og SCRPRT2 på den anden.
.sk 2
STYREORD
.sk;Der findes i SCRIPT mere end 100 forskellige styreord, men
det er muligt ved hjælp af en lille del heraf, at få brugbare
resultater.
.sk;En gruppe af styreord vedrører ;.us størrelsen
af den formatterede ;.us side
(med til disse styreord hører en talangivelse):
.sk
      .pl n    sidelængde= n linier
      .ll n    linielængde= n karakterer
      .pm n    venstremargin= n karakterer
.sk;hvor venstremargin er antallet af blanktegn i starten af hver
linie inden selve tekstlinien.
.sk;En anden gruppe styreord bestemmer ;.us placeringen
af teksten indenfor den fastlagte plads f.eks.:
.sk
      .sk n    lav n tomme linier (.sk 0 starter på næste linie)
      .cp n    start ny side, hvis færre end n linier tilbage
.sk;medens
.sk;.cp 3
      .ce      centrering af tekst
      .sx      adskillelse af tekst på linie
      .us      understregning
.sk;direkte bestemmer formatteringen af den nærmest følgende tekst.
Her vil '.ce' bevirke at teksten på resten af '.ce'-linien (eller
den næste, hvis styreordet står alene) anbringes
centreret på en ny linie, og '.us' bevirker at teksten på
'.us'-linien (eller den næste, hvis styreordet står alene) bliver
.us understreget.
Styreordet '.sx' bevirker, at den nærmest
følgende tekst adskilles i tre dele, der er specificeret ved et
specielt adskillelsestegn f. eks. tegnet '/', hvor første del
anbringes yderst til venstre, den midterste del benyttes som en
udfyldningsstreng i midten, og den tredie del anbringes til højre
i den formatterede linie (se linie 2 i eksemplet nedenfor).
.sk;Styreordene:
.sk
      .in n    ryk tekst n pladser ind fra venstre margin
      .ir n    ryk tekst n pladser ind fra højre margin
.sk;er i kraft indtil de annulleres ved styreord der rykker
margin på plads igen (f.eks. vil '.ir -5' annullere virkningen
af et foregående '.ir 5').
.in 5;.ir 5;.sk
SCRIPT opfatter hver linie i den oprindelige tekst som en enhed,
men det er muligt at få SCRIPT til at behandle en enkelt linie som
bestående af flere enheder. Dette gøres ved hjælp af
styreordskilletegnet ';', som anbragt i en linie der starter med
et styreord får SCRIPT til at dele linien i to ved tegnet ';', der
altså virker som en slags eoln-tegn. Endvidere vil tegnet ';' midt
i en linie, hvis det forekommer umiddelbart foran et styreord,
bevirke at linien af SCRIPT 'deles' ved ';' (se eksemplet).
```

Københavns Universitet
Matematisk Institut

November 1985.

DATALOGI A for matematikere

REDUCE.

REDUCE er en programpakke af en type, der på engelsk kaldes et 'computer algebra system'. Systemet kan regne med formelle udtryk, altså det der populært sagt kaldes at regne med bogstaver. Blandt mulighederne i systemet kan nævnes: simplifikation af udtryk, regning med polynomier i en eller flere variable, formel differentiation og integration af funktioner af en eller flere variable (heriblandt regning med de elementære funktioner sin, log, exp o.s.v.) og regning med matricer, hvis elementer er symboler. Kommunikation med systemet foregår ved at afgive kommandoer som sætninger i et sprog, der meget minder om PASCAL.

REDUCE på IBM 4341.

På IBM 4341 er REDUCE indlagt i en interaktiv version. For Dat A studerende ligger programpakken på D-disken, og systemet kaldes med kommandoen

```
REDUCE
```

Der går så lidt tid (hvorunder man bliver nødt til at 'cleare' skærmen med PA 2 tasten) indtil systemet melder klar med VM READ i statusfeltet. Herefter kan der afgives kommandoer til systemet. Kommandoer, der ønskes indtastet over flere linier, skilles ved at trykke ENTER, og alle kommandoer afsluttes med ';' og afsendes ved at trykke ENTER.

En serie kommandoer (afsluttet med et ekstra: ;END;), som på forhånd er indlagt i en CMS-fil af filtypen RED kan afgives til systemet med kommandoen

```
IN fn;
```

hvor fn er filnavnet.

Output dirigeres normalt til skærmen. Imidlertid vil kommandoen

```
OUT ft;
```

bevirke, at der på A-disken oprettes en fil med identifikationen

FILE ft A

(hvor det indtastede navn ft altså bliver bliver filtypen), hvori efterfølgende output skrives. Filen lukkes og output dirigeres tilbage til skærmen med kommandoen

SHUT ft;

På D-disken ligger manualen i filen

REDUCE DOC D.

Denne manual står også i terminalrummet. Indledningen og det meste af appendix B, C og D er tilføjet disse noter.

Desuden ligger på D-disken en række lektioner, der kan benyttes til indlæring af systemet. Disse lektioner kaldes med en kommando

IN LESSn;

hvor n=1..7. Systemet REDUCE forlades med kommandoen

BYE;

(efterfulgt af tryk på ENTER).

Nedenfor følger nogle vigtige grundregler og tips:

a) Husk altid at afslutte inddata med ';', da der ellers forventes flere linier.

b) Systemet arbejder med algebraiske udtryk. Evaluering (udregning) af udtryk består i reduktion af udtrykket ved hjælp af de velkendte omformningsregler.

c) Simple variable behøver ikke at blive erklæret, da der kun er 1 datatype, nemlig algebraiske udtryk. Hvis en variabel endnu ikke (ved et assignment) er tildelt en værdi, så har den sig selv som værdi.

d) Arrays erklæres ved sætninger som

```
ARRAY A(10) , B(2,3)
```

Indices løber fra 0 til den erklærede værdi. Elementerne initieres til 0.

Indtastes f.eks.

```
ARRAY A(5);
A(0):=1;
FOR I:=1:5 DO WRITE A(I):=A(I-1)*(X+I);
```

fås uddata

```
A(1) := X + 1

          2
A(2) := X  + 3*X + 2
```

```
          3       2
A(3) := X  + 6*X  + 11*X + 6

          4       3       2
A(4) := X  + 10*X  + 35*X  + 50*X + 24

          5       4       3       2
A(5) := X  + 15*X  + 85*X  + 225*X  + 274*X + 120
```

Bemærk, at værdierne af A(I)'erne udregnes uden udskrivning, hvis WRITE udelades. Bemærk iøvrigt FOR-løkken, som svarer til PASCAL's. Hvis der skal udføres flere sætninger for hver værdi af løkkevariablen, dannes sammensatte sætninger med BEGIN...END ligesom i PASCAL.

e) Indsættelse af udtryk i stedet for variable sker ved hjælp af operatoren SUB, idet værdien af

```
SUB( v = e1 , e2 )
```

er e2 reduceret med variablen v erstattet af e1.

Lad f.eks.

```
f(x) = x**2/(1+x).
```

De fem første Taylorapproksimationer til f i x kan udregnes på følgende måde:

```
ARRAY F(5),FAK(5),TAYLOR(5);
F(0):=X**2/(1+X);
FAK(0):=1;
TAYLOR(0):=F(0);
FOR I:=1:5 DO BEGIN
                  F(I):=DF(F(I-1),X);
                  FAK(I):=FAK(I-1)*I;
                  TAYLOR(I):=TAYLOR(I-1)+1/FAK(I)*F(I)*(T-X)**I;
               END;
```

og man kan herefter beregne den femte Taylorapproksimation i x og 1 ved at taste

```
TAYLOR(5);
```

der giver svaret

```
      5      4      4       3  2      3      3       2  3       2  2
( - T  + 6*T *X + T  - 15*T *X  - 6*T *X - T  + 20*T *X  + 15*T *X

        2      2      6       5    6    6      5       4       3
  + 6*T *X + T  + T*X  + 6*T*X  - X )/(X  + 6*X  + 15*X  + 20*X  +

        2
   15*X  + 6*X + 1)
```

og

```
SUB(X=1,TAYLOR(5));
```

der giver svaret

$$G := (- T^5 + 7*T^4 - 22*T^3 + 42*T^2 + 7*T - 1)/64$$

Indledningen til REDUCE manualen.

REDUCE is a system for carrying out algebraic operations accurately, no matter how complicated the expressions become. It can manipulate polynomials in a variety of forms, both expanding and factoring them, and extracting various parts of them as required. REDUCE can also do differentiation and integration, but we shall only show trivial examples of this in this introduction. Other topics which are not considered include the use of arrays, the definition of procedures and operators, the specific routines for high energy physics calculations, the use of files to eliminate repetitious typing and for saving results, and the editing of the input text.

Also not considered in any detail in this introduction are the many options that are available for varying computational procedures, output forms, number systems used, and so on.

REDUCE is designed to be an interactive system, so that the user can input an algebraic expression and see its value before moving on to the next calculation. Not all computer systems support interactive use, of course. However, REDUCE can also be used in batch mode by inputting a sequence of calculations and getting results without any possibility of interaction during the calculations.

In this introduction, we shall limit ourselves to the interactive use of REDUCE, since this illustrates most completely the capabilities of the system. When REDUCE is called, it begins by printing a banner message like:

```
REDUCE 3.0, 15-Apr-83 ...
```

where the version number (3.0) and the system release date will change from time to time. It then prompts the user for input by:

```
1:
```

The user can now type a valid REDUCE statement, terminated by a semicolon to indicate the end of the expression, for example:

```
(x+y+z)**2;
```

This expression would normally be followed by another character (a Return on an Ascii keyboard) to "wake up" the system, which would then input the expression, evaluate it, and return the result. in a form like:

$$X^2 + 2*X*Y + 2*X*Z + Y^2 + 2*Y*Z + Z^2$$

Let us review this simple example to learn a little more about the way that REDUCE works. First, we note that REDUCE deals with variables, and constants like other computer languages, but that in evaluating the former, a variable can stand for itself. Expression evaluation normally follows the rules of high school algebra, so the only surprise in the above might be that the expression was expanded. The normal mode of REDUCE

does in fact expand all expressions where possible, collecting like terms and ordering the variables in a specific manner. However, the expansion option, ordering of variables, the format of output and so on is under the control of the user, and various declarations are available to control these.

Another characteristic of the above example is the use of lower case on input and upper case on output. In fact, input may be in either mode, but lower case is converted to upper case by the system.

Finally, the numerical prompt can be used to reference the result in a later computation.

As a further illustration of the system characteristics, the user should try:

```
for i:= 1:50 product i;
```

The result in this case is the number

```
30414093201713378043612608166064768844377641568960512000000000000
```

Since we want exact results in algebraic calculations, it is essential that integer arithmetic be performed to arbitrary precision, as in the above example. Furthermore, the FOR statement in the above is illustrative of a whole range of combining forms which REDUCE supports for the convenience of the user.

Among the many options in REDUCE is the use of other number systems, such as multiple precision floating point with any specified number of digits -- of use if roundoff in, say, the 100th digit is all that can be tolerated.

In many cases, it is necessary to use the results of one calculation in succeeding calculations. One way to do this is via an assignment for a variable, such as

```
u := (x+y+z)**2;
```

If we now use u in later calculations, the value of the right hand side of the above will be used.

The results of a given calculation are in fact saved in the variable WS (for workspace), so this can be used in the next calculation for further processing.

For example, the expression

```
df(ws,x);
```

following the previous evaluation will calculate the derivative of (x+y+z)**2 with respect to x.

Alternatively,

```
int(ws,y);
```

would calculate the integral of the same expression with respect to y.

One nice feature of REDUCE is its handling of symbolic matrices. For example,

```
matrix m(2,2);
```

declares m to be a two by two matrix, and

```
m := mat((a,b),(c,d));
```

gives it specific element values. Expressions which include m and make algebraic sense may now be evaluated, such as 1/m to give the inverse, 2*m - u*m**2 to give us another matrix and det(m) to give us the determinant of m.

REDUCE also has a wide range of substitution capabilities. The system knows about elementary functions, but does not automatically invoke many of their well-known properties. For example, products of trigonometrical functions are not converted automatically into multiple angle expressions, but if the user wants this, he can say:

