HALDOR TOPSØE

19-

PROGRAMMERING AF KEMISKE BEREGNINGER

BIND 2: NUMERISKE METODER TIL LIGNINGER, POLYNOMIER og OPTIMERING

Jørgen Kjær

Haldor Topsøe, Vedbæk, Danmark

1965

Copy Nr. 1

Haldor Topsøe, Chemical Engineers

Vedbæk, Denmark

FORORD

I serien af bøger om programmering af kemiske beregninger følger hermed bind 2 om numeriske metoder til løsning af ligninger, beregning af polynomier og optimering. Bind 1 beskrev kodesproget ALGOL.

Det vigtigste formaal med denne bog er, sammen med et kommende bind, at give en samlet og systematisk oversigt over de numeriske metoder, som benyttes ved GIER-installationen hos Haldor Topsøe. Bogen skal benyttes ved undervisning af nye medarbejdere i programmeringsmetoder. Det er vigtigt, at nye programmører hurtigt bliver fortrolige med det kompleks af afprøvede numeriske metoder, som staar til deres raadighed. Paa den maade kan megen overflødig programmering spares.

Det er min erfaring, at de fleste begyndere i programmering har en tilbøjelighed til straks at kode løs paa de foreliggende problemer uden at tænke dybere over problemets natur. Hvis programmøren iøvrigt er dygtig, kan han meget vel naa et anvendeligt resultat paa denne maade, især hvis han har den nødvendige fantasi og intuition. Men programmer af denne art vil let fremtræde som et besynderligt konglomerat af numeriske smaafiduser, og programmet er svært at læse, ikke blot for andre, men ogsaa for programmøren selv. Det er mit haab, at en systematisk gennemgang af de vigtigste numeriske metoder vil gøre det lettere for begyndere at faa et overblik over, hvad det egentlig er, de ønsker at programmere, og derved hjælpe dem til at lave mere overskuelige programmer.

De numeriske metoder, som er beskrevet her, kan inddeles i to kategorier. Den første gruppe omfatter de metoder, som er beskrevet i litteraturen, og som vi ofte har kunnet kopiere uden nærmere analyse. Den anden gruppe indeholder de metoder, som i større eller mindre grad er udviklet hos Haldor Topsøe. Da det er svært at undgaa, at metoder, som er udviklet af ikke-matematikere, indeholder forskellige fejl og misforstaaelser, er det mit haab, at professionelle matematikere vil studere bogen og benytte den som inspiration til fremtidigt arbejde. Jeg er overbevist om, at adskillige af vore procedurer vil kunne forbedres betydeligt.

Ligesom i bind 1 har jeg lagt vægt paa at give mange illustrerende eksempler. Størstedelen af bogen kan sikkert læses uden anden forudsætning end gymnasiets matematikpensum. Flere steder har jeg med vilje gjort fremstillingen særlig elementær, f.eks. i afsnittet om lineære ligninger.

Civilingeniør J. Adriansen har formuleret vore procedurer til approksimation med ortogonale polynomier. Jeg vil gerne takke ham herfor, og ogsaa for anden bistand ved løsning af numeriske problemer.

Jeg takker ogsaa fru Birthe Sørensen for hulning af manuskriptet.

Vedbæk, Februar 1965

Jørgen Kjær

INDHOLDSFORTEGNELSE

1. INDLEDNING

1.1. Definition af Numeriske Metoder.

I bind 1 af denne serie om programmering af kemiske beregninger blev det generelle kodesprog ALGOL forklaret i detailler. Vi lærte bl.a. om de almindelige regneoperationer (+, -, ×, /, ↑) og om hvorledes man ved at deklarere procedurer kan skabe smaa stykker program, som løser en mere eller mindre kompliceret regneopgave, f.eks. beregning af summen af en række tal eller af trykfaldet i et rør.

I videste forstand kan man definere en numerisk metode som et vilkaarligt kompleks af regneoperationer, f.eks. nedskrevet i form af en ALGOL-procedure og med et passende antal parametre. I snævere forstand vil man stille to krav til en ALGOL-procedure, for at den med rimelighed kan kaldes en numerisk metode:

1. De anvendte parametre skal være af abstrakt matematisk natur, d.v.s. de maa ikke have en konkret betydning hentet fra et bestemt fagomraade (kemi, fysik, etc.). Vi kan bruge simple variable og talsæt:

```
a := b + c + d[j];
```

men de variable a, b, c o.s.v. maa ikke have nogen bestemt betydning af f.eks. kemisk natur. Saa længe vi bevæger os indenfor de numeriske metoder og indenfor den rene matematik i det hele taget, kan a, b, c o.s.v. være hvad som helst fra atomer til stjernetaager, vi ved det ikke og ønsker heller ikke at vide det, idet formlernes rigtighed er af ren logisk karakter, det er abstraktioner fra virkeligheden. Den ovennævnte procedure til trykfaldsberegning kan derfor ikke kaldes en numerisk metode i egentlig forstand.

2. Man maa ogsaa forlange, at proceduren skal være tilpas simpel og have et enkelt, veldefineret formaal for at anerkende proceduren som en numerisk metode. Hvad der i denne forbindelse skal opfattes som simpelt og enkelt er maaske ikke helt klart paa forhaand. Det maa i hvert fald afhænge af, hvor stort et maal man sætter sig, altsaa af hvor effektivt et

kompleks af numeriske metoder man ønsker at have til raadighed. Hvad der i een sammenhæng maaske er en simpel numerisk metode (f.eks. at finde et polynomium, der saa godt som muligt reproducerer en tabellagt funktion) vil maaske i en anden sammenhæng føles som en kompliceret metode uden praktisk betydning.

Den klassifikation af numeriske metoder, som gives i det følgende, har derfor ikke nødvendigvis nogen almen karakter. Den indeholder de fleste af de i dag anvendte numeriske metoder ved GIER-installationen hos Haldor Topsøe. Listen over numeriske metoder vil vokse, efterhaanden som flere praktiske problemer bliver taget op til løsning paa regnemaskinen, og efterhaanden som programmørerne opdager, hvilke nye matematiske procedurer, der herved bliver brug for. Det er altsaa en udviklingsproces, hvor man efterhaanden faar materiale nok til at abstrahere fra de variables praktiske betydning og kan give den generelle, matematiske formulering som nye numeriske metoder.

1.2. Klassifikation af Numeriske Metoder.

Som første numeriske metode omtales i kapitel 2 løsning af lineære ligninger. For at kunne forstaa, hvorfor dette problem er klassificeret som det simpleste, er det nødvendigt at se et øjeblik paa de endnu simplere numeriske metoder. De fire elementære regningsarter, ved hjælp af hvilke vi beregner udtryk af formen: a+b, a-b, a×b og a/b, er ogsaa en slags numeriske metoder, men blot saa simple, at vi normalt ikke tænker nærmere herover. Men for skolebørn paa et vist stadium er problemet alvorligt nok. De skal dels lære at udføre operationerne korrekt, og dels lære at afgøre, om et foreliggende regnestykke f.eks. er et gange-stykke eller et dividere-stykke.

Ser vi lidt nærmere paa divisionen:

$$X := A/B;$$

hvor X beregnes som A divideret med B, kan vi ogsaa sige, at X er løsningen til ligningen:

$$B \times X = A$$

X skal altsaa findes som det tal, der ganget med B giver A. Har vi ikke een ligning, der skal tilfredsstilles, men to:

$$B \times X1 + C \times X2 = A$$
$$D \times X1 + E \times X2 = F$$

er opgaven nu at finde de to ubekendte, X1 og X2, saaledes at ligningerne passer. Løsning af flere ligninger med flere ubekendte er altsaa en simpel generalisation af divisionsproblemet, og vi skal se, hvorledes der findes en operation (matrixinversion), som for en enkelt ubekendt degenererer til en division. Løsning af lineære ligninger er derfor en simpel udvidelse af de elementære regningsarter, og da man i mange af de mere komplicerede numeriske metoder bruger løsning af lineære ligninger som et standardhjælpemiddel undervejs, er det rimeligt at anbringe dette som det første punkt i vor klassifikation.

Metoderne i kapitel 3-5 kan alle klassificeres under en fælles overskrift: funktionsanalyse. Vi interesserer os her for funktioner af een variabel:

$$y = f(x)$$

eller af flere variable:

$$z = f(x1, x2, x3)$$

Af alle funktioner er vi særligt interesseret i dem, som kan skrives som polynomier, d.v.s. de kan beregnes udelukkende ved addition og multiplikation:

$$y := a0 + x \times (a1 + x \times (a2 + x \times a3));$$

Dette er et tredjegradspolynomium i x. Da polynomier er særligt lette at undersøge med henblik paa nulpunkter og maksimumspunkter, har vi brug for metoder, som til en opgivet tabel over en funktion beregner et tilsvarende polynomium. Dette beskrives i kapitel 3.

I kapitel 4 diskuterer vi metoder til undersøgelse af, hvornaar en funktion bliver nul eller antager en anden kendt værdi. Det kan være en funktion af een variabel:

$$f(x) = 0$$

eller flere funktioner af flere variable:

$$f1(x1, x2, x3) = 0$$
$$f2(x1, x2, x3) = 0$$
$$f3(x1, x2, x3) = 0$$

hvor vi altsaa skal finde et talsæt: x1, x2, x3, saaledes at alle 3 funktioner bliver nul.

I kapitel 5 omtales metoder til bestemmelse af, hvornaar en foreliggende funktion antager sin maksimumværdi. Dette problem er nært beslægtet med det foregaaende, idet man normalt har, at funktionens differentialkvotienter skal være nul.

Af pladsmæssige grunde er beregning af bestemte integraler og løsning af differentialligninger henlagt til bind 3 i denne serie.

Egentlige databehandlingsmetoder som sortering o. lign. omtales ikke i denne bog. Vi har enkelte procedurer hertil, og de vil eventuelt senere blive omtalt andetsteds.

2. LØSNING AF LINEÆRE LIGNINGER

2.1. Lineære Ligninger og Matrixinversion.

Som indledning til dette kapitel giver vi først et meget elementært eksempel paa et problem, der kan formuleres som et sæt lineære ligninger.

Vi foretager tre indkøb hos en købmand. Først køber vi 3 appelsiner, 4 bananer og 5 citroner og betaler ialt herfor 4.19 kr. Derefter køber vi 6 appelsiner, 2 bananer og 3 citroner til en samlet pris af 4.13 kr. Og endelig køber vi 1 appelsin, 7 bananer og 4 citroner til 4.17 kr. Opgaven er nu at beregne prisen paa 1 appelsin, paa 1 banan og paa 1 citron. Kalder vi de ukendte priser for A, B og C, kan vi formulere det som tre ligninger:

$$3\times A + 4\times B + 5\times C = 4.19$$
$$6\times A + 2\times B + 3\times C = 4.13$$
$$1\times A + 7\times B + 4\times C = 4.17$$

Vi kan ogsaa skrive de tre ubekendte som x[1], x[2] og x[3], altsaa et talsæt: array x[1:3]. Ligeledes kan selve talkoefficienterne og priserne paa højre side skrives som et andet talsæt: array a[1:3, 1:4]. De tre ligninger bliver da:

$$a[1,1]\times x[1] + a[1,2]\times x[2] + a[1,3]\times x[3] = a[1,4]$$
$$a[2,1]\times x[1] + a[2,2]\times x[2] + a[2,3]\times x[3] = a[2,4]$$
$$a[3,1]\times x[1] + a[3,2]\times x[2] + a[3,3]\times x[3] = a[3,4]$$

Lad os nu antage, at vi foretager denne indkøbsserie hos mange forskellige købmænd, men altid med de samme tre kombinationer, altsaa først 3 appelsiner, 4 bananer og 5 citroner, o.s.v. Vi ser da, at talsættet a[1:3, 1:3] vil være det samme for alle indkøbssæt, medens højresiderne sandsynligvis vil variere fra den ene købmand til den næste. Det er da mere praktisk at operere med de to talsæt A[1:3, 1:3] og B[1:3]. Nu kan ligningssystemet skrives saaledes:

$$A[1,1]\times x[1] + A[1,2]\times x[2] + A[1,3]\times x[3] = B[1]$$
$$A[2,1]\times x[1] + A[2,2]\times x[2] + A[2,3]\times x[3] = B[2]$$
$$A[3,1]\times x[1] + A[3,2]\times x[2] + A[3,3]\times x[3] = B[3]$$

Her skal ligningerne løses for et bestemt talsæt A og forskellige værdier af elementerne i talsættet B.

Talsættet A kaldes en matrix, og man kan vise, at det normalt er muligt at omregne den givne matrix A til en anden matrix, C[1:3, 1:3], der betegnes som den inverterede matrix svarende til A, og saaledes at løsningerne nu kan skrives som:

$$x[1] := C[1,1]\times B[1] + C[1,2]\times B[2] + C[1,3]\times B[3]$$
$$x[2] := C[2,1]\times B[1] + C[2,2]\times B[2] + C[2,3]\times B[3]$$
$$x[3] := C[3,1]\times B[1] + C[3,2]\times B[2] + C[3,3]\times B[3]$$

Vi ser, at elementet C[r, s] er et maal for, hvor stor en del af løsningen x[r], som hidrører fra B[s]. Man siger iøvrigt, at vektoren x[1:3] fremkommer ved matrixmultiplikation af matricen C[1:3, 1:3] med den anden vektor B[1:3].

Problemet kan altsaa formuleres paa to maader: enten en direkte behandling af talsættet a[1:3, 1:4] eller en invertering af A til C med paafølgende multiplikation med B. Da invertering af en matrix ogsaa kan bruges til andre ting end netop løsning af linesre ligninger, vil det være mest praktisk at arbejde med invertering som en standard-procedure. I de næste to sektioner viser vi dels en procedure til direkte løsning af et ligningssystem (LINEQ1) og dels en procedure til invertering (INVERT2).

2.2. Gauss-Elimination med Delvis Pivotering.

Vi gennemgaar først taleksemplet ovenfor med de tre ligninger:

(1) $3\times A + 4\times B + 5\times C = 4.19$

(2) $6\times A + 2\times B + 3\times C = 4.13$

(3) $1\times A + 7\times B + 4\times C = 4.17$

Vi har altid lov til at gange en ligning med en vilkaarlig faktor ($\neq 0$) og f.eks. addere resultatet til een af de andre ligninger. Princippet i Gauss-eliminationsmetoden er nu det, at vi vil omdanne den oprindelige matrix:

$$\begin{matrix} 3 & 4 & 5 \\ 6 & 2 & 3 \\ 1 & 7 & 4 \end{matrix}$$

til en anden matrix med nuller under diagonalen:

$$\begin{matrix} 3 & 4 & 5 \\ 0 & z & u \\ 0 & 0 & v \end{matrix}$$

Her er z, u og v nye elementer. Først skal der skaffes et nul paa 6-tallets plads. Det sker ved at addere -2×(ligning 1) til ligning 2:

$$\begin{matrix} 3 & 4 & 5 & 4.19 \\ 0 & -6 & -7 & -4.25 \\ 1 & 7 & 4 & 4.17 \end{matrix}$$

Derefter adderer vi -1/3×(ligning 1) til ligning 3 og faar:

$$\begin{matrix} 3 & 4 & 5 & 4.19 \\ 0 & -6 & -7 & -4.25 \\ 0 & 5.6667 & 2.3333 & 2.7733 \end{matrix}$$

hvorved der er skaffet nul paa 1-tallets plads. Den sidste operation bliver at faa nul i stedet for 5.6667, og det sker ved at addere 5.6667/6× (ligning 2) til ligning 3:

$$\begin{matrix} 3 & 4 & 5 & 4.19 \\ 0 & -6 & -7 & -4.25 \\ 0 & 0 & -4.2778 & -1.2406 \end{matrix}$$

Skrevet helt ud er ligningerne nu:

(1) $3 \times A + 4 \times B + 5 \times C = 4.19$

(2) $-6 \times B - 7 \times C = -4.25$

(3) $-4.2778 \times C = -1.2406$

Her kan vi straks finde C af ligning 3 til C = 0.29. Dette indsættes i ligning 2 og vi finder B = 0.37. Endelig indsættes B og C i ligning 1 og vi finder A = 0.42.