```
for all x,y let cos(x)*cos(y) = (cos(x+y)+cos(x-y))/2,
               cos(x)*sin(y) = (sin(x+y)-sin(x-y))/2,
               sin(x)*sin(y) = (cos(x-y)-cos(x+y))/2;
```

An expression such as sin(a+b)*cos(a-b) would now convert into the equivalent expression

```
(SIN(2*A) + SIN(2*B))/2.
```

Another very commonly used capability of the system, and an illustration of one of the many output modes of Reduce, is the ability to output results in a Fortran compatible form. Such results can then be used later in a Fortran-based numerical calculation. This is particularly useful as a way of generating algebraic formulas to be used as the basis of extensive numerical calculations.

For example, the statements

```
on fort;
df(log(x)*(sin(x)+cos(x))/sqrt(x),x,2);
```

will result in the output

```
 ANS=(-4.*LOG(X)*COS(X)*X**2-4.*LOG(X)*COS(X)*X+3.*LOG(X)*
. COS(X)-4.*LOG(X)*SIN(X)*X**2+4.*LOG(X)*SIN(X)*X+3.*LOG(X)
. *SIN(X)+8.*COS(X)*X-8.*COS(X)-8.*SIN(X)*X-8.*SIN(X))/(4.*
. SQRT(X)*X**2)
```

8. OPERATORS NORMALLY AVAILABLE IN REDUCE

Notation: Each operator is provided with a prototypical header line given here. Each formal parameter is given a name and followed by its allowed type. The names of classes referred to in the definition are printed in lower case, and parameter names in upper case. If a parameter type is not commonly used, it may be a specific set enclosed in brackets ↓...c. Operators which accept formal parameter lists of arbitrary length have the parameter and type class enclosed in square brackets indicating that zero or more occurrences of that argument are permitted. Optional parameters and their type classes are enclosed in angle brackets.

ARGLENGTH(EXPRN:algebraic)
Returns the number of arguments of the top level operator in EXPRN (Page 7-14)

COEFF(EXPRN:polynomial,VAR:kernel,ID:identifier)
Partitions polynomial EXPRN into its coefficients with respect to VAR and stores them in ID (Page 7-12)

DEG(EXPRN:polynomial,VAR:kernel):integer
Returns the leading degree of the polynomial EXPRN in the variable VAR (Page 8-6)

DEN(EXPRN:rational):polynomial
Returns the denominator of the rational expression EXPRN (Page 8-6)

DET(EXPRN:matrix_expression):algebraic
Returns the determinant of the matrix EXPRN (Page 12-2)

DF(EXPRN:algebraic,[VAR:kernel(,NUM:integer)]):algebraic
Returns the derivative of EXPRN wrt VAR, repeated NUM times (Page 6-2)

FACTORIZE(EXPRN:polynomial[,ID:identifier[,INTEXP:prime integer]])
Factorizes polynomial EXPRN, leaving factors in ID, and using optional prime INTEXP for internal computation (Page 8-2)

GCD(EXPRN1:polynomial,EXPRN2:polynomial):polynomial
Returns the greatest common divisor of the two polynomials EXPRN1 and EXPRN2 (Page 8-4)

LCOF(EXPRN:polynomial,VAR:kernel):polynomial
Returns the leading coefficient of the polynomial EXPRN in the variable VAR (Page 8-7)

LINELENGTH(NUM:integer):integer
Sets the output line length to NUM and returns previous line length (Page 7-3)

LTERM(EXPRN:polynomial,VAR:kernel):polynomial
Returns the leading term of EXPRN with respect to VAR (Page 8-7)

MAINVAR(EXPRN:polynomial):expression
Returns the main variable of EXPRN (Page 8-7)

MAT
Used to represent matrices (Page 12-1)

NUM(EXPRN:rational):polynomial
Returns the numerator of the rational expression EXPRN (Page 8-8)

PART(EXPRN:algebraic[,INTEXP:integer])
Returns the appropriate part of EXPRN as defined by INTEXP (Page 7-13)

PFACTORIZE(EXPRN:polynomial,ID:identifier,INTEXP:prime integer)
Factorizes the univariate polynomial EXPRN modulo the prime INTEXP, storing the resultant factors in ID (Page 8-3)

REDUCT(EXPRN:polynomial,VAR:kernel):polynomial
Returns the reductum of EXPRN with respect to' VAR (Page 8-8)

REMAINDER(EXPRN1:polynomial,EXPRN2:polynomial):polynomial
Returns the remainder when EXPRN1 is divided by EXPRN2 (Page 8-5)

RESULTANT(EXPRN1:polynomial,EXPRN2:polynomial,VAR:kernel):polynomial
Returns the resultant of EXPRN1 and EXPRN2 with respect to the variable VAR (Page 8-6)

SOLVE(EXPRN:algebraic[,VAR:kernel]):integer
Solves a set of linear equations in terms of the kernel VAR (Page 6-5)

STRUCTR(EXPRN:algebraic):EXPRN
Displays the structure of EXPRN (Page 7-11)

SUB([VAR1:kernel = EXPRN1:algebraic],EXPRN:algebraic):algebraic
Replaces every occurrence of VAR1 in EXPRN by EXPRN1 (Page 6-7)

TP(EXPRN:matrix_expression):matrix
Returns the transpose of the matrix EXPRN (Page 12-3)

TRACE(EXPRN:matrix_expression):algebraic
Returns the trace of the matrix EXPRN (Page 12-3)

VARNAME(ID:identifier) Sets the expression naming variable to ID (Page 7-10)

C. COMMANDS AND DECLARATIONS

This index summarizes the commands and declarations normally available in REDUCE.

Notation:	E, E1,...,En	denote expressions
	V, V1,...,Vn	denote variables (or more generally kernels)

ALGEBRAIC E;
If E is empty, the system mode is set to algebraic. Otherwise, E is evaluated in algebraic mode and the system mode is not changed (Page 14-1)

ARRAY V1(size),...,Vn(size)
Declares V1 through Vn as array names. (size) describes the maximum size of the array (Page 5-1)

BYE;
Stops the execution of REDUCE and returns you to the system monitor. The REDUCE job is destroyed (Page 5-3)

CLEAR E1,...,En;	Removes any substitutions declared for E1 through En from system (Page 9-3)
COMMENT ⟨any⟩;	Used for including comments in text. ⟨any⟩ is any sequence of characters not including a terminator (Page 2-4)
CONT;	An interactive command which causes the system to continue the calculation from the point in the input file where the last PAUSE was encountered (Page 11-2
DEFINE E1,...,En;	Allows for the replacement of the left-hand-side of the equations E1 through En by the corresponding right-hand-side (Page 5-3)
DEPEND V1,...,Vn;	Sets up a dependency of variable V1 on kernels V2 through Vn (Page 6-10)
DISPLAY E;	Causes previous inputs to be displayed. If E is a non-negative integer, then that many expressions will be displayed (Page 11-1)
EDITDEF ⟨name⟩	Causes the uncompiled procedure ⟨name⟩ to be displayed in interactive editing mode (Page 11-2)
ED ⟨null or number⟩	Invokes an interactive string editor for previous command or command ⟨number⟩ (Page 11-2)
END;	Used to terminate a program block, end a file, or transfer control to LISP (Page 5-2)
FACTOR E1,...,En;	Declares expressions as factors in output (Page 7-4)
FOR	Command used to define a variety of program loops (Page 4-3)
FORALL V1,...,Vn ⟨command⟩	Declares variables V1 through Vn as arbitrary in the substitution rule given by ⟨command⟩ (Page 9-2)
FOR EACH	Used for defining a variety of iterations on lists in symbolic mode (Page 14-3)
GO (TO) V;	Performs an unconditional transfer to label V Can only be used in compound statements (Page 4-8)
IF	Used to define conditional statements (Page 4-2)
INFIX ID1,...,IDn;	Declares ID1 through IDn to be infix operators (Page 6-7)
INTEGER V1,...,Vn;	Declares V1 through Vn as local integer variables in a block statement (Page 4-6)
IN V1,...,Vn;	Inputs the external REDUCE files V1 through Vn (Page 10-1)
KORDER V1,...,Vn;	Declares an internal ordering for variables V1 through Vn (Page 7-12)

LET E1,...,En;	Declares substitutions for the left hand sides of expressions E1 through En. In addition, LET can be used to input differentiation rules (Page 9-1)
LISP E;	A synonym for SYMBOLIC E; (Page 14-1)
MATCH E1,..., En;	Declares substitutions for the left hand sides of E1 through En when matching of explicit powers is required (Page 9-6)
MATRIX E1,...,En;	Declares matrix variables to the system. The Ei may be matrix variable names, or include details of the size of the matrix (Page 12-1)
NODEPEND V1,...,Vn;	Removes dependency of variable V1 on V2 through Vn (Page 6-10)
NONCOM ID1,...,IDn;	Declares operators ID1 through IDn to be non-commutative under multiplication (Page 6-9)
OFF V1,...,Vn;	Turns off the switches V1 through Vn (Page 5-2)
ON V1,...,Vn;	Turns on the switches V1 through Vn (Page 5-2)
OPERATOR V1,...,Vn;	Declares identifiers V1 through Vn as algebraic operators (Page 6-7)
ORDER V1,...,Vn;	Declares an ordering for variables V1 through Vn on output (Page 7-4)
OUT V;	Declares V as an output file (Page 10-1)
PAUSE;	An interactive command for use in an input file. When it is evaluated, control is transferred to the user's terminal (Page 11-2)
PRECEDENCE ID1,ID2;	Give infix operator ID1 a precedence higher than ID2 (Page 6-7)
PRECISION E;	Sets the real number precision to E decimal digits when arbitrary precision real arithmetic (BIGFLOAT) is used (Page 8-9)
PROCEDURE	Names a statement for repeated use in calculations. Type specification of procedure precedes the command name (Page 13-1)
QUIT;	Stops the execution of REDUCE and returns you to the system monitor. The REDUCE job is retained (Page 5-3)
REAL V1,...,Vn;	Declares V1 through Vn as local real variables in a block statement (Page 4-6)
REMFAC E1,...,En;	Removes expressions as factors in output (Page 7-4)
RETURN E;	Causes a transfer out of a compound statement to the next highest program level. Value of E is returned from compound statement. E may be

	empty (Page 4-8)
SAVEAS E;	Assigns E to the current expression in the workspace (Page 7-2)
SCALAR V1,...,Vn;	Declares V1 through Vn as local scalar variables in a block statement (Page 4-6)
SETMOD E;	Sets the modular base to E (used with the switch MODULAR) (Page 8-9)
SHARE V1,...,Vn;	Permits variables V1 through Vn to be accessed in both symbolic and algebraic modes (Page 14-5)
SHOWTIME;	Prints the elapsed time since the last call of this command or the beginning of the session (Page 5-3)
SHUT V1,...,Vn;	Closes the output files V1 through Vn (Page 10-2)
WEIGHT E1,...En;	Assigns an asymptotic weight to the left hand sides of E1 through En (Page 9-6)
WRITE E1,...,En;	Causes the values of E1 through En to be written on the current output channel (Page 7-6)
WTLEVEL E;	Sets the asymptotic weight level of the system to E (Page 9-6)

D. MODE SWITCHES IN REDUCE

This section lists the switches that may appear as arguments of ON and OFF. The action of the switch when it is ON is described here, unless stated otherwise. Unless otherwise indicated the default value of the switch is OFF.

ALLBRANCH	Used by the SOLVE module to select all or only principal branches of solutions. Default ON (Page 6-6)
ALLFAC	Causes the system to factor out common products on output of expressions. Default ON (Page 7-5)
BIGFLOAT	Provides for the use of arbitrary precision real coefficients in polynomials (Page 8-9)
CREF	If ON, causes a cross-reference analysis of subsequent inputs to occur. The actual table is printed following a later OFF CREF (Page 16-3)
DEFN	Causes the system to output the LISP equivalent of REDUCE input without evaluation (Page 14-4)
DEMO	Causes the system to pause after each command in a file until a Return is typed on the terminal (Page 5
DIV	Causes the system to divide out simple factors on output, so that negative powers or rational fraction can be produced (Page 7-5)

ECHO	Causes echoing of input (Page 10-1)
EXP	Causes expressions to be expanded during their evaluation. Default ON (Page 8-1)
FACTOR	If on, causes the system to factor expressions into factors with integer coefficients (Page 8-2)
FAILHARD	If on, causes integration algorithm to terminate with an error if integral not possible in closed terms (Page 6-4)
FLOAT	Allows for the use of floating point numbers during evaluation (Page 3-2)
FORT	Declares output in a FORTRAN notation (Page 7-9)
GCD	Causes the system to cancel greatest common divisors in rational expressions (Page 8-4)
INT	Specifies an interactive mode of operation. Default is system dependent (Page 11-2)
LCM	When on, uses least common multiple of denominators when combining rational expressions (Page 8-5)
LIST	Causes output to be listed one term to each line (Page 7-5)
MCD	Causes denominators to be combined when expressions are added. Default ON (Page 8-5)
MODULAR	Provides for the use of modular integer coefficients Base used is set by SETMOD (Page 8-9)
MSG	When off, suppresses the printing of warning messages. Error messages are still printed. Default ON (Page 16-4)
NAT	Specifies 'natural' style of output. Default ON (Page 7-11)
NERO	Inhibits printing of zero assignments (Page 7-8)
NOLNR	Suppresses the use of the linear properties in the integration algorithm (Page 6-4)
OUTPUT	If OFF, suppresses printing the value of any top level expression. Default ON (Page 7-3)
PERIOD	Causes a period to be printed after each integer coefficient in FORTRAN output. Default ON (Page 7-10)
PRET	Causes input commands to be printed in REDUCE syntax in a standard format (Page 16-4)
PRI	Specifies formatted printing for output. Default ON (Page 7-3)

RAISE	Causes input lower case characters to be converted into upper case. Characters in strings and those preceded by X are excluded. Default ON (Page 2-1)
RAT	Output switch used in conjunction with FACTOR. Cause the overall denominator in an expression to be printed with each factored sub-expression (Page 7-6)
RATIONAL	Provides for the use of rational number coefficients in polynomials (Page 8-8)
RESUBS	When RESUBS is off, the system does not re-examine an expression for further substitutions after one has been made. Default ON (Page 9-5)
SOLVEINTERVAL	Used by the SOLVE module to represent inexact roots by intervals (Page 6-5)
SOLVESINGULAR	Used by the SOLVE module to solve degenerate systems by introducing arbitrary constants. Default ON (Page 6-5)
SOLVEWRITE	Used by the SOLVE module to control printing of solutions. Default ON (Page 6-6)
TIME	Causes the system to print a message after each command giving the elapsed cpu time since the last command, or since TIME was last turned OFF or the session began (Page 5-2)
TRFAC	Traces the operation of the factorization algorithm (Page 8-3)
TRINT	Traces the operation of the integration algorithm (Page 6-4)

Københavns Universitet
Matematisk Institut

december 1985

DATALOGI A for matematikere

LIDT OM EDB GRAFIK.

Det har i en årrække været muligt at fremstille tegninger ved hjælp af datamaskiner. Fremgangsmåden er i korthed, at datamaskinen 'beregner' det ønskede billede, som derefter bliver 'tegnet' ved hjælp af ekstra tegneudstyr koblet til datamaskinen. Dette ekstra udstyr kan f.eks. være en speciel grafisk terminalskærm, en plotter eller en speciel grafisk printer. De seneste år er prisen for dette ekstra udstyr faldet drastisk, og i takt hermed er interessen for at kunne fremstille tegninger ved anvendelse af EDB steget stærkt.

Formålet med denne lille skrivelse er at give en kort oversigt over grafikfaciliteterne knyttet til IBM 4341 anlægget, og ved hjælp af nogle model-programmer vise hvorledes disse faciliteter kan udnyttes.

Der er for øjeblikket i det væsentlige to forskellige metoder til fremstilling af EDB-grafik. Den ene, den såkaldte vektorgrafik (eller streggrafik), går ud på at fremstille det ønskede billede ved hjælp af en samling liniestykker. Beregningen af billedet består i dette tilfælde af at finde ud af hvilke liniestykker billedet skal sammenstykkes af, og det beregnede billede kan derefter vises på specielle grafiske skærme af den såkaldte vektortype , eller tegnes på plottere. Den anden metode, den såkaldte rastergrafik , går ud på at fremstille billeder ved hjælp af de punkter der indgår i billedet, og beregningen består i dette tilfælde af at finde netop de punkter, som billedet er opbygget af. Det beregnede billede kan derefter vises på en (speciel) terminalskærm eller blive tegnet f.eks. af en matrix printer. Se også afsnittet i Politikens EDB bog om EDB-grafik.

De fleste EDB-grafiksystemer er stærkt knyttet til en speciel maskine og det specielle tilkoblede grafiske ekstraudstyr, men der er en udvikling mod generelle grafiksystemer, som kan benyttes på mange forskellige anlæg uden en nærmere viden om det tilkoblede grafiske udstyr. Den mest vidtgående standardisering findes i: Graphical Kernel System (forkortet GKS). Det er muligt på IBM 4341 anlægget at benytte GKS, men dokumentationen er for tiden ikke tilfredsstillende, og det følgende indeholder derfor kun en omtale af IBM's grafiksystem GDDM (som på mange punkter svarer til GKS systemet (eller omvendt)).

GRAFIK PÅ IBM 4341 - GDDM.

På IBM 4341 anlægget er det (for tiden) tilkoblede grafiske ekstraudstyr af rastergrafiktype. Det omfatter terminalskærme med mulighed for at vise grafik (det drejer sig om skærme af type: 2B og 3B) samt en matrixprinter med mulighed for at skrive grafik (printer nr. 1 i terminalrummet A109). Selve beregningen af rasterbillederne forestås af programpakken GDDM -'Graphical Data Display Manager'. GDDM består af en meget omfattende samling (med omkring 300) 'grafiske rutiner' og indeholder desuden adskillige hjælpeprogrammer til udnyttelse af disse rutiner.

En stor del af de grafiske rutiner bevirker tegning af såkaldte grafiske primitiver , de simple enkeltdele som en tegning består af som f.eks. liniestykker, cirkelbuer og tekster (eller samlinger af sådanne dele). Andre rutiner benyttes til at fastlægge enkeltdelenes såkaldte attributer , som f.eks. farve og tykkelse af de tegnede 'kurver', placering, farve og skrifttype af tekster eller placering af et eventuelt koordinatsystems akser.

Rutinerne til frembringelse af grafiske primitiver falder i to grupper: dels lavniveaurutiner fra GDDM Base , der kan producere simple bestanddele som liniestykker, cirkelbuer, tekster mm., dels højniveaurutiner fra underprogrammet GDDM PGF (hvor PGF står for: Presentation Graphics Feature), hvormed der kan tegnes næsten færdige tegninger som f.eks. grafer i et koordinatsystem med skalerede akser, histogrammer eller lagkagekort.

Fælles for disse rutiner er at attributerne er tillagt fornuftige startværdier (default-værdier), og for at frembringe simple tegninger er det ikke nødvendigt at specificere/ændre disse attributer.

GDDM's grafiske rutiner kan anvendes direkte som subrutiner fra et program, der kan være skrevet i: FORTRAN, PL/1, ASSEMBLER, PASCAL eller BASIC. Derudover kan de benyttes interaktivt ved hjælp af ICU (der står for: 'Interactive Chart Utility') eller fra APL (lettest ved at bruge det særlige 'workspace': GRAPHPAK).

Selvom antallet af grafikrutiner er overvældende, er det muligt med ganske få subrutinekald at frembringe rimelige tegninger (en enkel graf i et koordinatsystem med akseskalaer kan frembringes med ca. 5 kald), og en skræddersyet tegning kan tegnes med beskedent ekstra besvær.

De grafiske rutiner er beskrevet i en række manualer, hvor:

GDDM Application Programming Guide.

rummer en gennemgang, med en række programmeringseksempler, af de grafiske rutiner, medens:

GDDM Base Programming Reference.

GDDM PGF Programming Reference.

indeholder referenceinformation, med specifikation af parametre, for hver enkelt rutine.

GDDM rutinerne benyttes lettest via ICU, der er et menustyret interaktivt program til frembringelse af 'Business Graphics'. Efter at have indtastet de data som ønsket afbildet, kan man i forskellige (stort set selvforklarende) undermenuer yderligere specificere den tegning der skal udføres. I stedet for selv at indtaste data direkte til ICU, kan man lade et program levere data til ICU og derefter blot benytte de interaktive muligheder i ICU til 'afpudsning' af tegningen. Manualen:

GDDM PGF ICU User's Guide.

indeholder en række øvelser med ICU, samt referenceoplysninger. Den hurtigste måde at få indblik i GDDM's muligheder er nok at gennemarbejde (hvilket kan gøres på et par timer) øvelserne i denne manual.

De nævnte manualer kan alle beses i terminalrummet A109.

En anden mulighed for interaktiv udnyttelse af GDDM's rutiner er fra APL, hvor der tillige kan frembringes simpel '3-dimensional' grafik, som f.eks. tegning af overflader og niveaukurver og manipulation med 3-dimensionale objekter.

Til brug ved tegning af f.eks. tekster er der installeret en række sæt af symboler bl.a. indeholdende alfabeter i forskellige skrifttyper (fonte). Disse symboler falder i to typer: dels de såkaldte imagesymbols som er af fast størrelse, men kan flyttes omkring i billedet, dels de såkaldte vectorsymbols som kan udtegnes med forskellige størrelser og flyttes omkring i billedet. Der findes to hjælpeprogrammer under GDDM som kan benyttes til at fremstille egne specielle symbolsæt: en 'editor' for imagesymbols og en 'editor' for vectorsymbols.

BEMÆRKNINGER TIL MODELPROGRAMMERNE.

De grafikrutiner, der benyttes i PASCAL programmet skal erklæres med en specifikation af de parametre der indgår i kaldet af dem samt en besked til PASCAL systemet, at de skal behandles som FORTRAN underprogrammer. Dette gøres ved at anføre underprogramhovedet med parametre, efterfulgt af ordet FORTRAN . Ofte er parametrene arrays, og de specificeres på sædvanlig måde ved hjælp af navne på typer, som f.eks.

```
type values=array(.0..24.) of real;
```

som det er gjort i programmet SINUS . Parametre der kun skal overføre værdier til grafikrutinen skal være 'const' specificerede. Værdier, der af grafikrutinen skal benyttes som koordinater for punkter, specificeres af type: SHORTREAL , der er en speciel version af reelle tal som i modsætning til sædvanlige REAL-værdier i PASCAL (repræsenteret ved 8 bytes) repræsenteres ved hjælp 4 bytes. Disse SHORTREAL's kaldes enkelt nøjagtigheds flydende tal , i modsætning til REAL's i PASCAL, som kaldes dobbelt nøjagtigheds flydende tal , og SHORTREAL's er tilstrækkeligt nøjagtige til tegnebrug.

PROGRAMMET KASSE

Dette program tegner et rektangel på skærmen ved hjælp af de fundamentale grafikrutiner fra GDDM Base. Her bevirker FSINIT og FSTERM start henholdsvis afslutning af 'grafik' og gsseg(0) åbner et 'lag' (segment) med nummer 0 i billedet. Selve billedet tegnes i et felt med abscisseområde og ordinatområde begge fra 0.0 til 100.0 (defaultværdier). GSMOVE flytter 'blyanten' til det angivne punkt (der så er det aktuelle punkt) og GSLINE tegner en linie fra det aktuelle punkt til det specificerede punkt, der herefter bliver det aktuelle punkt. Endelig sender ASREAD det hidtil tegnede billede til skærmen og venter på, at der bliver trykket på ENTER-tasten eller en PF-tast (de tre parametre til ASREAD vil indeholde oplysning om tasten), og når dette sker vil grafikprogrammet gå videre, altså i dette tilfælde slutte.