Vi kan nu diskutere selve ALGOL-proceduren LINEQ1, der løser et sæt ligninger ved Gauss-elimination. Proceduredeklarationen er:

```
procedure  LINEQ1 (N, a, x, NOSOLUTION);
integer  N;
array  a, x;
label  NOSOLUTION;
begin
   integer p, i, j;
   real  M;
   for  p := 1 step 1 until N - 1 do
   begin
      for  i := p + 1 step 1 until N do
      begin
         if  a[p,p] ≠ 0 then go to L2;
         if  a[i,p] ≠ 0 then go to L1;
         if  i < N then go to L3;
         go to NOSOLUTION;
L1:      for j := p step 1 until N + 1 do
         begin
            M := a[p,j];
            a[p,j] := a[i,j];
            a[i,j] := M
         end of row exchange;
         go to L3;
L2:      if a[i,p] = 0 then go to L3;
         M := -a[i,p]/a[p,p];
         for j := p+1 step 1 until N+1 do
         a[i,j] := a[i,j] + M×a[p,j];
L3:   end for i;
   end for p;
```

```
        if a[N,N] = 0 then go to NOSOLUTION;
        for p := N step -1 until 1 do
        begin
           x[p] := a[p,N+1] := a[p,N+1]/a[p,p];
           if p = 1 then go to L4;
           for i := p-1 step -1 until 1 do
           a[i,N+1] := a[i,N+1] - x[p]×a[i,p]
        end for second p;
L4: end LINEQ-1;
```

Proceduren har de fire parametre:

N:	Antallet af ligninger og ubekendte.
a:	Talmaterialet: array a[1:N, 1:N+1].
x:	Den søgte løsning: array x[1:N].
NOSOLUTION:	En etikette, hvortil der hoppes, hvis løsningen ikke kan findes. Dette forklares nærmere nedenfor.

Proceduren bestaar af de to dele: nulstilling under diagonalen og den baglæns substitution. Nulstillingen sker med for-sætningen:

```
for p := 1 step 1 until N-1 do
```

som behandler ligning 1, 2, ..., N-1. For hvert gennemløb vil for-sætningen:

```
for i := p+1 step 1 until N do
```

behandle de nedenunderstaaende ligninger, idet den beregner faktoren

```
M := -a[i,p]/a[p,p];
```

og adderer M×ligning nr. p til ligning nr. i med for-sætningen:

```
for j := p+1 step 1 until N+1 do
a[i,j] := a[i,j] + M×a[p,j];
```

Den baglæns substitution sker med for-sætningen:

```
for p := N step -1 until 1 do
```

Hvis der under nulstillingen opstaar den situation, at et diagonalelement er nul, vil proceduren foretage en ombytning af to rækker, saaledes at dette nul kommer væk fra diagonalen. Betegnelsen delvis pivotering dækker denne rækkeombytning. Den er nødvendig, fordi faktoren M ikke kan beregnes, hvis diagonalelementet er nul (a[p,p] = 0).

Hvis to af ligningerne er helt ens eller kun adskiller sig ved en konstant faktor, er der ikke ligninger nok til at bestemme alle N ubekendte. Under nulstillingen vil dette vise sig paa den maade, at vi ikke kan faa fjernet et nul-element fra diagonalen. Proceduren foretager da et fejludhop til etiketten: NOSOLUTION, som tegn paa, at ligningssystemet ikke kan løses.

Her er et primitivt eksempel paa et program indeholdende proceduren LINEQ1. Det læser fra en inputstrimmel værdien af N, samt talsættet a[1:N, 1:N+1] og trykker blot resultatet: x[1:N].

```
begin comment løsning af lineære ligninger;
   integer N, k;
   comment library LINEQ1;
   læs(N);
   begin
      array x[1:N], a[1:N, 1:N+1];
      læs(a);
      LINEQ1(N, a, x, ERROR);
      for k := 1 step 1 until N do
      begin
         trykvr;
         tryk({-n.ddddd10-dd}, x[k])
      end for k;
      go to EX;
ERROR: skrvvr;
      skrvtekst({<ERROR}) ;
EX: end indre blok
end program;
```

Bemærk, hvorledes N indlæses i den ydre blok, medens talsættene x og a, hvis grænser afhænger af N, deklareres i den indre blok, efter indlæsning af N.

Proceduren LINEQ1 virker i de fleste tilfælde tilfredsstillende, men ikke altid. Hvis et diagonalelement ikke er nul, men meget lille, sker der ingen rækkeombytning. Faktoren M bliver meget stor, og naar ligning p adderes til ligning no. i med:

```
a[i,j] := a[i,j] + M×a[p,j];
```

er de oprindelige i-elementer forsvindende i sammenligning med de nye elementer med M-faktoren, og man faar daarlig regnenøjagtighed. Som eksempel giver vi de fire ligninger:

$1_{10}-6$	1	1	1	3.000001
1	2	1	1	5
1	1	2	1	5
1	1	1	2	5

hvor alle de ubekendte har den eksakte løsning: 1. Med LINEQ1 finder vi:

```
x[1] = 0.9984
x[2] = 1.0026
x[3] = x[4] = 0.9987
```

Lignende fejl opstaar, hvis man har et ligningssystem som ovenstaaende, men med et stort element i den øverste trekant af matricen (over diagonalen). Det anbefales derfor ikke at bruge proceduren LINEQ1, men i stedet inverteringsproceduren i det følgende afsnit.

2.3. Matrixinversion med Fuld Pivotering.

Proceduren INVERT2 beregner den inverterede matrix svarende til en opgivet matrix: array a[1:n, 1:n]. I princippet anvendes samme eliminationsmetode som i LINEQ1, men proceduren vil nu foretage passende ombytninger af rækker og søjler i matricen, saaledes at vi hele tiden faar det størst mulige element paa diagonalen. Der bruges to lokale talsæt:

```
integer array p, q[1:n];
```

til at holde regnskab med hvilke rækker og søjler, der er ombyttet. Naar proceduren er færdig, er matricen: a erstattet af den inverterede matrix. Den oprindelige matrix er gaaet tabt. De ombyttede rækker og søjler er byttet tilbage igen.

Foruden de to parametre: n og a er der to andre parametre:

```
real eps;
label ERROR;
```

Hvis et diagonalelement bliver mindre end eller lig med eps, selv efter at det størst mulige er anbragt paa dets plads, vil proceduren gaa til etiketten: ERROR. En passende værdi for eps kan f.eks. være $1_{10}-8$, men vil afhænge af størrelsesordenen af matricens elementer. Ønsker man ikke at udnytte denne fejludhopsmulighed, kan man sætte eps = 0, og faar da normalt ikke fejludhop. Men er der fare for at ligningerne slet ingen løsning har (hvis to af ligningerne er ens eller proportionale), bør fejludhoppet udnyttes, da resultatet ellers vil være helt forkert. Det er ogsaa vigtigt paa denne maade at kunne fange tilfælde, hvor der er lige ved ikke at være nogen løsning, altsaa hvis to af ligningerne er næsten ens eller næsten proportionale. Man siger, at matricen eller ligningssystemet er daarligt bestemt (ill-conditioned).

Deklarationen for INVERT2 er:

```
procedure INVERT2(n, a, eps, ERROR);
value n, eps;
integer n;
real eps;
array a;
label ERROR;
begin
   integer i, j, k;
   real pivot, z;
   integer array p, q[1:n];
   array b, c[1:n];
   for k := 1 step 1 until n do
   begin
      pivot := 0;
      for i := k step 1 until n do
      for j := k step 1 until n do
      if abs(a[i,j]) > abs(pivot) then
      begin
         pivot := a[i,j];
         p[k] := i;
         q[k] := j
      end;
      if abs(pivot) ≤ eps then go to ERROR;
      if p[k] ≠ k then
      for j := 1 step 1 until n do
      begin
         z := a[p[k], j];
         a[p[k], j] := a[k,j];
         a[k,j] := z
      end for j;
      if q[k] ≠ k then
      for i := 1 step 1 until n do
      begin
         z := a[i, q[k]];
         a[i, q[k]] := a[i,k];
         a[i,k] := z
      end for i;
```

```
      for j := 1 step 1 until n do
      begin
         if j = k then
         begin
            b[j] := 1/pivot;
            c[j] := 1
         end
         else
         begin
            b[j] := - a[k,j]/pivot;
            c[j] := a[j,k]
         end;
         a[k,j] := a[j,k] := 0
      end for j;
      for i := 1 step 1 until n do
      for j := 1 step 1 until n do
      a[i,j] := a[i,j] + c[i]×b[j]
   end for k;
   for k := n step -1 until 1 do
   begin
      if p[k] ≠ k then
      for i := 1 step 1 until n do
      begin
         z := a[i, p[k]];
         a[i, p[k]] := a[i,k];
         a[i,k] := z
      end;
      if q[k] ≠ k then
      for j := 1 step 1 until n do
      begin
         z := a[q[k], j];
         a[q[k], j] := a[k,j];
         a[k,j] := z
      end j
   end k
end INVERT2;
```

Proceduren INVERT2 er oprindeligt lavet af H. Rutishauser og offentliggjort af Schwarz (1962). Afprøvningsresultater er offentliggjort af Naur (1963). Vi har ikke underkastet den nogen yderligere analyse.

Vi kan omskrive det lille program paa side 16, saa det indeholder INVERT2 istedet for LINEQ1:

```
begin comment løsning af lineære ligninger;
   integer N, k, m;
   comment library INVERT2;
   læs(N);
   begin
      array x, b[1:N], a[1:N, 1:N];
      læs(a);
      INVERT2(N, a, 1_10-8, ERROR);
      læs(b);
      for k := 1 step 1 until N do
      begin
         x[k] := 0;
         for m := 1 step 1 until N do
         x[k] := x[k] + a[k,m]×b[m];
         trykvr;
         tryk({-n.ddddd_10-dd}, x[k])
      end for k;
      go to EX;
ERROR: skrvvr;
      skrvtekst({<ERROR});
EX: end indre blok
end program;
```

Som før indlæser vi først N, men derefter kun den kvadratiske matrix, som inverteres. Endelig indlæses højresiden: b[1:N]. Vi bruger ingen speciel procedure til at udføre matrixmultiplikationen efter inverteringen. Det er jo blot:

```
for m := 1 step 1 until N do
x[k] := x[k] + a[k,m]×b[m];
```

hvor x[k] skal nulstilles først. Værdien af k styres af den ydre forsætning, som vi ogsaa lader styre trykningen af x[k].

Proceduren LINEQ1 kan ikke bruges for N = 1, men det kan INVERT2 godt. Inverteringen bestaar da blot af at det ene element i matricen erstattes af den reciprokke værdi af elementet. Inversionen er altsaa degenereret til en division.

2.4. Tromlelagring.

Med proceduren INVERT2 kan man i GIER invertere en matrix a[1:N, 1:N] hvor N ikke overstiger ca. 25. Det nøjagtige tal afhænger af hvormange andre variable, der skal være plads til i lageret samtidigt.

For større matricer maa man bruge tromlelagring. Hertil bruger vi proceduren INVERT3, som inverterer en matrix lagret paa tromlen, men iøvrigt virker paa samme maade som INVERT2. Den tidligere parameter: a af typen array falder bort og erstattes af de to integer parametre: aplace og astep. Matricen skal være lagret rækkevis paa tromlen, saaledes at tromleplads for række nr. r er:

aplace + r×astep

Rækkerne opfattes altsaa som enkelte arrays af dimensionen [1:n]. Normalt vil man have, at n = astep. Men man kan ogsaa bruge astep > n. I saa fald maa rækkeelementerne være placeret i begyndelsen af de faktisk anvendte arrays: [1:astep].

Deklarationen af INVERT3 er:

```
procedure INVERT3(n, aplace, astep, eps, ERROR);
value n, aplace, astep, eps;
integer n, aplace, astep;
real eps;
label ERROR;
begin
   integer i, j, k, ii, pk, qk;
   real pivot;
```

```
integer array p, q[1:astep];
array b, c, R, S[1:astep], EL[1:1];
procedure ROW(r, T, from);
value r, from;
integer r;
boolean from;
array T;
begin
   tromleplads := aplace + r×astep;
   if from then fratromle (T) else tiltromle (T)
end ROW;
procedure COL(c, T, from);
value c, from;
integer c;
boolean from;
array T;
for ii := 1 step 1 until n do
begin
   tromleplads := aplace + (ii-1)×astep + c;
   if from then
   begin
      fratromle(EL);
      T[ii] := EL[1]
   end
   else
   begin
      EL[1] := T[ii];
      tiltromle(EL)
   end
end COL;
procedure EXROW(r1, r2);
value r1, r2;
integer r1, r2;
begin
   ROW(r1, R, true);
   ROW(r2, S, true);
   ROW(r2, R, false);
```

```
      ROW(r1, S, false)
end EXROW;
procedure EXCOL(c1, c2);
value c1, c2;
integer c1, c2;
begin
      COL(c1, R, true);
      COL(c2, S, true);
      COL(c2, R, false);
      COL(c1, S, false)
end EXCOL;
for k := 1 step 1 until n do
begin
      pivot := 0;
      for i := k step 1 until n do
      begin
            ROW(i, R, true);
            for j := k step 1 until n do
            if abs(R[j]) > abs(pivot) then
            begin
                  pivot := R[j];
                  p[k] := i;
                  q[k] := j
            end if
      end for i;
      if abs(pivot) ≤ eps then go to ERROR;
      pk := p[k];
      if pk ≠ k then EXROW (k, pk);
      qk := q[k];
      if qk ≠ k then EXCOL (k, qk);
      for j := 1 step 1 until n do
      begin
            ROW(k, R, true);
            if j = k then
            begin
                  b[j] := 1/pivot;
                  c[j] := 1
```

```
            end
            else
            begin
               b[j] := - R[j]/pivot;
               tromleplads := aplace + (j-1)×astep + k;
               fratromle(EL);
               c[j] := EL[1]
            end if not;
            R[j] := EL[1] := 0;
            tromleplads := aplace + (j-1)×astep + k;
            tiltromle(EL);
            ROW(k, R, false)
         end for j;
         for i := 1 step 1 until n do
         begin
            ROW(i, R, true);
            for j := 1 step 1 until n do
            R[j] := R[j] + c[i]×b[j];
            ROW(i, R, false)
         end for i
      end for k;
      for k := n step -1 until 1 do
      begin
         pk := p[k];
         if pk ≠ k then EXCOL(k, pk);
         qk := q[k];
         if qk ≠ k then EXROW(k, qk)
      end for k
   end INVERT3;
```

2.5. Diskussion af Program LINEAR EQUATIONS-4.

Vi har et specielt program, LINEAR EQUATIONS-4, som løser N lineære ligninger med N ubekendte. For N < 25 bruges INVERT2 og for N ≥ 25 INVERT3.

I appendikset viser vi inputspecifikationerne til dette program, samt en typisk resultatrapport og selve ALGOL-programmet.

Der er fem datagrupper i inputmaterialet. Gruppe 0 og 1 er forside og overskriftsdata, der er ens for næsten alle vore programmer.

Gruppe 2 indeholder 6 beregningsparametre. N er antallet af ubekendte eller matricens dimension: [1:N, 1:N] og NRS er antallet af højresider, som skal behandles sammen med den samme koefficientmatrix. Hvis NRS = 0, ønsker man blot matricen inverteret, men ingen løsning af lineære ligninger. Endvidere er der en parameter til at bestemme, om den indlæste matrix skal gemmes uforandret til næste sektion eller vi kan lagre den inverterede matrix oveni den originale. Det sidste er nødvendigt for N > 62, da der ellers ikke er plads til begge matricer. To andre parametre bruges til styring af trykning af den originale og den inverterede matrix.

Programmet indeholder en indbygget værdi: eps = 1_{10}-12, som den mindste tilladelige værdi af et diagonalelement under inverteringen. Bliver et diagonalelement mindre end eps, faar man fejludhop fra INVERT2 eller INVERT3 og inversionen kan ikke gennemføres. Resultatrapporten vil da indeholde bemærkningen:

ERROR IN MATRIX INVERSION

og man maa da gentage beregningen med en lavere værdi af eps. Det er det sidste inputtal i datagruppe 2.

Gruppe 3 er matricen: a[1:N, 1:N], og højresiden specificeres som gruppe 4.

Den højeste værdi af N med en enkelt højreside er N = 88. Er N for stor, vil resultatrapporten indeholde bemærkningen:

Too many equations

og der fortsættes straks med næste beregningssektion.

Det viste beregningseksempel bestaar af løsning af 10 ligninger med 10 ubekendte. Matricens elementer er valgt tilfældigt, og højresiden er tilpasset saaledes at alle de ubekendte er nøjagtigt 1. Den største fejl i løsningerne er ca. 5_{10}-5. Et lignende eksempel med N = 88 (maksimum af programmets kapacitet) findes som GIER-beregning nr. 7934. Rapporten fylder 53 sider og beregningen tog ca. 4 timer.

Det fremgaar af det viste ALGOL-program, at den yderste blok i programmet indeholder 5 sideordnede indre blokke. Disse bruges til:

1. Forside og overskrift
2. Indlæsning og trykning af den originale matrix
3. Inversion med INVERT2
4. Inversion med INVERT3
5. Trykning af den inverterede matrix og løsningen
6. Trykning af eps.

Vi knytter et par kommentarer til ALGOL-programmet.

Brugen af stop-e og indlæsning af datagrupper til bestemte tromlekanaler kommer ikke rigtigt til sin ret i dette program, da matricen og højre-siderne meget let kan fylde mere end een kanal. Det har derfor været nødvendigt enkelte steder at anbringe en besynderlig sætning:

```
for j := læstegn while tegn ≠ 53 do;
```

som blot læser videre paa strimlen til det næste stop-e er mødt.

Matrixelementer eller vektorelementer, som er eksakt nul, trykkes uden decimaler.

3. APPROKSIMATION MED POLYNOMIER

3.1. Problemstilling.

For en begynder i programmering vil den hyppige anvendelse af polynomier i regnemaskineprogrammer kræve en nærmere forklaring.

I bind 1, side 35, saa vi et eksempel paa beregning af entalpi ved hjælp af et fjerdegradspolynomium i temperaturen: T:

```
H := a0 + T×(a1 + T×(a2 + T×(a3 + T×a4)));
```

Entalpien, H, er matematisk set en funktion af den uafhængige variable, T, og da polynomieudtrykket kun indeholder regneoperationerne addition og multiplikation, er det meget hurtigt at beregne talværdien af polynomiet for en kendt værdi af T.

Hvis man af teoretiske grunde havde vidst, at entalpien f.eks. adlød følgende lovmæssighed:

```
H := a×exp(T) + b×sin(T↑2);
```

ville det have været mere naturligt at beregne H efter dette udtryk. Hertil maa dog straks bemærkes to vigtige forhold. For det første vil regnemaskinen altid beregne værdien af exp(T) og sin(T) og lignende funktioner ved hjælp af et lille polynomium i T med nogle faa koefficienter. For det andet kan man saa godt som altid omskrive et kompliceret udtryk indeholdende standardfunktioner til en rækkeudvikling i den uafhængige variable, f.eks. ved hjælp af Taylors formel.

Vi kan give følgende almindelige regler for, hvorledes man fremskaffer et funktionsudtryk, som skal indgaa som led i et regnemaskineprogram.

1. Hvis man ad teoretisk vej kender et funktionsudtryk, kan man bruge det umiddelbart, som det er.

2. Hvis man som den modsatte yderlighed absolut intet aner om funktionsafhængigheden, vil det normalt være nødvendigt at anstille forsøg for at bestemme udtrykket.

3. Vi vil ikke diskutere forsøgsplanlægning i almindelighed i denne

bog, men blot bemærke, at der eksisterer veludviklede statistiske metoder hertil. Her vil man ofte begynde med at finde ud af, hvilke uafhængige variable, der skal indgaa i funktionsudtrykket. Dette kan i princippet udføres ved at lave en forsøgsrække, hvor hver af de mistænkte variable gives to værdier, en høj og en lav, og den statistiske analyse vil da fortælle, om der er en signifikant afhængighed. Analysen kan let udvides til ogsaa at inkludere højere potenser af de uafhængige variable, idet disse simpelthen kan opfattes som nye uafhængige variable.

4. Naar den undersøgte funktion foreligger som en tabel, der giver sammenhørende værdier af den eller de uafhængige variable og funktionen, kan man benytte standardmetoder til bestemmelse af det bedst mulige polynomium, som reproducerer tabelværdierne. Disse omtales i de følgende afsnit. Vi skal ogsaa se eksempler paa, hvorledes man sommetider bør transformere de variable, inden polynomiet beregnes.

3.2. Mindste Kvadraters Metode.

Vi skal nu undersøge, hvilket kriterium man kan opstille, for at et beregnet polynomium passer saa godt som muligt til en foreliggende funktionsafhængighed.

Lad os antage, at der er opgivet nedenstaaende tabel over x og y.

x	y
300	2413
400	3323
500	4365
600	5549
700	6871
800	8321
900	9887
1000	11560
1100	13320
1200	15170
1300	17100
1400	19090
1500	21130

Her er x temperaturen i grader Kelvin og y entalpien i kcal/kgmol af metan.

Ved de mindste kvadraters metode forlanger man, at summen af kvadraterne paa fejlene i de 13 punkter skal være minimum. Fejlen i et punkt er differensen mellem det fundne polynomiums værdi i punktet og den opgivne y-værdi i punktet. Hvis vi dividerer kvadratsummen med antallet af punkter (minus 1) og tager kvadratroden, faar vi middelfejlen i punkterne, og man kan derfor sige, at de mindste kvadraters metode giver minimum af middelfejlen.

Der findes ogsaa et andet muligt kriterium: minimum af den største fejl. Dette har navnlig interesse, hvis man ønsker at gengive en analytisk funktion, f.eks. exp(x), indenfor et givet omraade. Her kendes funktionsværdien eksakt i alle punkter, og man kan ønske en polynomietilnærmelse, hvor den største fejl er saa lille som mulig, eller mindre end en opgivet tolerance. Dette kriterium giver anledning til brug af de saakaldte Chebyshevske polynomier. Ved reproduktion af funktioner, der er givet som tabeller, er det næppe rimeligt at minimalisere maksimumsfejlen i de tilfældigvis kendte punkter, da der jo udmærket godt kan være større absolutte fejl udenfor de opgivne punkter. De approksimationsprocedurer, der omtales her, er udelukkende baseret paa de mindste kvadraters metode, og de Chebyshevske polynomier har hidtil ikke været brugt ved GIER-installationen hos Haldor Topsøe.

3.2.1. Lineære funktioner af een variabel. Som et simpelt eksempel paa de mindste kvadraters metode giver vi nu proceduren: FIT1, som finder en lineær tilnærmelse (et førstegradspolynomium) til funktionen y = f(x). Det tilnærmede udtryk skrives simpelthen:

```
y := a + b×x;
```

Deklarationen af FIT1 er:

```
procedure FIT1(n, meanerror, a, b, x, y);
value n;
integer n;
real meanerror, a, b;
array x, y;
begin
   integer j;
   real SX, SX2, SY, SXY, SY2, DEN;
   SX := SX2 := SY := SXY := SY2 := 0;
   for j := 1 step 1 until n do
   begin
      SX := SX + x[j];
      SX2 := SX2 + x[j]↑2;
      SY := SY + y[j];
      SXY := SXY + x[j]×y[j];
      SY2 := SY2 + y[j]↑2
   end;
   DEN := n×SX2 - SX↑2;
   a := (SX2×SY-SX×SXY)/DEN;
   b := (n×SXY-SX×SY)/DEN;
   meanerror := sqrt((SY2+(2×SX×SY×SXY-n×SXY↑2-SX2×SY↑2)/DEN)/(n-1))
end of FIT-1;
```

Proceduren behandler de to talsæt x, y[1:n], altsaa en tabel over x og y med n værdier, f.eks. som tabellen side 30. Den beregner de to koefficienter, a og b, samt middelfejlen, meanerror. Beregningsmetoden kan forklares saaledes:

Fejlen i punkt nr. i er:

```
a + b×x[i] - y[i]
```

og fejlens kvadrat:

```
(a + b×x[i] - y[i])↑2
```

Summen af fejlkvadraterne bliver:

```
SFK := (a + b×x[1] - y[1])↑2
     + (a + b×x[2] - y[2])↑2
     + ......
       ......
     + (a + b×x[n] - y[n])↑2
```

Naar talsættet x og y er givet, er SFK en funktion af de to variable a og b. Betingelsen for at SFK bliver minimum er, at de to partielle differentialkvotienter af SFK med hensyn til a og b bliver nul. Vi beregner differentialkvotienterne:

```
dSFK/da = 2×(a + b×x[1] - y[1])×1
        + 2×(a + b×x[2] - y[2])×1
        + ......
          ......
        + 2×(a + b×x[n] - y[n])×1

dSFK/db = 2×(a + b×x[1] - y[1])×x[1]
        + 2×(a + b×x[2] - y[2])×x[2]
        + ......
          ......
        + 2×(a + b×x[n] - y[n])×x[n]
```

Vi sætter nu de to differentialkvotienter til nul, dividerer 2-tallerne væk, og summerer. Vi indfører ogsaa sumbetegnelserne:

```
SX  := x[1] + x[2] + ..............  + x[n];
SX2 := x[1]↑2 + x[2]↑2 + ..........  + x[n]↑2;
SY  := y[1] + y[2] + ..............  + y[n];
SXY := x[1]×y[1] + x[2]×y[2] + ....  + x[n]×y[n];
```

De to nulbetingelser bliver da:

$$n \times a + b \times SX - SY = 0$$
$$a \times SX + b \times SX2 - SXY = 0$$

Heraf findes a og b til:

```
a := (SX2×SY - SX×SXY)/DEN;
b := (n×SXY - SX×SY)/DEN;
```

hvor nævneren, DEN, er:

```
DEN := n×SX2 - (SX)↑2;
```

Disse formler er brugt i proceduren, der i det væsentlige bestaar af en for-sætning til beregning af summerne og den afsluttende beregning af a og b. Vi ser, at proceduren yderligere indeholder en summation af kvadraterne paa y-værdierne, samt at man paa basis heraf kan beregne middelfejlen. Formlen herfor vil dog ikke blive forklaret nærmere.

Et eksempel paa brugen af FIT1 er vist nedenfor i program d-174. Programmet læser x-y-tabellen side 30 og finder tilnærmelsen:

```
y := - 3573.1 + 15.773×x;
```

som naturligvis er langt fra at være tilstrækkelig nøjagtig til en normal anvendelse.

Programmet er:

```
Program d-174. Brug af FIT1.
begin
   integer i;
   real a, b, meanerror, yber;
   array x, y[1:13];
   comment library FIT1;
   for i := 1 step 1 until 13 do læs(x[i], y[i]);
   FIT1(13, meanerror, a, b, x, y);
```

```
    trykvr;
    tryktekst({<      x  y,inp.  y,ber. Fejl});
    trykvr;
    for i := 1 step 1 until 13 do
    begin
       trykvr;
       yber := a + b×x[i];
       tryk({-nddddd}, x[i], y[i], yber, yber - y[i])
    end for i;
    trykvr;
    trykvr;
    tryk({-nddd.d000}, tryktekst({<a:  }), a,
    tryktekst({<  b:  }), b,
    tryktekst({<  middelfejl:  }), meanerror);
    trykvr
end program;
```

Resultatudskriften er:

x	y,inp.	y,ber.	Fejl
300	2413	1159	-1254
400	3323	2736	-587
500	4365	4314	-51
600	5549	5891	342
700	6871	7468	597
800	8321	9046	725
900	9887	10623	736
1000	11560	12200	640
1100	13320	13778	458
1200	15170	15355	185
1300	17100	16932	-168
1400	19090	18510	-580
1500	21130	20087	-1043

a: -3573.1 b: 15.773 middelfejl: 681.08

3.2.2. Lineære funktioner af flere variable. Proceduren FIT1 kan let udvides, saaledes at den kan finde en lineær tilnærmelse til en funktion af flere variable. En funktion af to variable kan vi skrive saaledes:

```
y := y0 + x1×c1 + x2×c2;
```

De to uafhængige variable er x1 og x2, og funktionen er y. Som et eksempel tager vi materialet fra følgende tabel:

y	x1	x2
43.92	410	280
38.18	440	300
33.20	440	240
30.57	460	260
39.68	420	260
38.95	430	280

Her er y ammoniakligevægtsprocenten som funktion af temperaturen, x1, og trykket, x2. De seks værdier er plukket ud af tabellen side 16 i Kjær (1963b).