```
program kasse;
var i,j,k:integer;
procedure fsinit; FORTRAN;                          /*erklæring af de*/
procedure gsmove(const x,y:shortreal); FORTRAN; /*benyttede GDDM */
procedure gsline(const x,y:shortreal); FORTRAN; /*procedurer     */
procedure gsseg(const i:integer); FORTRAN;
procedure asread(var i,j,k:integer); FORTRAN;
procedure fsterm; FORTRAN;
begin
   fsinit;                                    /*initialisering af GDDM*/
   gsseg(0);                                    /*åbning af et segment*/
   gsmove(20,20);                              /*startposition (20,20)*/
   gsline(20,80);                                 /*linie til (20,80)*/
   gsline(80,80);                                 /*linie til (80,80)*/
   gsline(80,20);                                 /*linie til (80,20)*/
   gsline(20,20);                                 /*linie til (20,20)*/
   asread(i,j,k);                          /*billede til skærm, vente*/
   fsterm;                                       /*nedlukning af GDDM*/
end /*kasse*/.
```

PROGRAMMET SINUS

Dette program benytter grafikrutiner fra GDDM PGF til tegning af grafen for sinus funktionen i et koordinatsystem. Selve grafen med tilhørende koordinatsystem tegnes af CHPLOT, der selv ordner en række administrative ting som åbning af et segment og beregning af et rimeligt koordinatområde og akseskalaer. De fire parametre til CHPLOT angiver antal grafer i, antal punkter på hver graf j og derefter x-værdier i et array x og y-værdier i et array y. Hvis flere grafer skal tegnes på samme tegning f.eks. i=2 benyttes de anførte x-værdier to gange: til den første graf med de første j af y-værdierne og til den anden graf med de følgende j af y-værdierne. CHHEAD, CHXTTL og CHYTTL bevirker tegning af diverse tekster. Den første parameter angiver tekstens længde (i tegn) og den anden er selve teksten. Ved erklæringen af disse grafikrutiner (og analogt for CHPLOT) benyttes typeerklæringer af arrays, der er store nok til at dække de i programmet forekommende grafikrutinekald. Ønskes f.eks. tegning af en graf med 100 støttepunkter skal typen values ændres til:

```
type values=array(.0..99.) of shortreal;
```

CHPLOT tegner grafen ved successivt at tegne liniestykker, der forbinder de specificerede støttepunkter.

```
program sinus;
const pi=3.1416;
type tekst=packed array(.1..30.) of char;
     values=array(.0..24.) of shortreal;
var x,y:values;
    i:integer;
procedure fsinit; FORTRAN;      /*erklæring af GDDM og PGF rutiner*/
procedure chhead(const i:integer; const t:tekst); FORTRAN;
procedure chset(const a:alfa); FORTRAN;
procedure chxttl(const i:integer; const t:tekst); FORTRAN;
procedure chyttl(const i:integer; const t:tekst); FORTRAN;
procedure chplot(const i,j:integer; const x,y:values); FORTRAN;
procedure chterm; FORTRAN;
procedure asread(var i,j,k:integer); FORTRAN;
procedure fsterm; FORTRAN;
begin
   fsinit;
   chhead(17,'Sinus-funktionen');                    /*overskrift*/
   chset('CBOX');                                 /*ramme om billedet*/
   chxttl(7,'X-aksen');                              /*x-akse titel*/
   chyttl(7,'Y-aksen');                              /*y-akse titel*/
   for i:=0 to 24 do x(.i.):=i*pi/12;
   for i:=0 to 24 do y(.i.):=sin(x(.i.));
   chplot(1,25,x,y);                              /*tegning af grafer*/
   chterm;                                        /*afslutning af PGF*/
   asread(i,i,i);                                 /*billede til skærm*/
   fsterm;                                       /*nedlukning af GDDM*/
end.
```

KØRSEL AF GRAFIKPROGRAMMER.

Fremgangsmåden ved kørsel af PASCAL grafikprogrammer svarer i princippet til fremgangsmåden ved kørsel af andre PASCAL programmer, med nogle oplagte modifikationer:

Oversættelse af PASCALprogrammet, der ligger i filen: filnavn PASCAL ,sker som normalt ved kommandoen:

```
PASCALVS filnavn
```

(eventuelt i formen: 'PASCALVS filnavn (NOPRINT' for at undgå filen 'LISTING').Inden det oversatte program (med filtype: TEXT) kan indlæses i lageret, gives kommandoen:

```
GLOBAL TXTLIB PASCALVS ADMNLIB ADMRLIB
```

for at koden for de eksterne grafikrutiner, som benyttes i programmet, ved den følgende indlæsning bringes ind i lageret sammen med selve programmet (disse grafikrutiner er placeret i de to sidstnævnte grafikrutinebiblioteker: ADMNLIB og ADMRLIB , der ligesom TEXT-biblioteket PASCALVS består af køreklare filer af type TEXT). Derefter kan programmet indlæses i lageret med kommandoen:

LOAD filnavn ,

(eventuelt i formen: 'LOAD filnavn (NOMAP' for at undgå filen 'MAP'). Det der 'loades' er det TEXT-element (oversatte program), der blev frembragt ved oversættelsen af: filnavn PASCAL . Når eventuelle nødvendige fildefinitioner (FILEDEF's) af de af programmet benyttede filer er foretaget, startes programmet med kommandoen:

START prognavn .

Bemærk, at det her er nødvendigt at specificere programnavnet, altså det navn der er anført i PASCALprogrammets hoved, idet systemet ellers ikke ved hvor det skal starte. (Normalt er filnavnet og programnavnet det samme, men det kan nemt, i kampens hede, ske at de to navne bliver forskellige.)

STANDARD-GRAFIK-PROGRAMMER.

Selv om det i princippet er enkelt at frembringe EDB-grafik ved egne programmer, er der mange steder standardprogrammer til rådighed, der ud fra en datafil i et nøje specificeret format producerer en standardtegning svarende til disse data. Programmet KURVER , der blev brugt i forbindelse med Opgave 2 er netop et sådant standardprogram. Et andet eksempel er QUICKPLOT programmet på RECKU. Den tilhørende manual kan ses i terminalrummet.

Københavns Universitet
Matematisk Institut

september 1985

DATALOGI A for matematikere

OPGAVE 1

På B-disken findes et PASCAL program

TREKANT PASCAL

som indlæser koordinaterne for tre punkter og foretager nogle beregninger med disse koordinater (programmet er aftrykt nedenfor).

Opgaven består i at modificere dette PASCAL program og derefter afprøve det ændrede program med forskellige sæt testdata.

MODIFIKATIONEN skal for det første bestå i, at der udføres en forbedret test på koordinatsættene for at afgøre om der (med rimelig tolerance) foreligger en 'rigtig' trekant (f. eks. må punkterne ikke 'næsten' ligge på en fælles linie, ligesom ingen af vinklerne må være 'næsten' 0). Dette kan gøres ved at udregne determinanten for vilkårlige to af de 'udspændte' liniestykker og dernæst undersøge om denne determinant har en numerisk værdi mindre end den givne tolerance (den numeriske værdi af denne determinant er arealet af det udspændte parallelogram).

Desuden skal programmet udbygges, så der i tilfældet hvor koordinaterne bestemmer en rigtig trekant, udføres beregning og udskrift af sidelængder, vinkler samt omkreds og areal for trekanten. Ved beregningen af sidelængderne benyttes kvadratrodsfunktionen 'sqrt' (SQuaRooT) og kvadratfunktionen 'sqr' (SQuaRe). Ved vinkelberegningen kan man bruge funktionen 'arctan' (ARCTANgent) efter at man har fundet udtryk for tangens til vinklerne. Arcustangens funktionens værdi i et reelt tal er den vinkel i det åbne interval fra minus pi halve til pi halve hvis tangens er tallet, så husk at tage stumpe og rette vinkler i betragtning. Se WG side 121-122 for en omtale af PASCAL's matematiske funktioner.

SQuare RooT

Et program vil ved kørsel benytte forskellige ydre medier, typisk i form af filer, til at læse inddata fra og skrive uddata i, og det er en vigtig del af et programs SPECIFIKATION at fastlægge hvorledes denne vekselvirkning foregår, altså hvordan inddata er givet og hvorledes uddata skal præsenteres. Ofte skal resultaterne fra et program benyttes som inddata til et andet program, og det er derfor vigtigt at uddata får et helt præcist på forhånd fastlagt udseende (dette er tilfældet med resultaterne fra kørsel af opgaveprogrammet, der skal benyttes som inddata til afprøvningprogrammet). Husk at beskrivelse og specifikation i den indledende programkommentar skal ændres så den svarer til det nye programs virkning.

Inddata til programmet er de tre koordinatsæt som indlæses til programmet med en readln-sætning. De seks reelle værdier kan f.eks. stå i en enkelt linie i inddata filen (brug blanktegn som skilletegn).

Uddata fra programmet kan have to forskellige former, svarende til at de indlæste koordinater af programmet opfattes som bestemmende en rigtig trekant eller ej. I det første tilfælde skal uddata omfatte en række linier:

1. 'Koordinaterne for punkterne er:'
2. tre linier hvor de indlæste koordinater udskrives,
3. 'Punkterne danner en trekant'
4. 'Sidelængderne er (BC,AC,AB):' og BC,AC,AB,
5. 'Vinklerne er (i grader)(A,B,C):' og vinkelA,vinkelB,vinkelC,
6. 'Omkreds og areal er:' og omkreds,areal.

I tilfældet hvor de tre punkter ikke bestemmer en rigtig trekant skal der i stedet for 3. udskrives en linie:

3'. '*** Punkterne danner ikke en rigtig trekant'

og 4.-6. bortfalder. De tre stjerner i starten af fejlmeddelelsen tjener til at fremhæve denne som en fejludskrift. Nedenfor er der anført eksempler på kørselsudskrifter.

EKSEMPEL på udskrift.

```
Koordinaterne for punkterne er:
A : (       0.0,       0.0)
B : (   1.00000,       0.0)
C : (       0.0,   1.00000)

Punkterne danner en trekant

Sidelængderne er (BC,AC,AB):
   1.41421   1.00000   1.00000

Vinklerne er (i grader)(A,B,C):
  90.00000  45.00000  45.00000

Omkreds og areal er:
   3.41421   0.50000
```

EKSEMPLEL på udskrift.

```
Koordinaterne for punkterne er:
A : (       0.0,       0.0)
B : (   1.00000,   1.00000)
C : (   2.00000,   2.00000)

*** Punkterne danner ikke en rigtig trekant
```

I øvrigt kan udskriften gøres pænere ved (som gjort ovenfor) at tilføje blanke linier og linieskift. For at gøre udskriften af et programs resultater overskuelig og letlæselig, er det for det meste nødvendigt at specificere hvor meget de enkelte dele skal fylde. I programmet er udskriften af koordinaterne styret ved formatangivelser, hvor f. eks. variablen A1 af type REAL med formatangivelsen: A1:10:5 , ved udskriften fylder ialt 10 pladser (inklusive decimalpunktum), hvoraf de 5 benyttes til cifrene efter

decimalpunktum. Tilsvarende ville en variabel ANTAL af type INTEGER med formatangivelsen: ANTAL:8 , ved udskrift optage ialt 8 pladser.

AFPRØVNING af programmet foretages ved at skrive filer med forskellige sæt inddata og derefter køre programmet med disse inddata filer ved hjælp af TEST-programmet (beskrevet i Indledning til Datalogi A, Afsnit 5). Som et minimum af afprøvning skal der foretages prøvekørsel med inddata for de forskellige typer af beliggenhed for tre punkter, som f. eks. tre punkter hvoraf to falder sammen, tre parvis forskellige punkter som ligger på en fælles ret linie, tre punkter der bestemmer en rigtig trekant, tre punkter der bestemmer en retvinklet trekant og tre punkter der bestemmer en stumpvinklet trekant.

spidsvinklet?

PASCAL PROGRAMMET.