Proceduren FIT2 kan bruges til at finde en lineær tilnærmelse til en funktion af een eller flere variable. Deklarationen er:

```
procedure FIT2(var, obs, x, y, xmean, ymean, ycalc, coef, coeferror,
          meanerror, tol, ERROR);
value var, obs, tol;
integer var, obs;
real ymean, meanerror, tol;
array x, y, xmean, ycalc, coef, coeferror;
label ERROR;
begin
   integer i, j, k;
   real ym, mean, xj, c, yc;
   array SXX[1:var, 1:var], SXY[1:var];
   ym := mean := 0;
   for j := 1 step 1 until var do xmean[j] := 0;
```

```
for i := 1 step 1 until obs do
begin
   ym := ym + y[i]/obs;
   for j := 1 step 1 until var do
   xmean[j] := xmean[j] + x[i, j]/obs
end for i;
for j := 1 step 1 until var do
begin
   SXY[j] := 0;
   for k := 1 step 1 until var do SXX[j, k] := 0
end for j;
for i := 1 step 1 until obs do
for j := 1 step 1 until var do
begin
   xj := x[i, j] - xmean[j];
   SXY[j] := SXY[j] + xj×(y[i] - ym);
   for k := 1 step 1 until var do
   SXX[j, k] := SXX[j, k] + xj×(x[i, k] - xmean[k])
end for j;
INVERT2(var, SXX, tol, ERROR);
for j := 1 step 1 until var do
begin
   c := 0;
   for k := 1 step 1 until var do
   c := c + SXX[j, k]×SXY[k];
   coef[j] := c
end for j;
for i := 1 step 1 until obs do
begin
   yc := ym;
   for j := 1 step 1 until var do
   yc := yc + coef[j]×(x[i, j] - xmean[j]);
   ycalc[i] := yc;
   mean := mean + (yc - y[i])↑2
end for i;
mean := sqrt(mean/(obs-var-1));
for j := 1 step 1 until var do
```

```
    coeferror[j] := mean×sqrt(SXX[j, j]);
    ymean := ym;
    meanerror := mean
end FIT2;
```

Proceduren har følgende formelle parametre:

integer var:	Antallet af uafhængige variable.
integer obs:	Antallet af tabelværdier.
array x[1:obs, 1:var]:	Det givne talsæt af uafhængige variable.
array y[1:obs]:	Det tilsvarende sæt funktionsværdier.
array xmean[1:var]:	Middelværdierne af hver af de uafhængige variable. Beregnes af proceduren.
real ymean:	Middelværdien af funktionsværdierne. Beregnes ogsaa af proceduren.
array ycalc[1:obs]:	Dette er de funktionsværdier, som proceduren beregner i hvert punkt ud fra det fundne lineære udtryk.
array coef[1:var]:	De beregnede koefficienter til de uafhængige variable i det lineære udtryk.
array coeferror[1:var]:	Middelfejlen paa coef-værdierne. Beregnes af proceduren.
real meanerror:	Den beregnede middelfejl paa funktionsværdierne.
real tol:	Det mindste tilladelige diagonalelement ved matrixinversionen.
label ERROR:	Fejludhopsetikette, hvis et diagonalelement bliver mindre end tol.

Beregningsmetoden er taget fra Fisher (1948), pag. 156.

Først beregner vi middelværdien af hver af de uafhængige variable, xmean, og middelværdien af y-værdierne, ymean. Den lineære tilnærmelsesfunktion kan da skrives:

```
y := ymean
  + coef[1]×(x1-xmean[1])
  + coef[2]×(x2-xmean[2])
  + o.s.v.
```

Værdien af koefficienterne findes derefter ved løsning af et sæt lineære ligninger, der for var = 3 er:

```
coef[1]*SXX[1,1] + coef[2]*SXX[1,2] + coef[3]*SXX[1,3] = SXY[1]
coef[1]*SXX[2,1] + coef[2]*SXX[2,2] + coef[3]*SXX[2,3] = SXY[2]
coef[1]*SXX[3,1] + coef[2]*SXX[3,2] + coef[3]*SXX[3,3] = SXY[3]
```

I matricen SXX[1:var, 1:var] findes elementet SXX[j, k] ved summation af:

```
(x[i,j] - xmean[j])*(x[i,k] - xmean[k])
```

for i = 1 til i = obs. Medens vi i FIT1 kunne klare os med et enkelt tal (SX2) for summen af x-kvadraterne, maa vi i FIT2 have ialt var↑2 værdier af SXX. I højresiden SXY[1:var] af ligningssystemet fremkommer elementet SXY[j] ved summation af:

```
(x[i,j] - xmean[j])*(y[i] - ymean)
```

for i = 1 til i = obs.

Matricen SXX inverteres med proceduren INVERT2, og de søgte koefficienter findes ved multiplikation af den inverterede matrix med vektoren SXY.

Proceduren finder derefter den beregnede y-værdi, ycalc[i], i hvert punkt ud fra det lineære udtryk og summerer fejlenes kvadrater:

```
meanerror := meanerror + (ycalc[i] - y[i])↑2;
```

Middelfejlen findes derefter ved at dividere med antallet af punkter, eller for at være helt korrekt, antallet af frihedsgrader: obs - var - 1, og kvadratroden uddrages:

```
meanerror := sqrt(meanerror/(obs-var-1));
```

Som forklaret i ovennævnte reference kan man beregne middelfejlen paa de enkelte koefficienter som:

```
coeferror[j] := meanerror×sqrt(SXX[j,j]);
```

hvor SXX[j,j] er diagonalelementet i den inverterede matrix.

Vi giver nu et simpelt eksempel paa brugen af FIT2. Program d-175, som er vist nedenfor, læser tabellen side 35 og finder den lineære tilnærmelse:

```
y := 116.72 - 0.23451×x1 + 0.08263×x2;
```

Programmet er:

```
Program d-175. Brug af FIT2.
begin
   integer i;
   real ymean, meanerror;
   array y, ycalc[1:6], x[1:6, 1:2], xmean, coef, coeferror[1:2];
   comment library FIT2;
   comment library INVERT2;
   for i := 1 step 1 until 6 do
   læs(y[i], x[i, 1], x[i, 2]);
   FIT2(2, 6, x, y, xmean, ymean, ycalc, coef, coeferror, meanerror,
        1_10-12, E);
   trykvr;
   tryktekst({<  y,inp.  y,ber.  Fejl  x1  x2});
   trykvr;
   for i := 1 step 1 until 6 do
   begin
      trykvr;
      tryk({-nddd.dd}, y[i], ycalc[i], ycalc[i] - y[i]);
      tryk({-ndddd}, x[i, 1], x[i, 2])
   end for i;
   trykvr;
   trykvr;
   tryk({-ndd.ddddd}, tryktekst({<y0: }),
   ymean - xmean[1]×coef[1] - xmean[2]×coef[2],
   trykvr,
   tryktekst({<c1: }), coef[1],
```

```
   tryktekst(‹< ±›), coeferror[1],
   trykvr,
   tryktekst(‹<c2: ›), coef[2],
   tryktekst(‹< ±›), coeferror[2]);
   trykvr;
E: end program;
```

Resultatudskriften er:

y,inp.	y,ber.	Fejl	x1	x2
43.92	43.71	-0.21	410	280
38.18	38.33	0.15	440	300
33.20	33.37	0.17	440	240
30.57	30.34	-0.23	460	260
39.68	39.72	0.04	420	260
38.95	39.02	0.07	430	280

y0: 116.72552
c1: -0.23451 ± 0.00599
c2: 0.08263 ± 0.00500

De beregnede middelfejl paa de to koefficienter er væsentlig mindre end selve koefficienterne, og derfor tydeligvis signifikante. Der findes statistiske metoder til at afgøre koefficienternes paalidelighed. Metoden i FIT2 betegnes iøvrigt ofte for en regressionsanalyse, og koefficienterne kaldes regressionskoefficienter.

3.2.3. Diskussion af program PA-6. Proceduren FIT2 er indbygget i program PA-6, som i almindelighed finder en lineær tilnærmelse til et opgivet sæt værdier af en funktion af een eller flere variable. I appendikset vises inputspecifikationerne til PA-6 og en typisk resultatrapport, samt selve ALGOL-programmet.

Af de fire datagrupper er gruppe 0 og 1 som sædvanligt forside og overskriftsdata. Gruppe 2 indeholder antallet af uafhængige variable, antallet af observationer (punkter) samt korrektionstyperne. De sidste kan

bruges til at give oplysning om man vil have transformeret x-værdierne eller y-værdierne. Hvis vi har en funktionsafhængighed:

```
Y := A*x1↑n1*x2↑n2*x3↑n3;
```

beder vi programmet om at tage de naturlige logaritmer af baade x-værdierne og y-værdierne, inden den lineære approximation beregnes. Funktionen bliver da:

$$\ln(Y) = \ln(A) + n1*\ln(x1) + n2*\ln(x2) + n3*\ln(x3)$$

og potenserne n1, n2 og n3 bliver lig med regressionskoefficienterne.

Gruppe 3 er det egentlige datamateriale, hvor oplysningerne om hvert punkt skrives som en linie med et identifikationsnummer, y-værdien og x-værdierne.

Det viste beregningseksempel giver i sektion 1 det samme eksempel som vist side 40. Sektion 2 viser et eksempel paa approximation af et potensudtryk. Man bemærker, at middelfejlen paa c3 er væsentlig større end selve c3, som derfor slet ikke er signifikant.

ALGOL-programmet indeholder fire sideordnede blokke:

1. Forside og overskrift.
2. Indlæsning.
3. Beregning.
4. Trykning.

Det bemærkes, at brugeren af programmet ikke skal opgive nogen minimumsværdi, tol, af diagonalelementet ved invertering af matricen i FIT2. Programmet prøver med tol = $1_{10}-8$, og hvis der saa fremkommer et fejludhop, multipliceres tol med $1_{10}-4$, og man begynder forfra med FIT2.

Bemærk ogsaa, hvorledes der i trykningen er indsat en udskrivning af det formelle udseende af funktionsudtrykket.

3.3. Anvendelse af Ortogonale Polynomier.

3.3.1. Ulemper ved almindelig regressionsanalyse med højere potenser.

Med proceduren FIT2 kan vi finde et lineært tilnærmelsesudtryk til en given funktion, f.eks.:

```
y := y0 + c1×x1 + c2×x2 + c3×x3;
```

hvor vi har tre uafhængige variable. Hvis vi i stedet for ønsker at finde koefficienterne i et polynomium af een variabel, men med højere potenser, f.eks.:

```
y := y0 + c1×x + c2×x↑2 + c3×x↑3;
```

kan man gøre dette ved at lade som om, at de højere potenser er andre uafhængige variable:

```
x1 := x   ;
x2 := x↑2;
x3 := x↑3;
```

Hvis vi beregner x1, x2 og x3 saaledes, kan de tre koefficienter, c1, c2 og c3 straks findes med FIT2.

Men der opstaar alvorlige vanskeligheder, naar graden af det søgte polynomium i x bliver høj. Da værdierne af x, x↑2, x↑3 o.s.v. vil afvige mere og mere fra hinanden, jo større potensen bliver, vil elementerne i matricen SXX (see side 38) ogsaa blive meget forskellige. Inversionen af SXX bliver da unøjagtig, og ved tilstrækkelig høj potens bliver den inverterede matrix helt forkert. Vi har illustreret dette forhold med program d-173 vist nedenfor.

Program d-173. Sammenligning af POLY-1 og FIT-2.

```
begin
   integer i, g, j;
   real ERROR, Y, ymean;
   array x, y[1:13];
```

```
   comment library FIT2;
   comment library INVERT2;
   comment library POLY1;
   for i := 1 step 1 until 13 do læs(x[i], y[i]);
   tryktekst(‹<
  Grad     Middelfejl
           POLY-1  FIT-2
›);
   for g := 1 step 1 until 7 do
   begin
      trykvr;
      tryk(‹-ndd›, g);
      trykml(4);
      begin
         array a[0:g];
         POLY1(13, g, x, y, x, a, false);
         ERROR := 0;
         for i := 1 step 1 until 13 do
         begin
            Y := 0;
            for j := g step -1 until 0 do
            Y := Y×x[i] + a[j];
            ERROR := ERROR + (Y-y[i])↑2
         end for i;
         tryk(‹-ndddd.d›, sqrt(ERROR/12))
      end POLY-1 block;
      begin
         array xx[1:13, 1:g], xmean, coef, coeferror[1:g], ycalc[1:13];
         for i := 1 step 1 until 13 do
         for j := 1 step 1 until g do
         xx[i, j] := x[i]↑j;
         FIT2(g, 13, xx, y, xmean, ymean, ycalc, coef, coeferror, ERROR,
              1_10-20, E1);
         tryk(‹-ndddd.d›, ERROR×sqrt(1-g/12));
         go to F1;
E1:      tryktekst(‹<FEJL›);
F1:   end FIT-2 block
```

```
   end for g
end program;
```

Programmet behandler det samme inputmateriale som program d-174 (side 33), nemlig metans entalpi ved forskellige temperaturer. Nu lader vi programmet beregne et polynomium af stigende grad:

Grad:	Polynomium:
1	y0 + c1×x
2	y0 + c1×x + c2×x↑2
3	y0 + c1×x + c2×x↑2 + c3×x↑3
o.s.v.	

Beregningen sker med FIT2, idet vi sætter x2 lig med x↑2, x3 lig med x↑3, o.s.v.

Programmet beregner og trykker middelfejlen paa det beregnede polynomium i de opgivne punkter, og til sammenligning udføres ogsaa en beregning af approksimationspolynomiet med proceduren POLY1, der benytter ortogonale polynomier, og som omtales nedenfor. Ogsaa her beregnes middelfejlen.

Resultatudskriften er:

Grad	Middelfejl	
	POLY-1	FIT-2
1	681.1	681.1
2	67.6	67.6
3	4.7	4.7
4	3.3	15.6
5	2.7	884.7
6	1.1	49317.4
7	1.1	88415.9

Vi ser, at de to metoder giver samme nøjagtighed op til graden 3, men saa begynder fejlen med FIT2 at vokse, og allerede ved graden 5 giver FIT2 et daarligere resultat end en simpel lineær tilnærmelse. Brugen af ortogonale polynomier bør derfor foretrækkes.

3.3.2. Ortogonale polynomier for een variabel. Proceduren POLY1.

Brugen af ortogonale polynomier til approximationsberegninger hos Haldor Topsøe gaar tilbage til brugen af DASK, hvor vi havde adgang til et approksimationsprogram for funktioner af een variabel, udarbejdet af P. Mondrup. Senere har J. Adriansen udvidet metoden til flere variable. Det vil føre for vidt at give nogen dybtgaaende matematisk forklaring paa metoden her, men der henvises til Ascher og Forsythe (1958), Cadwell (1960) og Lapidus (1962), pag. 332. Vi vil derimod forklare den praktiske brug af procedurerne, samt visse programmeringsmæssige finesser i dem.

Vi omtaler først proceduren POLY1, der benyttes til funktioner af een variabel. Deklarationen er:

```
procedure POLY1(N, P, x, y, w, a, WEIGHING);
integer N, P;
boolean WEIGHING;
array x, y, w, a;
begin
   integer j, k, n;
   real alfa, beta, XPROD, YPROD, SQ, SQSUM, OLDSQSUM, R, olda;
   array error, orpol, oldorpol[1:N], cora[-1:P], oldcora[0:P];
   for n := 1 step 1 until N do
   begin
      error[n] := y[n];
      orpol[n] := 0;
      oldorpol[n] := 1
   end of initial setting;
   alfa := olda := cora[-1] := 0;
   beta := OLDSQSUM := 1;
   for k := 0 step 1 until P do
   begin
      XPROD := YPROD := SQSUM := 0;
      for n := 1 step 1 until N do
      begin
         error[n] := error[n] - olda×orpol[n];
         R := oldorpol[n]×beta;
         oldorpol[n] := orpol[n];
         R := orpol[n] := R + orpol[n]×(x[n] + alfa);
```

```
            if WEIGHING then R := orpol[n]×w[n];
            SQ := R×orpol[n];
            SQSUM := SQSUM + SQ;
            YPROD := YPROD + R×error[n];
            XPROD := XPROD + SQ×x[n]
         end for n;
         a[k] := olda := YPROD/SQSUM;
         oldcora[k] := 0;
         cora[k] := 1;
         if k>0 then
         for j := k-1 step -1 until 0 do
         begin
            R := beta×oldcora[j];
            oldcora[j] := cora[j];
            cora[j] := alfa×oldcora[j] + R + cora[j-1];
            a[j] := a[j] + olda×cora[j]
         end for j;
         beta := -SQSUM/OLDSQSUM;
         OLDSQSUM := SQSUM;
         alfa := -XPROD/SQSUM
      end for k
   end POLY-1;
```

De formelle parametre i POLY1 er:

integer N:	Antallet af tabelværdier (punkter).
integer P:	Den ønskede grad af polynomiet.
array x[1:N]:	Tabelværdierne af den uafhængige variable.
array y[1:N]:	De tilsvarende funktionsværdier.
array w[1:N]:	Tabelværdierne kan forsynes med vægte. w[i] er vægten for punkt nr. i.
array a[0:P]:	De søgte koefficienter i polynomiet. For den aktuelle værdi, X, af den uafhængige variable er polynomiet:

$$a[0] + a[1]\times X + a[2]\times X\uparrow 2 + \ldots\ldots a[P]\times X\uparrow P$$

boolean WEIGHING: Denne indsættes som true, hvis man ønsker at bruge vægtene w[1:N], ellers indsættes den som false, og man behøver da slet ikke at have noget array w[1:N], men kan f.eks. skrive talsættet x[1:N] paa dets plads, saaledes som vi har gjort det i program d-173 (se side 42), som viser en typisk anvendelse af POLY1.