```
program TREKANT; /*Oplæg til opgave 1/GF*/

 /*Beskrivelse: Programmet læser koordinaterne til tre punkter  */
 /*beregner indbyrdes afstande mellem punkterne og afgør ud fra */
 /*disse om punkterne bestemmer en trekant.                     */

 /*Specifikation:                                               */
 /*Input før: Indeholder 6 reelle tal.                          */
 /*      efter: Uændret.                                        */
 /*Output før: Ubestemt.                                        */
 /*      efter: Indeholder 4 linier med udskrift af punkterne   */
 /*         samt 1 linie med en tekst der beskriver om punkterne*/
 /*         danner en trekant.                                  */

   const tolerance=0.00001;

   var A1,A2,B1,B2,C1,C2,                    /*koordinaterne */
       AB,AC,BC:real;                        /*de tre 'sider'*/

begin
   readln(A1,A2,B1,B2,C1,C2);                   /*indlæsning*/

   writeln('Koordinaterne for punkterne er:');     /*udskrift*/
   writeln('A : (',A1:10:5,',',A2:10:5,')');
   writeln('B : (',B1:10:5,',',B2:10:5,')');
   writeln('C : (',C1:10:5,',',C2:10:5,')');
   writeln;

   AB:=sqrt(sqr(A1-B1)+sqr(A2-B2));                /*afstande*/
   AC:=sqrt(sqr(A1-C1)+sqr(A2-C2));
   BC:=sqrt(sqr(B1-C1)+sqr(B2-C2));

   if (AB<tolerance) or                               /*test*/
      (AC<tolerance) or
      (BC<tolerance)
   then writeln('*** Punkterne danner ikke en rigtig trekant')
   else begin
           writeln('Punkterne danner en trekant');
           /*Her kan foretages flere beregninger og udskrift*/
        end;
end /*TREKANT*/ .
```

Københavns Universitet
Matematisk Institut

september 1985

DATALOGI A for matematikere

OPGAVE 2

På B-disken findes et PASCAL program

SIMUL PASCAL

som kan benyttes til simulering af et (over)simplificeret biologisk system bestående af planter og dyr. PASCAL programmet er aftrykt nedenfor.

Opgaven består i at modificere programmet og afprøve det ændrede program med forskellige sæt prøvedata.

Den BIOLOGISKE MODEL forudsætter, at

Tilvækst i antallet af planter skyldes formering af planterne og er proportional med antallet af planter.

Fald i antallet af planter skyldes at planterne ædes af dyrene, og faldet er proportionalt med produktet af antallet af planter og antallet af dyr.

Tilvækst i antallet af dyr skyldes at dyrene æder planterne, og tilvæksten er proportional med produktet af antallet af planter og antallet af dyr.

Fald i antallet af dyr skyldes en naturlig dødelighed, og faldet er proportionalt med antallet af dyr.

Kaldes plantemængden for x og dyremængden for y betyder disse antagelser, at differentialkvotienterne med hensyn til tiden dx/dt og dy/dt opfylder

$$dx/dt = A*x - B*x*y$$

$$dy/dt = C*x*y - D*y$$

hvor A, B, C og D er (konstante) proportionalitetsfaktorer.

Det ses, at konstanten A beskriver, hvor hurtigt plantemængden x vokser eksponentielt, hvis der ikke er nogen dyr, og at konstanten D beskriver, hvor hurtigt dyremængden y aftager eksponentielt, hvis der ikke er nogen planter. Systemet er i ligevægt for

$$(x,y) = (D/C, A/B),$$

idet der da gælder

$$(dx/dt, dy/dt) = (0,0).$$

I programmet bliver differentialligningssystemet 'løst' numerisk ved EULER's metode, som ud fra begyndelsesværdier for x,y (t=0), giver en tilnærmet tabel over (x(t),y(t)) for en række t-værdier, successivt ved i hvert t-delinterval fra r til s at benytte konstanterne x'(r),y'(r) (der findes fra ligningssystemet) som approksimation til funktionerne x'(t),y'(t) på delintervallet fra r til s hvorefter x(s),y(s) kan beregnes.

PROGRAMMET får INDDATA fra input filen, der består af en eller flere linier hver indeholdende tre reelle tal og et heltal. Disse værdier opfattes af programmet som henholdsvis begyndelsesværdier for x og y , tidshorisonten , dvs. den tid systemet skal føres frem, og antal tidsskridt der skal benyttes. Det sidste må ikke være for lille, hvis der skal opnås en rimelig nøjagtighed, idet x og y normalt vil ændre sig indenfor det enkelte tidsskridt.

Programmets UDDATA placeres i output filen: Først en linie der udskriver de indlæste værdier og dernæst linier med startværdierne for x og y og de nye (beregnede) værdier for hvert tidsskridt. Der afsluttes med en tom linie, og hver ny linie (hvis der er flere) i input filen giver anledning til en tilsvarende udskrift i output filen.

OPGAVEN går ud på at ændre programmet så der udføres en kontrol af de indlæste talværdier: at startværdierne af x, y og total_t alle er positive og at antallet af tidskridt n ligger mellem 10 og 100 (inklusive). Hvis disse betingelser er opfyldt skal programmet gennemregne simuleringen, og ellers skal programmet reagere med meddelelsen:

*** Ugyldige inddata: x=xx,y=yy,total_t=tt,n=nn

hvor xx,yy,tt og nn er de indlæste værdier, samt en tom linie og derefter gå videre til eventuelle følgende sæt inddata.

Desuden skal selve den tilgrundliggende model ændres, så proportionalitetsfaktoren A kun er konstant 0.2 for x<1000, men derefter aftager lineært mod 0, når x vokser fra 1000 til 2000, for da at forblive 0. For den biologiske model svarer dette til at væksten i plantemængden dæmpes ved voksende plantemængde for helt at standse når plantemængden når 2000.

AFPRØVNING foretages ved at skrive filer med forskellige sæt testdata og kørsel ved hjælp af TEST.

EN PRØVEUDSKRIFT

```
Indlæste værdier: x=900.0000 y=100.0000 total_t= 50.0000 n=  10
   900.000   100.000
   900.000    95.000
   945.000    92.388
  1016.938    93.170
  1069.171    96.392
  1033.788    98.021
  1019.320    98.968
  1010.150    99.470
  1005.252    99.731
  1002.676    99.865
  1001.351    99.932

*** Ugyldige inddata: x=950.0000 y= 90.0000 total_t= 50.0000 n= 200
```

PASCAL PROGRAMMET

```
program SIMUL;/*Oplæg til Opgave 2/GF*/

 /*Beskrivelse: Programmet læser startværdier for en simulering*/
 /*af et biologiske system og udskriver simulationsresultatet. */
 /*Differentialligningerne løses numerisk vha. EULER's metode. */

 /*Specifikation:                                                */
 /*Input før: Indeholder en eller flere linier hver med 3      */
 /*           reelle (x,y,total_t) og 1 helt tal (n).          */
 /*      efter: Uændret.                                       */
 /*Output før: Ubestemt.                                       */
 /*      efter: Består af en samling tabeller, en for hver li- */
 /*           nie i inputfilen. Hver tabel består af en linie  */
 /*           linie med de indlæste værdier, linier med de be- */
 /*           regnede x,y-værdier samt en blank linie.         */

const
    A=0.2;
    B=0.002;
    C=0.0001;
    D=0.1;
var
    x,                                          /* plantemængde */
    y,                                            /* dyremængde */
    total_t,                                    /* tidshorisont */
    delta_t   :real;                              /* tidsskridt */
    n,                                        /* antal tidsskridt */
    i         :integer;                      /* nummer tidsskridt */
begin
    repeat
        readln(x,y,total_t,n);                    /* indlæsning */
        writeln('Indlæste værdier: x=',x:8:4,      /* udskrift */
                ' y=',y:8:4,
                ' total_t=',total_t:8:4,
                ' n=',n:4);
        delta_t:=total_t/n;          /* udregning af tidsskridt */
        writeln(x:10:3,y:10:3);           /* udskrift af x og y */
        for i:=1 to n do
        begin
            x:=x+(A*x-B*x*y)*delta_t;  /* systemet føres et  */
            y:=y+(C*x*y-D*y)*delta_t;  /* tidsskridt frem og */
            writeln(x:10:3,y:10:3);    /* udskrift af x og y */
        end;
        writeln; /*udskrift af tom linie inden evt. nye uddata*/
    until eof
end /*SIMUL*/.
```

GRAFISK FREMSTILLING AF SIMULERINGEN

Uddatafilerne kan ses grafisk med et tegneprogram KURVER som findes på B-disken. Prøv at se på resultater af forskellige kørsler med de to programmer ved hjælp af KURVER.

Programmet læser en datafil, som indeholder koordinatsættene for en række punkter, og konstruerer på basis af disse punkter et 'linieplot': Der tegnes koordinatakser, og hvert par af på hinanden

følgende koordinatsæt i filen giver på tegningen anledning til liniestykket der forbinder punkterne.

Datafilen skal have følgende udseende: Hver datalinie SKAL indeholde to tal, adskilt af blanktegn, og disse to tal opfattes som x og y koordinaterne for et punkt. Blanke (tomme) linier og linier med andet indhold end netop to tal i datafilen opfattes som adskillelse mellem data for forskellige kurver, og programmet KURVER kan tegne op til seks kurver på samme tegning (forskellige kurver tegnes med forskellige farver).

Datafilen må indeholde op til 1000 koordinatsæt.

Datafilen specificeres ved filnavn og filtype (udelades filtypen sættes denne til 'FILE').

Indeholder filen: UDDATA FILE A , koordinatsættene for de kurver, der ønskes tegnet, vil kommandoen:

KURVER UDDATA

bevirke udtegning af kurverne.

Kommandoen KURVER kan også anvendes fra fillisten (i linien der indeholder datafilen).

NB! Kun dataskærme af type 2B og 3B kan benyttes til grafik. I terminalrummet er terminalerne: 1,2,3,4,7,11,12 udstyret med skærme af disse typer.

Bedre eksempel:

... filen: Mxy-2 TESTOUT , koord ...

... vil kommandoen:

KURVER Mxy-2 TESTOUT

...

Københavns Universitet
Matematisk Institut

september 1985

DATALOGI A for matematikere

OPGAVE 3

Opgaven går ud på at skrive et PASCAL program til løsning af lineære ligningssystemer. Metoden der skal benyttes er Gauss elimination, se MAT1 LD noter, afsnit 2.3, og programskrivningen kan tage udgangspunkt i et PASCAL programskelet, der findes i filen:

LINLIGN PASCAL

på B-disken (programmet er aftrykt nedenfor).

METODEN består løst sagt i at eliminere den første ubekendte fra ligningerne 2,3,... ved at addere passende multipla af den første ligning, dernæst at eliminere den anden ubekendte fra ligningerne 3,4,... ved at addere passende multipla af den (nye) anden ligning og således videre indtil ligningssystemet har øvre trekantform. Derefter kan ligningssystemet løses 'nedefra', eller det kan gives diagonalform hvorefter løsningerne direkte kan udregnes.

Programmet LINLIGN virker kun, hvis det ved eliminationen af den i'te ubekendte gælder, at koefficienten til den i'te ubekendte i den i'te ligning er forskellig fra nul. Ellers skal der foretages såkaldte diagonalkorrektioner. Det færdige program skal fungere for alle tilpas 'regulære' koefficientmatricer og reagere med meddelelsen:

'*** Systemet er singulært'

hvis det under udregningerne konstateres, at koefficientmatricen er ikke-regulær.

Det er imidlertid ikke helt uproblematisk at afgøre om en variabel af type REAL efter forskellige beregninger tillægges værdien 0 .

PASCAL variable af type integer er i maskinen (i vores version af PASCAL) lagret i såkaldte 'ord' bestående af 32 binære cifre (0 eller 1), eller bit , hvoraf et benyttes til fortegn og resten til selve tallet. Datatypen INTEGER giver således mulighed for en nøjagtig repræsentation af alle hele tal
fra -2147483648 (= -(2 opløftet til 31 potens))
til 2147483647 (= (2 opløftet til 31 potens) - 1).

PASCAL variable af type real er i maskinen lagret i ord af længde 64 (bit), hvoraf 1 bruges til fortegnsangivelse, 7 bruges til 'eksponent' og de resterende 56 til 'mantissen'. Dette giver mulighed for i datatypen REAL at repræsentere visse (men ikke alle) reelle tal. Den numeriske værdi af et sådant maskintal eller som det også kaldes flydende tal ligger i området: fra 5.4E-79 til 7.2E75 (cirka) . Disse tal skal læses som 5.4 gange 10 opløftet til minus 79'te potens henholdsvis 7.2 gange 10 opløftet til 75'te potens. Derudover kan tallet 0 repræsenteres som et flydende tal. I

modsætning til repræsentationen af heltal, som er eksakt indenfor talområdet, er repræsentationen af (matematiske) reelle tal for det meste tilnærmet, og dette må der tages hensyn til ved anvendelserne. F.eks. er det ikke uden videre meningsfuldt at spørge om resultatet af en beregning med real's er = 0 , selv om beregningen matematisk set skulle give resultatet 0 . I programmet skal det for at afgøre om en størrelse er 'matematisk set' lig 0 undersøges om dens numeriske værdi er mindre end en på forhånd valgt tolerance f.eks. af størrelsesorden 1E-9 .

Udtrykt i decimale cifre betyder repræsentationen af reelle tal, at reelle tal er givet med ca. 14 korrekte cifre, og dette giver anledning til forskellige problemer i forbindelse med beregninger. Betragt det lineære ligningssystem:

(1) x + 1E20*y = 1E20 .

(2) x - y = 0 ,

Trækkes ligning (1) (multipliceret med 1) fra ligning (2) fås systemet:

x + 1E20*y = 1E20 ,

(1+1E20)*y = 1E20 ,

med løsningerne: y = 1E20/(1 + 1E20) som i maskinen er repræsenteret som tallet 1 og x = 1E20 - 1E20 , som i maskinen er repræsenteret ved et tal af størrelsesorden 1E-75 dvs. 0 . Benyttes i stedet ligning (2) til at eliminere x fra ligning (1) fås systemet:

x - y = 0 ,

(1+1E20)*y = 1E20 ,

med løsningerne: y = 1E20/(1 + 1E20) som i maskinen er repræsenteret som 1 og x = y (som er repræsenteret som 1).

Den eksakte løsning er: x = y = 1E20/(1 + 1E20) som er lig 1 pånær 1E-20 (cirka). Problemet ovenfor ligger i at koefficienten til x i (1) er forsvindende i forhold til koefficienten til y og derfor er uegnet til brug i 'eliminationen'. Dette problem kan afhjælpes (men ikke i alle tilfælde løses fuldstændigt) ved en hensigtsmæssig tilrettelægning af beregningerne.

En overkommelig metode, som i praksis er tilstrækkeligt god, består i ved reduktionen af ligningssystemet til øvre trekantform at foretage RÆKKEOMBYTNING , således at der ved behandlingen af den j'te søjle sørges for, at rækken med det numerisk største a(.i,j.), i>=j, bringes på den j'te rækkes plads, og at det derefter er multipla af denne række der adderes til de øvrige rækker (i>j). Derved reduceres afrundingsfejlene ved disse operationer.

For at sikre at ligningssystemets koefficienter er 'sammenlignelige' skal ligningssystemet ækvilibreres inden reduktionen til øvre trekantform foretages. Denne ÆKVILIBRERING sker ved at multiplicere hver af ligningerne med en passende konstant, så mindst en af koefficienterne til de ubekendte i hver række har værdi 1, og så alle disse koefficienter ligger mellem -1 og 1. Under denne ækvilibrering kan det opdages om en af ligningerne har lutter 'nul'-koefficienter (om alle koefficienter numerisk er mindre end tolerancen), i hvilket tilfælde programmet

Er det krystalklart? Skal der ækvilibreres 20 inden behandlingen af j-te søjle for j = 1..n ?

skal stoppe. Videre skal det, ved reduktionen til øvre trekantform, undersøges om den numeriske værdi af det diagonalelement der skal benyttes ved eliminationen, er mindre end tolerancen, i hvilket tilfælde programmet skal stoppe. Glem ikke at undersøge det sidste diagonalelement. En bekvem måde at få programmet til at stoppe er ved hjælp af sætningen (procedure'n) HALT , anbragt på det sted i programmet hvor der skal stoppes. For en nærmere omtale af HALT henvises til PASCAL manualerne i terminalrummet.

Udover disse ændringer, skal der i det færdige program foretages en KONTROL af det indlæste antal ligninger, og hvis antallet er større end maxn skal programmet reagere med meddelelsen:

'*** For mange ligninger'

og derefter stoppe.

AFPRØVNING foretages på sædvanlig måde ved hjælp af TEST. Som et minimum skal der foretages testkørsel med inddata filer, der afprøver de forskellige 'fejl'-reaktioner fra programmet.

RESULTATER FRA EN KØRSEL.

Inddata:
3
1 0 0 1
0 1 0 1
0 0 1 1

Uddata:
x1 = 1.0000000000000E+00
x2 = 1.0000000000000E+00
x3 = 1.0000000000000E+00

RESULTATER FRA EN KØRSEL.

Inddata:
3
1 2 3 1
4 5 6 2
7 8 9 3

Uddata:
*** Systemet er singulært

PASCAL PROGRAMMET