Virkemaaden af POLY1 vil ikke blive omtalt her, men i forbindelse med proceduren POLY8, der som en underblok indeholder de samme beregninger, som foregaar i POLY1.

Vi bruger ingen særlig procedure til beregning af polynomiets værdi, Y, for en opgivet værdi af X og ud fra de fundne koefficienter, a. Det kan f.eks. gøres saaledes:

```
Y := 0;
for j := P step -1 until 0 do
Y := Y*X + a[j];
```

hvor j er en integer variabel. Skal det fundne polynomium indbygges fast i et andet program, er det mest praktisk at skrive koefficienternes talværdi direkte, uden brug af a-talsættet:

```
H := - 19625.11 + T*(3.359595 + T*
(8.495905_10-3 + T*(-7.112638_10-7 - T*2.548678_10-10)));
```

Proceduren POLY1 er indbygget som væsentlig bestanddel af program PA-8, der omtales nedenfor (afsnit 3.3.7).

3.3.3. Beregning af polynomier af mange variable. Proceduren POLY7.

Før vi studerer, hvorledes man beregner koefficienterne i et polynomium af flere variable, er det praktisk at undersøge, hvordan man beregner talværdien af det færdige polynomium for aktuelle værdier af de uafhængige variable.

Som et illustrerende eksempel vælger vi beregning af et polynomium i to variable, x1 og x2, og skriver det først som:

```
y := 1.234       + 2.345×x1       + 3.456×x1↑2
  + 4.567×x2     + 5.678×x1×x2    + 6.789×x1↑2×x2
  + 7.890×x2↑2 + 8.901×x1×x2↑2 + 9.012×x1↑2×x2↑2;
```

Som sædvanligt vil man helst undgaa at skrive potenserne udtrykkeligt og i stedet bruge parenteser. Dette giver metode 2:

```
y := (((9.012×x1 + 8.901)×x1 + 7.890)×x2
  +    (6.789×x1 + 5.678)×x1 + 4.567)×x2
  +    (3.456×x1 + 2.345)×x1 + 1.234;
```

Bemærk, hvorledes koefficienterne nu staar i den omvendte rækkefølge. Hvis der er mange koefficienter, er metode 2 stadig noget upraktisk. Man kan gøre det lidt kortere ved at gøre hver række til et procedurekald. Vi faar da metode 3:

```
begin
   real factor;
   procedure P(a2, a1, a0);
   value a2, a1, a0;
   real a2, a1, a0;
   begin
      y := y + ((a2×x1 + a1)×x1 + a0)×factor;
      factor := factor×x2
   end P;
   y := 0;
   factor := 1;
   P(3.456, 2.345, 1.234);
   P(6.789, 5.678, 4.567);
   P(9.012, 8.901, 7.809)
end;
```

Ved beregning af flere smaapolynomier af to variable er metode 3 ret bekvem, selv om de ekstra procedurekald er noget tidskrævende.

Hvis de ni koefficienter paa een eller anden maade er opstaaet som et rigtigt talsæt: array coef[0:2, 0:2], er beregningen let. Metode 4:

```
begin
   integer i, j;
   real R;
   y := 0;
   for i := 2 step -1 until 0 do
   begin
      R := 0;
      for j := 2 step -1 until 0 do
      R := R*x1 + coef[i, j];
      y := y*x2 + R
   end for i
end;
```

Men vanskeligheden bestaar i at faa tildelt værdier til talsættet coef[0:2, 0:2], naar disse kendes paa forhaand. Det er meget uøkonomisk at indsætte værdierne udtrykkeligt:

```
coef[0,0] := 1.234;
coef[0,1] := 2.345;
coef[0,2] := 3.456;
coef[1,0] := 4.567;
o.s.v.
```

Det er noget bedre at indsætte værdierne med en procedure, som vist i bind 1, side 110 og 242. Den bedste metode er formentlig at indlæse talsættet fra en datastrimmel, enten under selve beregningen eller under programmets udarbejdelse, som vist i bind 1, side 245.

De her viste metoder er stadigvæk upraktiske for meget store polynomier, og især hvis man i samme program ønsker at beregne polynomier med et forskelligt antal variable. Man vil ogsaa gerne kunne beregne polynomier med en afkortet koefficientmatrix, f.eks.:

```
coef[0,0]   coef[0,1]   coef[0,2]
coef[1,0]   coef[1,1]
coef[2,0]
```

uden at de tomme elementer tager plads op.

Proceduren POLY7 er velegnet for store polynomier. Deklarationen er:

```
real procedure POLY7(var, i, first, X, dplace);
value var, first, dplace;
integer var, i, first, dplace;
real X;
begin
   integer count1, count2, INDEX, ROW, COL1, COLMAX, row, col;
   real RES, S1, X1, X2;
   array XX, SUM[1:var], DATA[0:39];
   procedure NEW DATA;
   begin
      tromleplads := dplace;
      dplace := dplace + fra tromle(DATA);
      count1 := 39
   end NEW DATA;
   procedure CUT(high, low);
   integer high, low;
   begin
      high := low:100;
      low := low - high×100
   end CUT;
   X1 := X2 := 0;
   for count1 := 1 step 1 until var do
   begin
      i := first + count1 - 1;
      XX[count1] := X;
      SUM[count1] := 0;
      if count1 = 1 then X1 := XX[1];
      if count1 = 2 then X2 := XX[2]
   end for count1;
   count1 := -1;
L1: if count1 < 0 then NEW DATA;
   COLMAX := DATA[count1];
   count1 := count1 - 1;
   CUT(COL1, COLMAX);
```

```
    CUT(ROW, COL1);
    CUT(INDEX, ROW);
    S1 := 0;
    for row := 1 step 1 until ROW do
    begin
       RES := 0;
       for col := 1 step 1 until COL1 do
       begin
          if count1 < 0 then NEW DATA;
          RES := RES×X1 + DATA[count1];
          count1 := count1 - 1
       end for col;
       S1 := S1×X2 + RES;
       if COL1 < COLMAX then COL1 := COL1 + 1
    end for row;
    for count2 := 2 step 1 until INDEX do
    S1 := S1 + SUM[count2];
    if INDEX = var then POLY7 := S1
    else
    begin
       SUM[INDEX] := S1×XX[INDEX+1];
       for count2 := INDEX-1 step -1 until 2 do
       SUM[count2] := 0;
       go to L1
    end
end POLY7;
```

De fem parametre er:

integer var: Antallet af uafhængige variable. Det maa gerne være 1.
integer i: En tællecelle, der forklares nedenfor.
integer first: Startværdien af i.
real X: Dette udtryk skal give de forskellige værdier af de uafhængige variable for forskellige værdier af tællecellen i. Hvis vi har tre variable, og de uafhængige variable er talsættet XA[1:3], kan et kald af POLY7 se saaledes ud:

```
Z := POLY7(3, i, 1, XA[i], dplace);
```

Det er Jensens device, der er brugt her.

integer dplace: Koefficienterne skal paa forhaand være anbragt paa tromlen med tromleplads lig med dplace for første tal, og de følgende tal lagret for lavere værdier af tromleplads. Koefficienterne er normalt fremkommet ved et kald af proceduren POLY8, der afleverer tallene paa tromlen. Parametren coefplace i POLY8 skal være lig med dplace i POLY7 (med mindre man har flyttet tallene bagefter).

Da POLY7 skal kunne behandle polynomier af vilkaarlig grad i de enkelte variable, og da vi ikke har ønsket at give oplysninger om disse grader iblandt de aktuelle parametre, har vi valgt at lade disse oplysninger staa sammen med koefficienterne paa tromlen. Her staar først en celle med en styreparameter og derefter nogle koefficienter:

Celleindhold:	Tromleplads
PARAMETER	dplace
KOEFFICIENT	dplace-1
KOEFFICIENT	dplace-2
.....	.....
.....	
KOEFFICIENT	

Hvis der er mere end to uafhængige variable, fortsætter listen med en ny parameter, nogle flere koefficienter, o.s.v.

Styreparametren er i virkeligheden fire parametre, som vi betegner med:

INDEX, ROW, COL1, COLMAX

De er anbragt i en enkelt celle som heltallet:

1000000×INDEX + 10000×ROW + 100×COL1 + COLMAX

De koefficienter, som nu følger efter parametercellen, skal opfattes som en lille todimensional tabel, hvor ROW er antallet af rækker, COL1 er

antallet af søjler i første række, og COLMAX er det største antal søjler. Hvis COL1 er mindre end COLMAX, øges antallet af søjler i en række med 1, hver gang vi gaar fra een række til den næste, indtil COLMAX er naaet. Vi tager eksemplet:

INDEX	ROW	COL1	COLMAX
2	4	2	4

Der er her 4 rækker ialt. Første række har 2 søjler, anden række 3 søjler og de to sidste rækker har 4 søjler. Dette giver udseendet:

```
B   B
B   B   B
B   B   B   B
B   B   B   B
```

hvor B-erne betegner koefficienterne, der naturligvis er forskellige. Inden vi regner tabellen ud, ser vi paa de indledende manøvrer i proceduren.

Først overføres listen af de uafhængige variable til et lokalt talsæt: XX[1:var]. Elementerne i et andet lokalt talsæt: SUM[1:var] nulstilles.

Da alle koefficienterne er fast lagret paa tromlen fra dplace og nedefter, maa vi have et lokalt talsæt, DATA[0:39], som talmaterialet kan føres over til. En ny portion paa 40 koefficienter og parametre kan hentes frem med proceduren: NEW DATA.

Styreparametrene opdeles i de fire enkelte parametre med proceduren: CUT. Hver række i den todimensionale koefficienttabel beregnes som et polynomium i den første uafhængige variable: XX[1], som vi har skrevet som X1 for at gøre det hurtigere:

```
R1 := B×X1 + B
R2 := (B×X1 + B)×X1 + B
R3 := ((B×X1 + B)×X1 + B)×X1 + B
R4 := ((B×X1 + B)×X1 + B)×X1 + B
```

Resultatet af hver rækkeberegning gemmes ikke som saadan, men bruges direkte som koefficient i et polynomium i den anden variable: XX[2] = X2:

```
S1 := ((R1×X2+R2)×X2 + R3)×X2 + R4;
```

Nu har vi altsaa beregnet et lille delpolynomium i de to første variable. Til værdien af S1 adderes nu indholdet af de tidligere beregnede sumceller:

```
SUM[2], SUM[3], ....., SUM[INDEX]
```

Cellerne er tomme første gang. Hvis værdien af INDEX er mindre end antallet af variable (var), multipliceres S1 med værdien af den variable nr. INDEX + 1 og resultatet lagres som SUM[INDEX]. Sumcellerne fra SUM[INDEX-1] og nedefter nulstilles. Derefter hentes næste parametercelle frem, og beregningen fortsættes med den næste todimensionale koefficienttabel.

Naar INDEX bliver lig med var, er beregningen slut.

Et eksempel paa brugen af POLY7 er vist i næste afsnit.

3.3.4. Ortogonale polynomier for flere variable. Proceduren POLY8.

Naar vi skal finde en polynomieapproksimation til en funktion af flere variable, er forholdene langt mere kompliceret end ved funktioner af een variabel. Alene beregningen af polynomiets værdi er en kompliceret affære, som vist i sidste afsnit.

For at gøre det en lille smule lettere at forstaa, giver vi straks et lille simpelt eksempel paa brugen af proceduren POLY8 til beregning af polynomiekoefficienterne og af POLY7 til udregning af polynomiet. Vi vil saa bagefter gennemgaa POLY8, og ligeledes forklare et mere generelt program, PA-7, med samme funktion.

Det simple program er:

Program d-176. Brug af POLY-7 og POLY-8.

```
begin
   integer xplace, zplace, coefplace, coefbot, drumbot, i, j, k, ii;
   integer array obslist, deglist[1:3];
   array XX[1:3];
   comment library POLY7;
   comment library POLY8;
```

```
xplace := tromleplads;
læs(obslist);
læs(deglist);
for i := 1 step 1 until 3 do
begin
   array x[1:obslist[i]];
   læs(x);
   til tromle(x)
end for i;
coefplace := tromleplads;
zplace := 3000;
for i := 1 step 1 until obslist[3] do
begin
   array z[1:obslist[2], 1:obslist[1]];
   læs(z);
   drumbot := tromleplads := zplace - 1 + i×obslist[1]×obslist[2];
   til tromle(z)
end for i;
POLY8(3, 6, obslist, deglist, xplace, zplace, coefplace, coefbot,
      drumbot, 0, E);
begin
   array x1[1:obslist[1]], x2[1:obslist[2]], x3[1:obslist[3]];
   tromleplads := xplace;
   fra tromle(x1);
   fra tromle(x2);
   fra tromle(x3);
   trykvr;
   for i := 1 step 1 until obslist[3] do
   begin
      trykvr;
      XX[3] := x3[i];
      for j := 1 step 1 until obslist[2] do
      begin
         trykvr;
         XX[2] := x2[j];
         for k := 1 step 1 until obslist[1] do
         begin
```

```
                XX[i] := x1[k];
                tryk({-ndd.dddd}, POLY7(3, ii, 1, XX[ii], coefplace))
            end for k
          end for j
        end for i
    end blok;
E: end program;
```

Programmet læser en inputstrimmel med følgende tal:

```
Ff
5  6  3
3  3  2
200   220   240   260   280
400  408  416  424  432  440
0  10  20

38.8210  40.9274  42.9013  44.7590  46.5139
36.8153  38.8940  40.8475  42.6906  44.4357
34.8732  36.9184  38.8457  40.6688  42.3990
32.9982  35.0047  36.9007  38.6987  40.4090
31.1934  33.1563  35.0163  36.7846  38.4705
29.4607  31.3760  33.1958  34.9301  36.5875

31.6245  33.3405  34.9477  36.4593  37.8863
29.9900  31.6841  33.2752  34.7754  36.1950
28.4074  30.0746  31.6449  33.1294  34.5374
26.8795  28.5155  30.0608  31.5254  32.9177
25.4086  27.0096  28.5259  29.9668  31.3398
23.9965  25.5590  27.0430  28.4567  29.8070

25.4249  26.8176  28.1221  29.3491  30.5074
24.0996  25.4739  26.7648  27.9822  29.1341
22.8171  24.1690  25.4426  26.6468  27.7890
21.5798  22.9058  24.1586  25.3463  26.4754
20.3894  21.6864  22.9153  24.0833  25.1964
19.2474  20.5126  21.7147  22.8603  23.9546
```

og afleverer følgende resultatstrimmel:

```
38.8209  40.9270  42.9013  44.7586  46.5137
36.8149  38.8935  40.8473  42.6900  44.4353
34.8731  36.9181  38.8457  40.6684  42.3988
32.9981  35.0044  36.9007  38.6983  40.4088
31.1930  33.1558  35.0160  36.7840  38.4700
29.4605  31.3757  33.1957  34.9297  36.5872

31.6243  33.3402  34.9476  36.4589  37.8860
29.9897  31.6836  33.2749  34.7749  36.1946
28.4072  30.0743  31.6449  33.1291  34.5371
26.8793  28.5152  30.0607  31.5250  32.9174
25.4082  27.0091  28.5256  29.9663  31.3394
23.9963  25.5587  27.0428  28.4563  29.8067

25.4247  26.8173  28.1220  29.3487  30.5071
24.0993  25.4734  26.7645  27.9816  29.1337
22.8169  24.1687  25.4425  26.6465  27.7887
21.5796  22.9055  24.1585  25.3459  26.4751
20.3891  21.6859  22.9150  24.0828  25.1960
19.2472  20.5123  21.7145  22.8599  23.9543
```

Program d-176 behandler en funktion af tre variable. Først indlæses værdien af talsættet: obslist[1:3]:

```
obslist[1] := 5;
obslist[2] := 6;
obslist[3] := 3;
```

Dette betyder, at der er 5 værdier af den første variable, 6 værdier af den anden variable og 3 værdier af den tredje. Vi skal da have alle 5×6×3 = 90 kombinationer af de tilsvarende funktionsværdier.