```
program LINLIGN; /*Oplæg til opgave 3/GF*/

 /*Beskrivelse: Programmet indlæser et heltal n og derefter et   */
 /*lineært ligningssystem af n ligninger med n ubekendte.        */
 /*Ligningssystemet løses ved GAUSS-elimination.                 */
 /*Virker kun når der ikke skal foretages diagonalkorrektioner.  */

 /*Specifikation:                                                */
 /*Input før: Indeholder en linie med et positivt heltal n, samt */
 /*         linier med (n+1)*n reelle tal (koefficienterne).     */
 /*      efter: Uændret.                                         */
 /*Output før: Ubestemt.                                         */
 /*      efter: Indeholder n linier med de fundne løsninger.     */


const maxn=10;

type matrix=array(.1..maxn,1..maxn.) of real;
     vektor=array(.1..maxn.) of real;
var n,                                          /*antal ligninger*/
    i,j,k:integer;                               /*indeksvariable*/
    kvot:real;                                   /*hjælpevariabel*/
    a:matrix;                                 /*koefficientmatricen*/
    b,x:vektor;                              /*højreside og løsning*/

begin

   readln(n);                                          /*indlæsning*/
   for i:=1 to n do
   begin
      for j:=1 to n do read(a(.i,j.));
      read(b(.i.));
   end;

   for i:=1 to n-1 do            /*reduktion til øvre trekantform*/
   for j:=i+1 to n do
   begin
      kvot:=-a(.j,i.)/a(.i,i.);
      for k:=i to n do a(.j,k.):=a(.j,k.)+kvot*a(.i,k.);
      b(.j.):=b(.j.)+kvot*b(.i.);
   end;

   for j:=n downto 2 do  /*beregning af højresider i diagonalsys.*/
   for i:=j-1 downto 1 do         /*diagonalelementerne står i a */
      b(.i.):=b(.i.)-b(.j.)*a(.i,j.)/a(.j,j.);

   for i:=1 to n do x(.i.):=b(.i.)/a(.i,i.);        /*løsningerne*/

   for i:=1 to n do                          /*udskrift af løsninger*/
      writeln('x',i:0,' =',x(.i.));

end /*LINLIGN*/ .
```

Københavns Universitet
Matematisk Institut

september 1985

DATALOGI A for matematikere

OPGAVE 4

Opgaven går ud på at skrive et PASCAL program, der finder og udskriver samtlige delmængder bestående af m elementer af mængden:

1,2, ... ,n .

En delmængde A af 1,2,...,n med m elementer (0<m<=n) er givet ved sine elementer og kan altså beskrives som et voksende m-sæt af tallene 1,2,...,n , og et sådant voksende m-sæt kan bestemmes ved indholdet af en array variabel a

var a: array(.1..m.) of integer

hvor a(.1.),...,a(.m.) netop er de af tallene 1,2,...,n , der indgår i det voksende m-sæt (eller delmængden). Der gælder:

1<=a(.1.)<a(.2.)< ... <a(.m.)<=n .

Disse voksende m-sæt er på naturlig måde ordnede efter et leksikon-princip: For to forskellige voksende m-sæt A og B bestemt ved a(.1.),...,a(.m.) og b(.1.),...,b(.m.) siges A at komme før B (eller B at komme efter A) hvis

a(.j.) < b(.j.) ,

hvor j er nummeret på den første plads hvor a og b er forskellige.

Herved fastlægges er ordning (også i matematisk betydning) af de voksende m-sæt (eller delmængder) med en række nyttige egenskaber:

1) Vilkårlige voksende m-sæt A og B kan sammenlignes idet der gælder at A kommer før B eller at B kommer før A (eller de er ens).

2) m-sættet bestående af 1,2,...,m er det første dvs. det kommer før et vilkårligt andet voksende m-sæt og m-sættet bestående af n-m+1,n-m+2,...,n er det sidste dvs. det kommer efter et vilkårligt andet voksende m-sæt.

3) Endvidere findes til hvert voksende m-sæt A pånær det sidste et entydigt bestemt efterfølger voksende m-sæt B som opfylder, at A kommer før B og B kommer før ethvert andet voksende m-sæt der kommer efter A.

PASCAL PROGRAMMET skal indeholde definition af en konstant nmax, der angiver den maksimale værdi af n som akcepteres af programmet. Værdien af nmax sættes til 20, men programmet skal kunne virke efter udskiftning af nmax.

Programmet skal starte med at indlæse to heltal m og n fra input filen og undersøge betingelsen

0 < m <= n <= nmax .

Hvis betingelsen er opfyldt skal programmet udskrive de indlæste værdier samt en ordnet liste over de voksende m-sæt (delmængder) af 1,2,...,n , og ellers skal der udskrives en meddelelse, om at inddata er ugyldige. Se eksempler på udskrifter nedenfor.

De voksende m-sæt repræsenteres ved hjælp af datatyperne specificeret ved:

```
type element=1..nmax;
     delm   =array(.1..nmax.) of element;
```

(hvor det i praksis kun er de m første pladser der bruges):

Selve beregningen af listen af delmængder skal ske ved hjælp af en procedure, der som parameter får et voksende m-sæt, som ikke er det sidste, og derudfra finder efterfølger m-sættet (beskrevet ovenfor) og leverer dette til hovedprogrammet i parameteren.

Specifikationen af proceduren skal være følgende:

```
procedure efterf(var a:delm);
/* a før:   Et voksende m-sæt af 1,2,...,n der ikke er det sidste*/
/*   efter: Det næste voksende m-sæt                              */
/* n før:   Det totale antal elementer                            */
/*   efter: Uændret                                               */
/* m før:   Antal aktive elementer i a                            */
/*   efter: Uændret                                               */
```

Hovedprogrammet skal tage sig af indlæsning, kontrol af inddata, udskrift og styre kaldene af proceduren efterf. Husk at give en beskrivelse af hovedprogrammets virkning og en specifikation af programmets vekselvirkning med kørselsomgivelserne i de indledende kommentarer til programmet.

EKSEMPEL på en kørselsudskrift:

```
*** Ugyldige inddata: m=  19 n=   8
```

EKSEMPEL på en kørselsudskrift:

```
De indlæste værdier: m=   3 n=   5
  1  2  3
  1  2  4
  1  2  5
  1  3  4
  1  3  5
  1  4  5
  2  3  4
  2  3  5
  2  4  5
  3  4  5
```

Københavns Universitet
Matematisk Institut

oktober 1985

DATALOGI A for matematikere

OPGAVE 5

Opgaven går ud på at skrive et PASCAL program, der indlæser et LP-problem på den i LP-noterne beskrevne normalform, og udskriver en tilladt basisløsning, for hvilken objektfunktionen er mindst mulig, samt objektfunktionens minimumsværdi i denne basisløsning.

Antag, at LP-problemet (på normalform) har n variable og m bibetingelser (m<=n). Programmet skal lede samtlige delsæt bestående af m af de n variable igennem, undersøge om der findes en tilhørende basisløsning, om en eventuel sådan er tilladt og i bekræftende fald beregne objektfunktionens værdi i den. Derved kan en eventuel tilladt basisløsning med minimal værdi af objektfunktionen findes.

Programmet skal indeholde definition af to konstanter mmax og nmax, der angiver de maksimale værdier af m og n , som accepteres af programmet. Disse konstanter sættes begge til 6 , men programmet skal kunne virke, når disse grænser udskiftes.

Programmet skal styre søgningen efter basisløsninger ved hjælp af en variabel