Derefter indlæses det andet talsæt: deglist[1:3]:

```
deglist[1] := 3;
deglist[2] := 3;
deglist[3] := 2;
```

Det er de maksimale grader af polynomiet for hver af de tre variable. Et vilkaarligt led i polynomiet kan skrives som:

$$B*x1\uparrow p1*x2\uparrow p2*x3\uparrow p3$$

og vi har da:

$$0 \leq p1 \leq 3$$
$$0 \leq p2 \leq 3$$
$$0 \leq p3 \leq 2$$

Summen af alle exponenter er ogsaa begrænset:

$$p1 + p2 + p3 \leq degmax$$

Den øvre grænse, degmax, er en parameter i POLY8, men er sat til 6 i programmet. Kun de koefficienter, for hvilke ovennævnte krav er opfyldt, medtages i polynomiet.

Nu indlæses de 5 værdier af x1, de 6 værdier af x2 og de 3 værdier af x3, og de lagres i den øverste frie ende af tromlen og nedefter:

Element	værdi	Tromleplads
x1[5]	280	xplace
x1[4]	260	xplace-1
x1[3]	240	xplace-2
x1[2]	220	xplace-3
x1[1]	200	xplace-4
x2[6]	440	xplace-5
x2[5]	432	xplace-6
x2[4]	424	xplace-7
x2[3]	416	xplace-8
x2[2]	408	xplace-9
x2[1]	400	xplace-10
x3[3]	20	xplace-11
x3[2]	10	xplace-12
x3[1]	0	xplace-13

De tre variable er:

x1: Tryk (atm.abs.)
x2: Temperatur (gr.C)
x3: Molprocent argon.

Funktionsværdierne er ammoniakligevægtsprocenten. Disse indlæses nu og overføres til tromlen:

Element	værdi	Tromleplads
z[x1[1], x2[1], x3[1]]	38.8210	zplace
z[x1[2], x2[1], x3[1]]	40.9274	zplace + 1
z[x1[3], x2[1], x3[1]]	42.9013	zplace + 2
z[x1[4], x2[1], x3[1]]	44.7590	zplace + 3
z[x1[5], x2[1], x3[1]]	46.5139	zplace + 4
z[x1[1], x2[2], x3[1]]	36.8153	zplace + 5
z[x1[2], x2[2], x3[1]]	38.8940	zplace + 6
.....		
.....		
.....		
z[x1[5], x2[6], x3[3]]	23.9546	zplace + 89

Bemærk, hvorledes vi begynder tællingen i den første variables index, samt at funktionsværdierne anbringes paa voksende tromleplads.

Derefter kaldes POLY8, som beregner koefficienterne. Programmet slutter med tre for-sætninger indeni hinanden, som beregner polynomiets værdi med POLY7 og trykker den. De beregnede koefficienter bliver ikke trykt, men kan findes i det typiske eksempel i program PA-7, som udfører samme beregning (dog med normalisering af de variable).

Vi kan nu gennemgaa proceduren POLY8, som har deklarationen:

```
procedure POLY8(var, degmax, obslist, deglist, xplace, zplace, coefplace,
        coefbot, drumbot, cortype, FULL);
value var, degmax, xplace, zplace, coefplace, drumbot, cortype;
integer var, degmax, xplace, zplace, coefplace, coefbot, drumbot, cortype;
integer array obslist, deglist;
label FULL;
begin
   integer DEGPROD, DEGSQ, NDEG, DEGSUM, i, deg, outcell, outplace,
           incell, inplace;
   integer array plist, ilist[1:var];
   array INDATA, OUTDATA[0:39];
   procedure STORE(x);
   value x;
   real x;
```

```
begin
   if outcell < 0 then
   begin
      tromleplads := outplace;
      til tromle(OUTDATA);
      outplace := outplace + fra tromle(OUTDATA);
      outcell := 39
   end if negative;
   OUTDATA[outcell] := x;
   outcell := outcell - 1
end STORE;
real procedure INDOWN;
begin
   if incell < 0 then
   begin
      tromleplads := inplace;
      inplace := inplace + fra tromle(INDATA);
      incell := 39
   end if negative;
   INDOWN := INDATA[incell];
   incell := incell - 1
end INDOWN;
real procedure INUP;
begin
   if incell > 39 then
   begin
      tromleplads := inplace;
      inplace := inplace - fra tromle(INDATA);
      incell := 0
   end if high;
   INUP := INDATA[incell];
   incell := incell + 1
end INUP;
procedure DISPLAY(product, word, radix, cor1, cor2);
value product, cor1, cor2;
integer product, cor1, cor2;
integer array word, radix;
```

```
begin
   integer j, k;
   word[1] := product - 1;
   for j := 2 step 1 until var do
   begin
      k := cor1 + radix[j-1];
      word[j] := word[j-1]÷k;
      word[j-1] := word[j-1] - word[j]×k + cor2
   end for j;
   word[var] := word[var] + cor2
end DISPLAY;
DEGPROD := 1;
DEGSQ := NDEG := DEGSUM := 0;
for i := 1 step 1 until var do
begin
   deg := deglist[i];
   DEGPROD := DEGPROD×(deg+1);
   DEGSQ := DEGSQ + (deg↑2 + 3×deg + 2)÷2;
   NDEG := NDEG + obslist[i]×(deg+1);
   DEGSUM := DEGSUM + deg + 1
end for i;
if 2×DEGPROD + DEGSQ > coefplace - drumbot
then go to FULL;
begin comment calculation of ORPOL, POLC, and SSQPOL;
   integer polplace, orplace, ssplace, n, lim, LIM;
   polplace := coefplace - DEGPROD - DEGSQ;
   orplace := polplace - NDEG;
   ssplace := orplace - DEGSUM;
   for i := 1 step 1 until var do
   begin
      n := obslist[i];
      deg := deglist[i];
      lim := deg×(deg+3)÷2;
      LIM := lim + 2×deg + 3;
      begin comment inner block;
         integer e, k, t;
         real A, B, D, F, G;
```

```
array x[1:n], ORPOL[-2:deg, 1:n], POLC[0:LIM], SSQPOL[0:deg];
integer procedure elem(p, q);
value p, q;
integer p, q;
elem := (p↑2 + 5×p + 6)÷2 + q;
tromleplads := xplace;
xplace := xplace + fra tromle(x);
for k := 1 step 1 until n do
begin
   ORPOL[-2, k] := 1;
   ORPOL[-1, k] := 0
end for k;
for t := 0 step 1 until deg do
begin
   POLC[elem(t, t)] := 1;
   POLC[elem(t-1, t)] := POLC[elem(t, -1)] := 0
end for t;
A := 0;
B := F := 1;
for t := 0 step 1 until deg do
begin
   SSQPOL[t] := G := 0;
   for k := 1 step 1 until n do
   begin
      D := ORPOL[t, k] := ORPOL[t-2, k]×B + ORPOL[t-1, k]×
           (x[k] + A);
      D := D↑2;
      SSQPOL[t] := SSQPOL[t] + D;
      G := G + D×x[k]
   end for k;
   for e := t-1 step -1 until 0 do
   POLC[elem(t,e)] := POLC[elem(t-1, e)]×A
       + POLC[elem(t-2, e)]×B + POLC[elem(t-1, e-1)];
   B := -SSQPOL[t]/F;
   F := SSQPOL[t];
   A := - G/F
end for t;
```

```
            tromleplads := ssplace := ssplace + deg + 1;
            til tromle(SSQPOL);
            begin comment orpol block;
               array orpol[0:deg, 1:n];
               for k := 1 step 1 until n do
               for t := 0 step 1 until deg do
               orpol[t, k] := ORPOL[t, k];
               tromleplads := orplace := orplace + n×(deg + 1);
               til tromle(orpol)
            end orpol block;
            begin comment polc block;
               array polc[0:lim];
               k := 0;
               for t := 0 step 1 until deg do
               for e := 0 step 1 until t do
               begin
                  polc[k] := POLC[elem(t, e)];
                  k := k + 1
               end for t and e;
               tromleplads := polplace := polplace + lim + 1;
               til tromle(polc)
            end polc block
         end inner block
      end for i
   end calculation of ORPOL etc.;
   begin comment scanning of z-data;
      boolean logcor;
      integer base, base1, n1, degprod, nprod, NPROD, psum;
      real SUM, FACTOR, Z;
      array ORPOL[1:NDEG];
      logcor := cortype = 1;
      NPROD := 1;
      for i := 1 step 1 until var do NPROD := NPROD×obslist[i];
      n1 := obslist[1];
      base1 := n1×(1+deglist[1]);
      tromleplads := coefplace - DEGPROD - DEGSQ;
      fra tromle(ORPOL);
```

```
      outcell := 39;
      tromleplads := outplace := coefplace;
      fra tromle(OUTDATA);
      for degprod := 1 step 1 until DEGPROD do
      begin
         SUM := 0;
         DISPLAY(degprod, plist, deglist, 1, 0);
         incell := 40;
         inplace := zplace + 39;
         psum := 0;
         for i := 1 step 1 until var do psum := psum + plist[i];
         if psum ≤ degmax then
         for nprod := 1 step n1 until NPROD do
         begin
            DISPLAY(nprod, ilist, obslist, 0, 1);
            FACTOR := 1;
            base := base1;
            for i := 2 step 1 until var do
            begin
               FACTOR := FACTOR×ORPOL[base+plist[i]×obslist[i] + ilist[i]];
               base := base + obslist[i]×(1+deglist[i])
            end for i;
            base := n1×plist[1];
            for i := 1 step 1 until n1 do
            begin
               Z := INUP;
               if logcor then Z := ln(Z);
               SUM := SUM + Z×ORPOL[base+i]×FACTOR
            end for i
         end for nprod;
         STORE(SUM)
      end for degprod;
      tromleplads := outplace;
      til tromle(OUTDATA)
   end scanning of z-data;
   begin comment division by SSQPOL;
      integer degprod, base, base1;
```

```
    real FACTOR;
    array SSQPOL[0:DEGSUM-1];
    deg := deglist[1] + 1;  base1 := deg - 1;
    tromleplads := coefplace - DEGPROD - DEGSQ - NDEG;
    fra tromle(SSQPOL);
    outcell := 39;
    tromleplads := inplace := outplace := coefplace;
    fra tromle(OUTDATA);
    incell := -1;
    for degprod := 1 step deg until DEGPROD do
    begin
       DISPLAY(degprod, plist, deglist, 1, 0);
       FACTOR := 1;
       base := deg;
       for i := 2 step 1 until var do
       begin
          FACTOR := FACTOR×SSQPOL[base+plist[i]];
          base := base + deglist[i] + 1
       end for i;
       for i := 0 step 1 until base1 do
       STORE(INDOWN/SSQPOL[i]/FACTOR)
    end for degprod;
    tromleplads := outplace;
    til tromle(OUTDATA)
end division by SSQPOL;
begin comment calculation of b-coefficients;
    integer degpr1, degpr2, base, h1, p1, base1, deg1;
    real SUM, FACTOR, a;
    integer array hlist, limlist[1:var];
    boolean range;
    array POLC[0:DEGSQ-1];
    for i := 1 step 1 until var do
    limlist[i] := deglist[i]×(deglist[i] + 3):2 + 1;
    base1 := limlist[1];
    deg1 := deglist[1];
    tromleplads := coefplace - DEGPROD;
    fra tromle(POLC);
```

```
outcell := 39;
tromleplads := outplace := coefplace - DEGPROD - DEGSQ;
fra tromle(OUTDATA);
for degpr1 := 1 step deg until DEGPROD do
begin
   DISPLAY(degpr1, hlist, deglist, 1, 0);
   for h1 := 0 step 1 until deg1 do
   begin
      SUM := 0;
      inplace := coefplace;
      incell := -1;
      for degpr2 := 1 step deg until DEGPROD do
      begin
         DISPLAY(degpr2, plist, deglist, 1, 0);
         range := true;
         for i := 2 step 1 until var do
         if plist[i] < hlist[i] then range := false;
         FACTOR := 1;
         base := base1;
         if range then for i := 2 step 1 until var do
         begin
            FACTOR := FACTOR×POLC[base + plist[i]×(plist[i] + 1):2
                      + hlist[i]];
            base := base + limlist[i]
         end for i;
         for p1 := 0 step 1 until deg1 do
         begin
            a := INDOWN;
            if p1 ≥ h1 ∧ range then
            SUM := SUM + a×FACTOR×POLC[p1×(p1+1):2 + h1]
         end for p1;
      end for degpr2;
      STORE(SUM)
   end for h1
end for degpr1;
tromleplads := outplace;
til tromle(OUTDATA)
```

```
    end calculation of b-coefficients;
    begin comment inversion of coefficients;
       boolean out;
       integer deg1, deg2, prod, INDEX, ROW, ROWMAX, COL1, COLMAX,
               degprod, degsum, j, k;
       real B;
       incell := 40;
       inplace := coefplace - DEGSQ - 2×DEGPROD + 40;
       outcell := 39;
       tromleplads := outplace := coefplace;
       fra tromle(OUTDATA);
       deg1 := deglist[1];
       deg2 := if var > 1 then deglist[2] else 0;
       prod := (1+deg1)×(1+deg2);
       COLMAX := 1 + deg1;
       ROWMAX := 1 + deg2;
       for degprod := DEGPROD step - prod until 1 do
       begin
          DISPLAY(degprod, plist, deglist, 1, 0);
          degsum := 0;
          for i := 3 step 1 until var do
          degsum := degsum + plist[i];
          INDEX := var + 1;
          if degsum ≤ degmax then
          begin
             if var > 2 then
             for INDEX := 3 step 1 until var do
             if plist[INDEX] > 0 then go to L2;
             INDEX := var + 1;
L2:          INDEX := INDEX - 1;
             COL1 := COLMAX;
             ROW := ROWMAX;
             degsum := degmax - degsum + 1;
             if ROW > degsum then ROW := degsum;
             COL1 := degsum - ROW + 1;
             if COL1 > COLMAX then COL1 := COLMAX;
             STORE(COLMAX + 100×COL1 + 10000×ROW + 1000000×INDEX);
             out := true
```

```
        end
        else
        out := false;
        for j := 0 step 1 until deg1 do
        for k := 0 step 1 until deg2 do
        begin
           B := INUP;
           if out ∧ B ≠ 0 then STORE(B)
        end for j, k
      end for degprod;
      tromleplads := outplace;
      til tromle(OUTDATA);
      coefbot := tromleplads + outcell
   end inversion of coefficients
end POLY8;
```

Parametrene i POLY8 er:

integer var: Antallet af variable.

integer degmax: Den maksimale sum af graderne i de enkelte variable.

integer array obslist[1:var]: Antallet af værdier af de enkelte variable.

integer array deglist[1:var]: Den maksimale grad af polynomiet for hver variabel.

integer xplace: Værdien af tromleplads for sidste element i talsættet med værdierne af den første variable. De næste variable er anbragt umiddelbart herunder.

integer zplace: Værdien af tromleplads for den første funktionsværdi. De næste er anbragt over denne plads, idet der tælles først i det første index.

integer coefplace: Proceduren anbringer de beregnede koefficienter fra denne værdi af tromleplads og nedefter. Pladsen her benyttes ogsaa som arbejdslager, og bør derfor gøres saa stor som muligt.

integer coefbot: Naar proceduren er færdig med at regne, vil koefficienternes tromleplads gaa fra coefplace til coefbot + 1. Værdien af coefbot, som POLY8 har tildelt, er den første frie plads paa tromlen.

integer drumbot: Dette er den laveste værdi af tromleplads, som proceduren maa have lov at bruge som arbejdscelle.

integer cortype: Denne kan specificeres som 0 eller 1. Hvis den er 1, vil proceduren tage den naturlige logaritme af funktionsværdierne før de indgaar i beregningerne. De ændres ikke paa tromlen. For cortype = 0 sker der ingen korrektion.

label FULL: Hvis det nødvendige antal arbejdsceller overskrider coefplace - drumbot, kan beregningerne ikke gennemføres, og proceduren foretager et fejludhop til denne etikette.

I proceduren POLY8 benyttes fire lokale procedurer. De er:

procedure STORE(x). Denne procedure lagrer værdien af x i elementet: OUTDATA[outcell] og tæller 1 ned i parametren: outcell. Talsættet OUTDATA[0:39] bruges som et sæt lokale arbejdsceller. Hvis outcell bliver negativ, overføres talsættet til tromlen, og de næste lavere 40 celler overføres fra tromlen til talsættet.

real procedure INDOWN. Denne procedure henter det næste tal:

```
INDOWN := INDATA[incell]
```

fra et lokalt talsæt: INDATA[0:39] og tæller 1 ned i parametren: incell. Hvis incell bliver negativ, overføres de næste lavere 40 celler fra tromlen til INDATA.

De to procedurer STORE og INDOWN udgør en automatisk mekanisme, hvor man hele tiden kan lagre eller hente det næste element i et stort talsæt, som i realiteten er anbragt paa tromlen.

real procedure INUP. Denne gør det samme som INDOWN, men tæller op i incell og paa tromlen. Den bruges til at hente funktionsværdierne, som er lagret ved stigende tromleplads.

procedure DISPLAY. Denne procedure er indført for at undgaa at skulle behandle talsæt af et ukendt antal dimensioner og programmering af et ukendt antal for-sætninger indeni hinanden. I det ovennævnte eksempel med tre variable har vi brug for at kunne udføre operationer af typen:

```
for i3 := 1 step 1 until 3 do
for i2 := 1 step 1 until 6 do
for i1 := 1 step 1 until 5 do
```

```
begin
    ........
    A := A + z[x1[i1], x2[i2], x3[i3]]
    *orpol1[p1, i1]*orpol2[p2, i2]*orpol3[p3, i3];
    ........
end;
```

De tre tal 5, 6 og 3 er obslist[1:3]. Elementet orpol1[p1, i1] er det ortogonale polynomium af graden p1 for værdien x1[i1] af den første variable, og analogt for de to andre variable. Programstumpen er her skrevet for tre variable, men havde der været fire variable, skulle der have været fire for-sætninger indeni hinanden, z-elementet skulle have haft et index mere og der skulle have været yderligere en faktor orpol4[p4, i4].

Dette kan ikke skrives direkte i ALGOL. Vi omformer z-talsættet med de var dimensioner til et eendimensionalt talsæt og ligeledes omskrives de var talsæt orpol1, orpol2, o.s.v. til et enkelt talsæt med een dimension. Endelig omskrives de var for-sætninger til en enkelt for-sætning, som her gennemløbes 5*6*3 = 90 gange:

```
for nprod := 1 step 1 until 90 do
```

Vi benytter saa proceduren DISPLAY til for hver værdi af nprod at finde ud af hvilken værdi i1, i2, o.s.v. ville have haft, hvis vi havde lavet det som var for-sætninger. Som eksempel kan vi tage kaldet:

```
DISPLAY(nprod, ilist, obslist, 0, 1);
```

Med ovennævnte værdier af obslist (5, 6 og 3) vil vi faa følgende værdier af ilist[1:3] (der skal bruges som i1, i2 og i3) for forskellige værdier af nprod:

nprod	ilist[1]	ilist[2]	ilist[3]
1	1	1	1
2	2	1	1
3	3	1	1
4	4	1	1
5	5	1	1
6	1	2	1
7	2	2	1
.....			
.....			
90	5	6	3

Vi kan opfatte det paa den maade, at tallet nprod bliver omskrevet til et andet tal i et talsystem med blandede grundtal, nemlig 5, 6 og 3, som defineres af obslist. Hvis vi sætter:

```
obslist[1] := 12;
obslist[2] := 20;
obslist[3] := vilkaarligt, stort.
```

kan proceduren **DISPLAY** bruges til at omregne et givet antal pence til pence, shilling og pund.

Grundtallisten i **DISPLAY** kan ogsaa være listen over grader: deglist, som i kaldet:

```
DISPLAY(degprod, plist, deglist, 1, 0);
```

Bemærk, at den laveste værdi af et element i plist er 0, men i ilist er det 1. Denne lille forskel behandles af de to sidste parametre (0, 1 eller 1, 0).

Proceduren **POLY8** er inddelt i de fem blokke:

1. Beregning af **ORPOL**, **POLC** og **SSQPOL**.
2. Behandling af z-værdier.
3. Division med **SSQPOL**.
4. Beregning af b-koefficienter.
5. Omflytning af b-koefficienter.

Skemaet paa næste side viser, hvordan talsættene er lagret paa tromlen og hvilke talsæt, der bruges i de forskellige blokke.

Hovedtalsættet er array a[1:DEGPROD], med DEGPROD givet ved:

```
DEGPROD := (1 + deglist[1])×
           (1 + deglist[2])×
            .
            .
            .
           (1 + deglist[var]);
```

Under dette er anbragt array polc [1:DEGSQ] med DEGSQ givet ved:

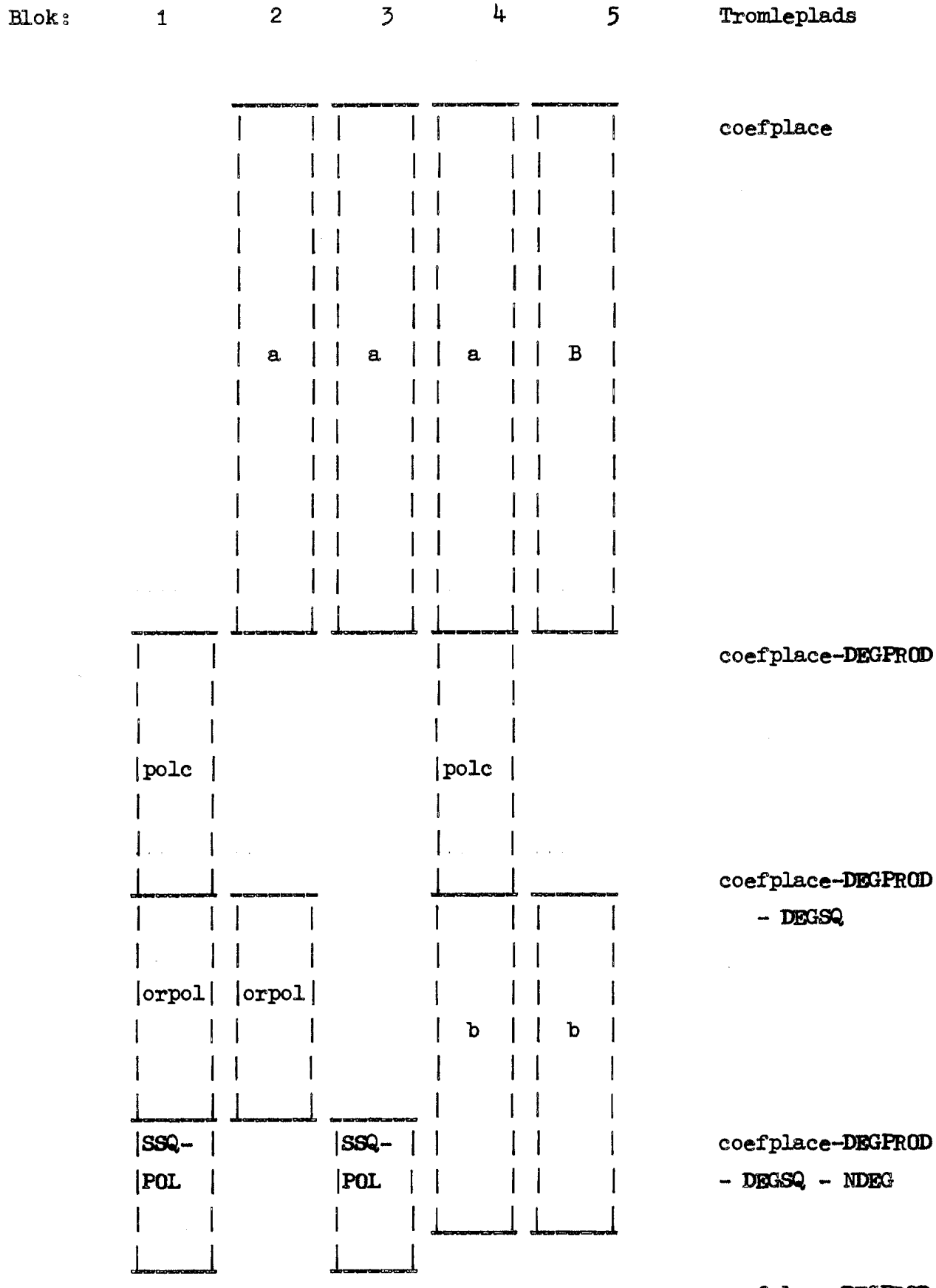
```
DEGSQ := (deglist[1]↑2 + 3×deglist[1] + 2):2
       + (deglist[2]↑2 + 3×deglist[2] + 2):2
       +  ........
          ........
       + (deglist[var]↑2 + 3×deglist[var] + 2):2;
```

Under dette talsæt bruges tromlen først til lagring af de to talsæt: array orpol[1:NDEG], SSQPOL[1:DEGSUM] og derefter til talsættet: array b[1:DEGPROD]. Her er:

```
NDEG := obslist[1]×(1 + deglist[1])
      + obslist[2]×(1 + deglist[2])
      +  ........
         ........
      + obslist[var]×(1 + deglist[var]);
```

og

```
DEGSUM := (1 + deglist[1])
        + (1 + deglist[2])
        +  ........
           ........
        + (1 + deglist[var]);
```


Blok: 1 2 3 4 5
Tromleplads
coefplace
a
a
a
B
coefplace-DEGPROD
polc
polc
coefplace-DEGPROD
- DEGSQ
orpol
orpol
b
b
SSQ-
POL
SSQ-
POL
coefplace-DEGPROD
- DEGSQ - NDEG
coefplace-DEGPROD
- DEGSQ - NDEG
- DEGSUM

Efter beregning af DEGPROD, DEGSQ, NDEG og DEGSUM kontrollerer proceduren, at der er plads til talsættene, ellers hoppes til etiketten: FULL.

Blok 1. Beregning af ORPOL, POLC og SSQPOL. I denne blok beregnes de ortogonale polynomier for hver af de uafhængige variable. Funktionsværdierne udnyttes endnu ikke. Blokken har en for-sætning:

for i := 1 step 1 until var do

som behandler x-værdierne af hver af de uafhængige variable uden hensyn til de andre uafhængige variable. For hver værdi af i hentes det tilsvarende sæt x-værdier frem fra tromlen som array x[1:n], hvor n := obslist[i]. I eksemplet ovenfor bliver det for i = 1 tallene:

200, 220, 240, 260, 280

Den maksimale grad, deglist[i], betegnes for nemheds skyld med: deg. Vi skal nu beregne de tre lokale talsæt: array ORPOL[-2:deg, 1:n], POLC [0:LIM], SSQPOL[0:deg].

Elementet ORPOL[t, k] er det ortogonale polynomium af graden t for x-værdien: x[k]. For n = 5, deg = 3 og de fem x-værdier givet ovenfor giver udregningerne:

ORPOL[t, k]:

t: \ k:	1	2	3	4	5
-2					
-1					
0	1	1	1	1	1
1	-40	-20	0	20	40
2	800	-400	-800	-400	800
3	-9600	19200	0	-19200	9600

De to første rækker (t = -2 og t = -1) bruges kun under beregningen, men ikke i de følgende blokke. Efter beregningen af ORPOL er der derfor anbragt en særlig lille blok med talsættet: array orpol[0:deg, 1:n], hvis elementer sættes lig de tilsvarende elementer i ORPOL, og som derefter overføres til tromlen. De enkelte orpol-talsæt for i = 1, 2, o.s.v. lagres lige under hinanden paa tromlen.