```
basis
```

af type

```
ARRAY(.1..mmax.) of INTEGER ,
```

som indeholder numrene på de koordinater der indgår i det aktuelle m-delsæt. Denne arrayvariabel basis gives startværdierne 1, 2, ... ,mmax , og programmet skal benytte proceduren EFTERF, der blev brugt i Opgave 4, til at gennemlede m-delsættene med.

For hvert m-delsæt af de n koordinater skal en eventuel tilhørende basisløsning bestemmes, og dette skal programmet gøre vha. en procedure GAUSS , der i det væsentlige er programmet fra Opgave 3 i procedure forklædning.

Til brug for kommunikationen mellem hovedprogrammet og proceduren GAUSS erklæres en variabel af type MATRIXTYPE , hvor MATRIXTYPE er erklæret ved:

```
TYPE  MATRIXTYPE=(REGULAR,SINGULAR)
```

og GAUSS giver denne variabel værdien: REGULAR eller SINGULAR afhængigt af om den anvendte koefficientmatrix har været regulær eller ej.

Proceduren GAUSS skal forsynes med en omhyggelig specifikation af underprogrammets vekselvirkning med omgivelserne svarende til den, der blev givet for proceduren EFTERF , i oplægget til Opgave 4. Hvis resultatet REGULAR/SINGULAR kommunikeres til hovedprogrammet via en global variabel, skal dette fremgå af specifikationen. Proceduren HALT , som blev nævnt i oplægget til Opgave 3 , erstattes naturligt af proceduren RETURN , der ligeledes er beskrevet i PASCAL-manualerne i terminalrummet.

Programmet starter med at indlæse en linie med værdierne for m og n og kontrollerer, at disse værdier opfylder:

$$1 <= m <= mmax \quad \text{og} \quad m <= n <= nmax .$$

Er disse betingelser ikke opfyldt skal programmet reagere med meddelelsen:

'*** Ugyldige inddataværdier for (m,n).'

og derefter stoppe.

I tilfældet hvor betingelserne er opfyldt skal programmet derefter indlæse n reelle tal (koefficienterne i objektfunktionen), samt m linier hver med n+1 reelle tal (koefficienter og højreside i bibetingelserne).

Programmet skal så begynde at lede efter tilladte basisløsninger svarende til de forskellige m-delsæt af de n variable. Denne ledeproces kan have forskellige resultater: Hvis rangen af koefficientmatricen er < m , vil ingen af de ligningssystemer som GAUSS skal løse, have løsning, og i dette tilfælde skal programmet reagere med meddelelsen:

'*** Ingen regulære delmatricer.' .

En anden mulighed er, at der findes en eller flere regulære delmatricer og dermed basisløsninger, men at ingen af disse er tilladte. I så fald skal programmet reagere med meddelelsen:

'*** Ingen tilladte basisløsninger.' .

Endelig er der det 'normale' tilfælde hvor der findes mindst een tilladt basisløsning, og her skal programmet udskrive koordinaterne for den første (se nærmere nedenfor) af de tilladte basisløsninger, der giver minimal værdi af objektfunktionen samt objektfunktionens værdi i denne basisløsning.

Programmet skal, mens de forskellige m-delsæt gennemledes, gemme en eventuel tilladt basisløsning og tilhørende værdi af objektfunktionen, og hvis der senere i søgeprocessen findes en anden tilladt basisløsning skal objektfunktionens værdi i denne basisløsning beregnes og sammenlignes med den tidligere fundne værdi. Er objektfunktionens værdi i den nye basisløsning mindre end værdien i den 'huskede' basisløsning, skal den nye basisløsning og tilhørende værdi af objektfunktionen erstatte den 'huskede'. For at tage hensyn til afrundingsfejl ved beregningerne skal denne udskiftning dog kun foretages hvis den nye objektfunktionsværdi er 'skarpt mindre' end den huskede, dvs. mindre end eller lig med den gamle værdi minus tolerancen (som er af størrelsesorden: 1E-8). På denne måde sikres, at det i LP-problemer med flere tilladte

basisløsninger, der giver samme minimale værdi af objektfunktionen, altid er den første af disse (i den naturlige rækkefølge), som programmet udpeger.

For det nærmere udseende af udskriften i normaltilfældet, se prøveudskrifterne nedenfor.

RESULTATER FRA EN KØRSEL.

```
Inddata:
 3 4
 1 1 1 1
 1 2 3 4 5
 2 3 4 5 6
 3 4 5 6 7

Uddata:
*** Ingen regulære delmatricer.
```

RESULTATER FRA EN KØRSEL.

```
Inddata:
 2 2
 0 0
 1 0 -4
 0 1  2

Uddata:
*** Ingen tilladte basisløsninger.
```

RESULTATER FRA EN KØRSEL.

```
Inddata:
 2 2
 1 1
 1 0  5
 0 1  4

Uddata:
Objektfunktionens minimum i tilladte basisløsninger er:
   9.00000
Den første basisløsning hvor minimum antages har koordinaterne:
x1=   5.00000
x2=   4.00000
```

Københavns Universitet
Matematisk Institut

november 1985

DATALOGI A for matematikere

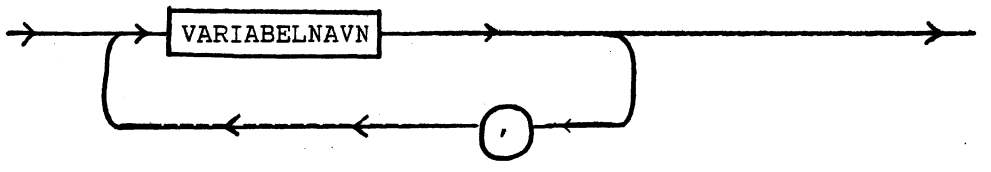
OPGAVE 6

Opgaven går ud på at skrive et PASCAL program, der indlæser et LP-problem skrevet symbolsk, og omformer dette til et LP-problem på normalform.

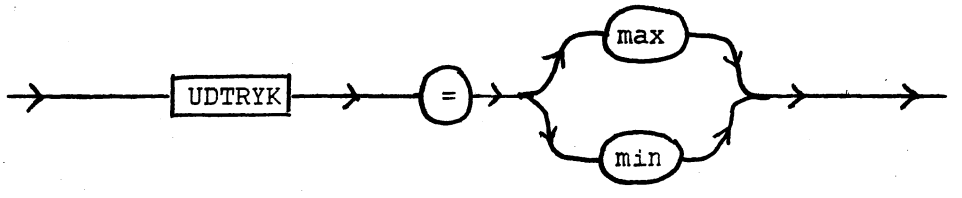
Inddata filen indeholder en linie med en VARIABELNAVNELISTE, en linie med specifikation af OPTIMERINGSBETINGELSEn, dvs. af formen et UDTRYK opbygget ved hjælp af variabelnavnene, konstanter og '+','-' sat lig med enten 'max', hvis udtrykket skal maksimeres, eller 'min' , hvis udtrykket skal minimeres. Derefter kommer en eller flere linier med BIBETINGELSEr, der alle er af formen et udtryk efterfulgt af et af relationssymbolerne '<=' , '=' og '>=' efterfulgt af et udtryk. Den nærmere fastlæggelse fremgår af syntaksdiagrammerne nedenfor (se også eksemplerne senere):

SYNTAKSDIAGRAMMER.

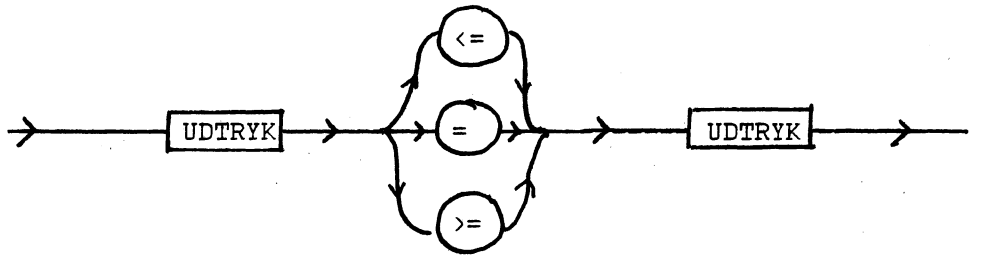
VARIABELNAVNELISTE



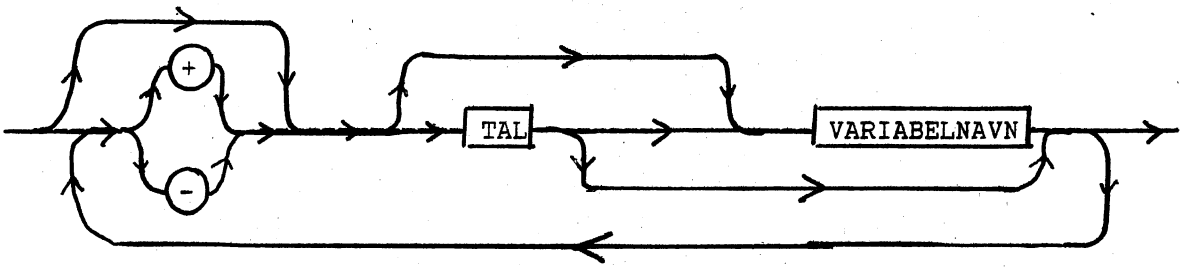
OPTIMERINGSBETINGELSE



BIBETINGELSE



UDTRYK



Her står VARIABELNAVN for en tegnstreng af bogstaver (også store og små 'æ', 'ø' og 'å') og cifre. Første tegn skal være et bogstav.

TAL ~~Tal~~ står for en tegnstreng, der starter med et ciffer og kan indlæses i en PASCAL variabel af type REAL med proceduren READ . Blanktegn skal behandles som separatorer ligesom i PASCAL, (se (WG) nederst side 51).

Programmet skal indeholde definitionen af tre konstanter mmax (det maksimale antal bibetingelser), vmax (det maksimale antal variabelnavne) og nmax, som er summen af mmax og vmax. Konstanterne mmax og vmax sættes begge til 6, men programmet skal kunne fungere efter ændring af værdierne.

Inddatafilen skal udover syntaksreglerne for de respektive linier opfylde en række andre krav:

1) Variabelnavne må ikke begynde med 'E' eller 'e', da man ellers kan få problemer med eksponent-notationen for reelle tal (hvor 'E' indgår).
2) Der må ikke være gentagelser i VARIABELNAVNelisten (eventuelle gentagelser skyldes nok fejlindtastning).
3) Der skal være mindst 1 og højst vmax variabelnavne erklæret.
4) Der skal være mindst 1 og højst mmax bibetingelse(r).
5) Variabelnavne, der indgår i optimeringsbetingelsen og bibetingelserne skal være erklærede i variabelnavnelisten.

Programmet skal læse inddata filen tegn for tegn (næsten) og efterhånden kontrollere, at linien under behandling er i overensstemmelse med (syntaks)reglerne for den pågældende linie. Hvis programmet konstaterer, at en linie indeholder en fejl skal dette meddeles med angivelse af linienummeret og en beskrivelse af fejlens art, hvorefter programmet skal stoppe. Programmet kan antage, at der findes mindst 3 linier i inputfilen.

I tilfældet hvor der ikke findes fejl i inddatafilen skal uddata se ud på følgende måde:

1) En linie med de benyttede variabelnavne.
2) En linie med oplysningen 'max' eller 'min'.
3) En linie med konstanten i objektfunktionen.
4) En linie med antal bibetingelser m og antal variable n. Her er n lig med antal erklærede variable plus antal restvariable, idet hver bibetingelse med relationstegn '<=' eller '>=' giver anledning til en 'restvariabel'.
5) En linie med n reelle konstanter, koefficienterne i objektfunktionen (med modsat fortegn, hvis der er tale om

en maksimeringsopgave).
6) m linier hver med n+1 reelle konstanter, koefficienterne og højresiden i bibetingelserne.

For det nærmere udseende af resultatudskriften henvises til eksemplerne nedenfor. Bemærk, at 4)-6) ovenfor svarer til inddata for opgave 5.

Idet hovedparten af programmet skal indgå i Opgave 7 skal det struktureres på følgende måde: selve indlæsningen og behandlingen af input filen foretages af en (stor) procedure mens et (lille) hovedprogram kalder denne procedure og udskriver resultater (eller fejlmeddelelser).

FEJLMEDDELELSER.

Indlæsningen af VARIABELNAVNELISTEn skal ske på følgende måde: for hvert element i listen undersøges om det første tegn er et tilladt bogstav og hvis dette er tilfældet læses ~~(og huskes)~~ alle følgende tegn indtil (men ikke med) det første tegn, der ikke kan indgå i et navn. Dernæst undersøges om der kommer et ~~(eller flere) blanktegn~~ komma eller EOLN. Et variabelnavn må gerne indeholde mere end 8 tegn, men kun de første 8 huskes og benyttes. En eventuel fejl angives med en af følgende fejlmeddelelser:

```
*** Linie 1: Syntaksfejl.
*** Linie 1: Variabelnavn begynder med 'E' eller 'e'.
*** Linie 1: Gentagelse af variabelnavn.
*** Linie 1: For mange variabelnavne.
```

Indlæsningen af OPTIMERINGSBETINGELSEn sker i henhold til syntaksdiagrammet for optimeringsbetingelse. Man kan tænke på indlæsningen som følgende en vej i dette diagram. Afhængigt af den aktuelle position i diagrammet og det næste tegn i linien vælges en fortsættelse af turen gennem diagrammet. Konstateres det f.eks., at det næste tegn er begyndelsen på et variabelnavn læses de følgende tegn indtil der mødes et tegn, som ikke kan indgå i et variabelnavn, og dette tegn fastlægger så fortsættelsen af diagramruten (eller afbrydelse i tilfælde af fejl). Fejl ved indlæsning af OPTIMERINGSBETINGELSEn kan give en af følgende fejlmeddelelser:

```
*** Linie 2: Syntaksfejl.
*** Linie 2: Ikke erklæret variabelnavn.
```

Ved indlæsning af BIBETINGELSErne kan følgende fejl optræde (x skal så angive linienummeret for linien med fejlen):

```
*** Linie x: Syntaksfejl.
*** Linie x: Ikke erklæret variabelnavn.
*** Linie x: For mange bibetingelser.
```

Det er nemt at konstruere inddata for hvilke flere forskellige fejl kan opdages på samme tidspunkt under indlæsningen. Ved sådanne inddata kan det tænkes at facitprogrammet giver en afvigende fejlmeddelelse, selvom det testede program er korrekt. I MATTEST-maskinen benyttes sådanne tesdata ikke.

EKSEMPLER PÅ PROGRAMUDSKRIFTER.

EKSEMPEL 1.

```
Inddata:
x,y
10x + 4y - 7 = max
3x + 2y      <= 60
2y + 7x       = 84-4x
1 + 3x + 6y  >= 0

Uddata:
 x         y
max
  -7.00
   3   4
  -10.00    -4.00      0.0      0.0
    3.00     2.00     1.00      0.0    60.00
   11.00     2.00      0.0      0.0    84.00
    3.00     6.00      0.0    -1.00    -1.00
```

EKSEMPEL 2.

```
Inddata:
 a,b,c
 +10a+b-2c+15=min
 a+b-14c>=21
 7a-41c>=65
 a+b+c+d=

Uddata:
*** Linie 5: Ikke erklæret variabelnavn.
```

EKSEMPEL 3.

```
Inddata:
 a,b,c
 10a++b-2c+15=max
 c-a=15

Uddata:
*** Linie 2: Syntaksfejl.
```

BEMÆRKNINGER.

Det er af hensyn til overskueligheden fornuftigt at opdele programmet i en række underprogrammer, der løser delopgaver som f.eks. indlæsning af et VARIABELNAVN (kan benyttes ved indlæsningen af VARIABELNAVNELISTEn og ved indlæsning af VARIABELNAVNe i UDTRYK), indlæsning af et UDTRYK (kan benyttes ved indlæsning af venstre side af OPTIMERINGSBETINGELSEn og de to sider i BIBETINGELSErne), undersøgelse af et tegn (er det et bogstav, et ciffer, et relationstegn eller et ikke-tilladt tegn) samt udskrift af en fejlmeddelelse med efterfølgende programstandsning.

Programmet kan opbevare variabelnavnene (fra inddatafilens linie 1) i et array af type

```
ARRAY(.1..vmax.) OF ALFA .
```

(Vedrørende typen ALFA, se PASCAL LANGUAGE REFERENCE MANUAL side 54)

Det er nyttigt at vide, at variablen INPUTØ indeholder det tegn, som INPUT-filpilen peger på, dvs. det tegn, det næste READ-kald starter med at læse.

Bemærk, at Pascal-VS kræver at inputfilen er åbnet med Reset(input) inden inputØ kaldes første gang. Ved det første kald af read udføres dette auto.

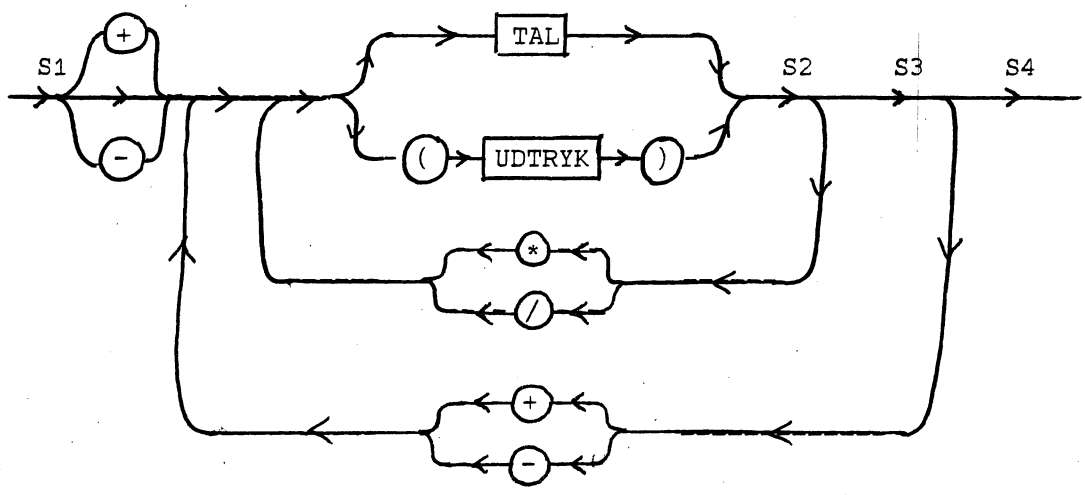
kemptisk

Det er nok lettest at indsætte tallene i de benyttede matricer efterhånden som de indlæses. Bemærk, at der må oprettes restvariable for hver bibetingelse med 'ulighedstegn'. Den signifikante koefficient bliver 1 eller -1 afhængigt af om den tilhørende relation er '<=' eller '>='.

Den praktiske udformning af den del af programmet, der skal indlæse et udtryk, kan f.eks. benytte en variabel hvis værdi angiver 'tilstanden' for indlæsningen, dvs. hvor i 'syntaksdiagrammet' for et udtryk det (eller de) senest indlæste tegn hører hjemme. Som model for denne programmeringsteknik henvises til det nedenfor aftrykte PASCAL program: COMPUTE , der er et interaktivt program til beregning af aritmetiske udtryk. I COMPUTE undersøges om et indtastet udtryk er i overensstemmelse med syntaksen (den sædvanlige) for aritmetiske udtryk opbygget af talkonstanter ved hjælp af operationerne + - * og / samt parenteser. Denne syntaks kan beskrives i nedenstående syntaksdiagram, hvor de anførte tilstande S1,S2,S3,S4 svarer til de mulige værdier af den i COMPUTE benyttede tilstandsvariabel s.

SYNTAKSDIAGRAM for COMPUTE.

UDTRYK



PASCALprogrammet COMPUTE.

```
program compute;
 /*Interaktivt program til udregning af aritmetiske udtryk */
 /*opbygget fra tal ved hjælp af + - * / og parenteser.    */
 /*Kørslen afsluttes ved afsendelse af tom linie.          */

label startagain;

var x:real;

procedure error;
begin
   writeln('*** Syntaksfejl.');
   readln;
   goto startagain;
end /*error*/;

function nextch:char;
 /*Dette underprogram læser forbi blanke inputtegn indtil  */
 /*filpilen står for et ikke blankt tegn eller eoln. Funk- */
 /*tionsværdien er det ikke blanke tegn eller ' ' ved eoln.*/
begin
   while not eoln and (inputØ=' ') do get(input);
   if eoln then nextch:=' ' else nextch:=inputØ;
end /*nextch*/;

function arithread:real;
 /*Dette underprogram læser et aritmetisk udtryk, beregner */
 /*værdien, og standser ved det første tegn, som ikke kan  */
 /*være en del af udtrykket.                               */

type state=(s1,s2,s3,s4);
var s:state;
    c,stackoperation:char;
    sum,product:real;

function numberread:real;
 /*Dette underprogram læser enten et tal uden fortegn eller*/
 /*et udtryk omgivet af parenteser.                        */

var x:real;
    c:char;
begin
   case nextch of
      '0'..'9':read(x);
      '('     : begin              /************************/
                   read(c);        /* Udelades denne blok  */
                   x:=arithread;   /* undgås rekursivt kald*/
                   if nextch=')'   /* af arithread, men så */
                   then read(c)    /* kan parenteser ikke  */
                   else error;     /* indgå i et udtryk.   */
                end;               /************************/
     otherwise error;
   end;
   numberread:=x;
end /*numberread*/;

begin /*arithread*/
```

```
   s:=s1;
   sum:=0.0;
   while s<>s4 do
   begin
      case s of
         s1: case nextch of
             '+','-': begin
                         read(stackoperation);
                         product:=numberread;
                         s:=s2;
                      end;
             otherwise begin
                         stackoperation:='+';
                         product:=numberread;
                         s:=s2;
                       end;
             end;
         s2: case nextch of
             '*','/': begin
                         read(c);
                         if c='*'
                         then product:=product*numberread
                         else product:=product/numberread;
                      end;
             otherwise begin
                         if stackoperation='+'
                         then sum:=sum+product
                         else sum:=sum-product;
                         s:=s3;
                       end;
             end;
         s3: case nextch of
             '+','-': begin
                         read(stackoperation);
                         product:=numberread;
                         s:=s2;
                      end;
             otherwise s:=s4;
             end;
      end;
   end;
   arithread:=sum;
end /*arithread*/;

begin                          /*her begynder hovedprogrammet*/

   termin(input);          /*Input/output direkte til terminal*/
   termout(output);

startagain:                        /*Fejlrutinen hopper hertil*/
   get(input);                     /*Indlæsning af næste linie*/
   while not eoln do
   begin
      x:=arithread;
      if not eoln then error;
      writeln(x:20:10);
      readln;
      get(input);
   end;
end /*compute*/.
```

Københavns Universitet
Matematisk Institut

december 1985

DATALOGI A for matematikere

OPGAVE 7

Opgaven går ud på at kombinere programmerne fra opgave 5 og opgave 6 til et LP-program, der bygger på metoden beskrevet på side 9 i LP-noterne. Inddata til programmet svarer til opgave 6. Uddata er enten en af fejlmeddelelserne fra opgave 5 og 6 (bortset fra opgave 5 meddelelsen om ugyldige værdier af m og n), eller en af fejlmeddelelserne

```
'*** Intet minimum.'
'*** Intet maksimum.'
```

eller en "normal" udskrift som i følgende eksempel:

```
Inddata:
x,y
 3x+y=max
 x+y<=3
 x+2y<=5

Uddata:
Objektfunktionens maksimum er:
    9.00000
Maksimum antages for:
    x = 3.00000
    y = 0.00000
```

Da der skal tilføjes en ekstra bibetingelse, når det undersøges om den fundne optimale basisløsning virkelig er det søgte optimum, må de arrays, der indeholder koefficienterne og højresiderne i bibetingelserne erklæres, så dette kan lade sig gøre.