Talsættet POLC indeholder koefficienterne til de ortogonale polynomier. Det er i virkeligheden en triangulær matrix, men lagres som et eendimensionalt talsæt. For at gøre det lettere at følge, hvad der sker under beregningen, skriver vi et element i POLC som:

```
POLC[elem(p, q)]
```

hvor elem er en integer procedure:

```
elem := (p↑2 + 5×p + 6)÷2 + q;
```

Den triangulære matrix har grænserne [-1:deg, -1:deg], og i tilfældet ovenfor med deg = 3 faar vi følgende sammenhæng mellem p, q og elem:

elem(p, q):

q:	-1	0	1	2	3
p:					
-1	0	1			
0	2	3	4		
1	5	6	7	8	
2	9	10	11	12	13
3	14	15	16	17	18

Værdien af LIM i deklarationen array POLC[0:LIM] er:

```
LIM := (deg↑2 + 7×deg + 6)÷2;
```

Efter beregningen har vi ikke brug for elementerne med p = -1 og q = -1 og heller ikke for linien over diagonalen. Vi laver derfor en anden lille blok med talsættet: array polc[0:lim] og lim givet ved:

```
lim := (deg↑2 + 3×deg):2;
```

POLC-elementerne overføres til polc-elementerne og derefter til tromlen. Elementerne i polc har da numrene:

elem(p, q):

p: \ q:	0	1	2	3
0	0			
1	1	2		
2	3	4	5	
3	6	7	8	9

Med x-værdierne ovenfor faar vi følgende talværdier:

polc[elem(p, q)];

p: \ q:	0	1	2	3
0	1			
1	-240	1		
2	56800	-480	1	
3	-13497600	171440	-720	1

I det tredje talsæt: array **SSQPOL**[0:deg] er elementet **SSQPOL**[t] summen af kvadraterne paa **ORPOL**[t, k], summeret for $1 \leq k \leq n$. Dette talsæt overføres direkte til tromlen uden nedsættelse af størrelsen. Taleksemplet ovenfor giver:

SSQPOL[t]:

t:	0	1	2	3
	5	4000	2240000	921600000

Beregningen af de tre talsæt gøres saaledes. Først sættes startværdier af **ORPOL** og **POLC**:

ORPOL[-2, k] := 1
ORPOL[-1, k] := 0

for $1 \leq k \leq n$, og:

POLC[elem(t, t)] := 1
POLC[elem(t-1, t)] := 0
POLC[elem(t, -1)] := 0

for $0 \leq t \leq$ deg. De følgende elementer dannes derefter ved hjælp af to rekursionsformler. Den første er:

ORPOL[t, k] := ORPOL[t-2, k]×B + ORPOL[t-1, k]×(x[k]+A);

Koefficienterne A og B er givet ved:

$$A := \frac{-(x[1]\times ORPOL[t-1, 1]\uparrow 2 + x[2]\times ORPOL[t-1, 2]\uparrow 2 + \ldots\ldots\ldots + x[n]\times ORPOL[t-1, n]\uparrow 2)}{SSQPOL[t-1]}$$

$$B := -\frac{SSQPOL[t-1]}{SSQPOL[t-2]}$$

og

SSQPOL[t] := ORPOL[t, 1]↑2 + ORPOL[t, 2]↑2 +
..... + ORPOL[t, n]↑2

Den anden rekursionsformel er:

POLC[elem(t, e)] := POLC[elem(t-1, e)]×A
+ POLC[elem(t-2, e)]×B + POLC[elem(t-1, e-1)]

Beregningen bestaar da af en for-sætning, hvor t gaar fra 0 til deg. For hver værdi af t udføres to andre for-sætninger. I den første gaar k fra 1 til n, og den første rekursionsformel udføres, sammen med summation af SSQPOL[t]. Den anden for-sætning kontrolleres af parametren: e, som gaar fra t-1 til 0, og som bruger den anden rekursionsformel.

Blok 2. Behandling af z-værdier. I denne blok bruger vi z-værdierne og de var orpol-talsæt, der er lagret paa tromlen, til at generere talsættet: a[1:DEGPROD], eller:

```
a[0:deglist[1], 0:deglist[2], ........
                ........, 0:deglist[var]]
```

Hvert a-element:

```
a[p1, p2, p3, .........]
```

findes ved summation af produktet:

```
z[x1[i1], x2[i2], x3[i3], .........]
×orpol1[p1, i1]×orpol2[p2, i2]×orpol3[p3, i3]×.........
```

for alle værdier af i1, i2, i3 o.s.v. i omraadet:

```
1 ≤ i1 ≤ obslist[1]
1 ≤ i2 ≤ obslist[2]
1 ≤ i3 ≤ obslist[3]
.........
o.s.v.
```

Som forklaret ovenfor, er de var for-sætninger indeni hinanden, som egentlig krævedes til denne summation, erstattet af en enkelt for-sætning og brug af proceduren DISPLAY. Under beregningen er alle orpol-talsættene anbragt i lageret som et enkelt stort talsæt: ORPOL[1:NDEG], og de enkelte elementer heri findes ved beregning af deres index.

For hvert a-element skal alle z-værdier gennemløbes, men hvis

$$p1 + p2 + p3 + \ldots\ldots\ldots > degmax$$

bliver a sat til nul og z-værdierne gennemløbes ikke.

Blok 3. Division med SSQPOL. I denne blok divideres hvert a-element:

$$a[p1, p2, p3, \ldots\ldots\ldots]$$

med produktet:

$$SSQPOL1[p1]\times SSQPOL2[p2]\times SSQPOL3[p3]\times\ldots\ldots\ldots$$

Disse var talsæt overføres fra tromlen og anbringes i lageret som eet stort eendimensionalt talsæt: SSQPOL[0:DEGSUM-1]. Efter division af a-elementerne føres disse tilbage paa samme sted af tromlen.

Blok 4. Beregning af b-koefficienter. Her omdannes a-elementerne til de egentlige polynomiekoefficienter: b-koefficienterne. Hver koefficient:

$$b[h1, h2, h3, \ldots\ldots\ldots]$$

er en koefficient i det endelige polynomium, hvor den skal multipliceres med:

$$x1\uparrow h1\times x2\uparrow h2\times x3\uparrow h3\times\ldots\ldots\ldots$$

Hver b-koefficient findes ved summation af alle led af formen:

$$a[p1, p2, p3, \ldots\ldots\ldots]$$
$$\times polc1[p1, h1]\times polc2[p2, h2]\times polc3[p3, h3]\times\ldots\ldots\ldots$$

i omraadet:

$$h1 \leq p1 \leq deglist[1]$$
$$h2 \leq p2 \leq deglist[2]$$
$$h3 \leq p3 \leq deglist[3]$$
$$\ldots\ldots\ldots$$
o.s.v.

De var polc-talsæt overføres fra tromlen som et enkelt eendimensionalt talsæt: POLC[0:DEGSQ-1], hvor de enkelte elementer findes ved indexberegning.

Blok 5. Omflytning af b-koefficienter. Her anbringes b-koefficienterne i omvendt rækkefølge, idet nul-elementerne overspringes, og der indsættes den nødvendige styreparameter før hver lille todimensionale koefficientgruppe, som forklaret side 52.

3.3.5. Diskussion af program PA-7. Procedurerne POLY7 og POLY8 er indbygget som de væsentlige bestanddele af program PA-7. I appendikset vises inputspecifikationerne til PA-7, en typisk resultatrapport, samt selve ALGOL-programmet.

Der er 7 inputdatagrupper, og gruppe 0 og 1 er som sædvanligt forside og overskriftsdata. Gruppe 2 indeholder antallet af variable, V, og antallet af funktioner, F, idet vi ønsker at kunne behandle flere forskellige funktioner, alle opgivet for de samme værdier af de uafhængige variable. Her opgives ogsaa forskellige styreparametre til beregningen.

I datagruppe 3 skal man for hver af de V variable opgive tre tal:

1. Antallet af værdier (obslist[i]).
2. Den maksimale grad af polynomiet i denne variable (deglist[i]).
3. En korrektionstype.

Ved passende opgivelse af korrektionstypen kan man faa programmet til at ombytte de originale x-værdier med de reciprokke værdier eller den naturlige logaritme eller den normaliserede værdi, d.v.s. saaledes at x-værdierne parallelforskydes, idet den største værdi og den mindste værdi gøres lige store med modsat fortegn. Der kan ogsaa bruges kombinationer af disse korrektionstyper.

I datagruppe 4 opgives de V sæt af de uafhængige variable, og i datagruppe 5 de F sæt funktionsværdier.

Programmet beregner polynomieværdierne i alle de opgivne punkter, men man kan ogsaa faa beregnet en udvidet tabel over funktionsværdierne. Oplysninger herom specificeres i datagruppe 6. Endelig kan man faa programmet til at udføre en nærmere analyse af det beregnede polynomium.

Der er fire muligheder:

1. Find maksimum af den første funktion.
2. Find maksimum af den første funktion, idet de øvrige funktioner samtidigt skal antage bestemte værdier.
3. Find bestemte værdier af alle funktioner.
4. Varier den første uafhængige variable, saaledes at den første funktion antager en bestemt værdi og de partielle differentialkvotienter med hensyn til de andre variable bliver nul.

Funktionsanalysen udføres med proceduren OPT7, og den forklares nærmere i afsnit 5.3.4. Data for analysen specificeres i datagruppe 7.

Det viste beregningseksempel giver i sektion 1 det samme eksempel som paa side 56. Sektion 2 giver et eksempel paa brug af funktionsanalysen.

ALGOL-programmet indeholder følgende blokke:

1. Forside og overskrift.
2. Indlæsning.
3. Trykning af generelle data.
4. Korrektion af x-værdier.
5. Beregning.
6. Trykning af tabeller.
7. Trykning af polynomiets koefficienter.
8. Polynomieanalyse.

Den væsentlige del af beregningen er kaldet af POLY8, der sker i blok 5.

I blok 6 bruges en lokal procedure: TABLES, der kan trykke to forskellige slags tabeller, enten indeholdende de originale funktionsværdier, værdierne beregnet fra polynomiet og fejlen, eller den udvidede funktionstabel.

Begge typer af tabeller trykkes som todimensionale tabeller med de to første variable som abscisse og ordinat. Hvis der er flere end to variable, sker trykningen som en serie af todimensionale tabeller. For hver tabel i denne serie holdes variabel nr. 3, 4, o.s.v. konstant og deres talværdier trykkes som en lille tabel ovenover hovedtabellen. Hvis der er mere end fem værdier af den første variable, inddeles hver tabel i flere

grupper, som vist i sektion 2.

3.3.6. Specielle krav til approksimationspolynomiet. I nogle tilfælde stiller man særlige krav til det polynomium, som skal udtrykke et foreliggende sæt tabelværdier. Man kan f.eks. forlange, at polynomiet skal passere nøjagtigt gennem eet eller flere punkter uden nogen fejl. Ligeledes kan man forlange, at polynomiets differentialkvotient skal antage bestemte værdier i bestemte punkter.

Som eksempel tager vi en funktion af een variabel: $y = f(x)$, som foreligger i tabelform. Vi ønsker, at polynomiet skal passere nøjagtigt gennem de to punkter: (a, A) og (b, B), altsaa at:

$$y = A \text{ for } x = a$$

og

$$y = B \text{ for } x = b$$

Til løsning heraf skriver vi det søgte polynomium, y, som en sum af to led:

```
y := MODPOL + PROD×MAINPOL;
```

Her er MODPOL, PROD og MAINPOL alle polynomier. Hvis vi sørger for at MODPOL har værdien A for x = a og værdien B for x = b, samt at PROD = 0 for baade x = a og x = b, har vi dermed tilfredsstillet det ønskede krav, og skal da blot sørge for at polynomiet MAINPOL tilpasses saa godt som muligt til de opgivne tabelværdier.

Da PROD skal være nul for x = a og x = b maa det være:

```
PROD := (x-a)×(x-b);
```

Da MODPOL skal gaa eksakt igennem de to punkter (a, A) og (b, B) maa det være et førstegradspolynomium, som vi kan skrive som:

```
y := c0 + c1×x;
```

Vi finder c0 og c1 af de to ligninger:

```
A = c0 + c1×a
B = c0 + c1×b
```

til:

```
c0 := (B×a-A×b)/(a-b);
c1 := (A-B)/(a-b);
```

og derfor:

```
MODPOL := A + (A-B)/(a-b)×(x-a);
```

Vi kender nu MODPOL og PROD og kan beregne deres værdi i hvert af de opgivne tabelpunkter. I hvert af disse punkter beregner vi nu en ny y-værdi, yn, som:

```
yn := (y-MODPOL)/PROD;
```

og vi kan nu finde koefficienterne i MAINPOL ved at behandle yn-værdierne sammen med x-værdierne, f.eks. med proceduren POLY1.

3.3.7. Diskussion af program PA-8. I dette program beregner man polynomietilnærmelser med proceduren POLY1 og kan faa eventuelle specielle krav til polynomiet opfyldt som antydet ovenfor.

I appendikset vises inputspecifikationerne til PA-8, samt en typisk resultatrapport og ALGOL-programmet.

Af de syv inputdatagrupper er gruppe 0 og 1 de sædvanlige forside- og overskriftsdata. Gruppe 2 indeholder forskellige beregningsparametre, som antallet af funktioner, F, antallet af tabelpunkter, obs, og antallet af specielle punkter, spec.

For graden af det ønskede approksimationspolynomium skal brugeren opgive en minimumsværdi og en maksimumsværdi. Specificeres disse som f.eks. 2 og 7, vil programmet gennemprøve værdierne 2, 3, 4, 5, 6 og 7, idet det i hvert forsøg beregner middelfejlen ved brug af approksimationspolynomiet.

Normalt vil denne fejl aftage med stigende grad, men det kan ske, at den begynder at vokse igen, og programmet vil da blive staaende ved den grad, der gav den laveste middelfejl.

Ligesom i program PA-7 kan man her faa korrigeret værdierne af x og y, og man kan faa trykt en udvidet funktionstabel og faa analyseret det fundne polynomium.

Hvis der er specielle krav til polynomiet, skal de tilsvarende oplysninger anføres i datagruppe 3. For hvert specielt punkt skal man opgive værdien af x, værdien af y og et typenummer. Hvis typenummeret opgives som nul, opfattes y som selve funktionen, men er typen et positivt heltal, N, opfatter programmet den specificerede y-værdi som funktionens differentialkvotient af orden N.

I datagruppe 4 opgives de normale tabelpunkter, som skal tilpasses saa godt som muligt, men ikke nødvendigvis passe eksakt.

Datagruppe 5 bruges til specifikation af vægte til de normale tabelpunkter, men kan udelades, hvis man ikke ønsker at bruge vægte.

Datagruppe 6 bruges til oplysninger om udvidet tabeltrykning. Foruden funktionen selv kan man ogsaa faa trykt en tabel over differentialkvotienten af første eller anden orden.

Det gælder iøvrigt for baade program PA-7 og PA-8, at inputdatagrupperne med mange tal pakkes saa tæt som muligt paa tromlen. Talmaterialet staar uforandret fra den ene sektion til den næste, og man kan derfor bruge bogstaverne d og q paa inputstrimlen, hvis talmaterialet er uændret (se bind 1, side 228). Men hvis man ændrer størrelsen af de enkelte datagrupper fra een sektion til den næste, flyttes talmaterialet paa tromlen og dittosystemet bryder sammen. Hele det nye datamateriale maa da gentages paa inputstrimlen.

I det viste beregningseksempel illustrerer sektion 1-4 den automatiske udvælgelse af den polynomiegrad, der giver lavest middelfejl (her 5). I sektion 5 har vi bedt om at faa taget logaritmen til y før approksimationen, og i sektion 6 har vi yderligere bedt om at faa taget den reciprokke x-værdi. Endelig viser sektion 7 brugen af specielle punkter. Det er kogepunktskurven for ætanol. Molbrøken af ætanol er x og y er kogepunktet i celcius. Vi har de to specielle punkter, hvor kogepunktet skal passe eksakt:

x	y	type
0.	100	0
0.89404	78.15	0

Endvidere skal der være vandret tangent i azeotroppunktet:

x	y	type
0.89404	0	1

Naar man bruger specielle punkter, kan man faa programmet til automatisk at indsætte vægte, som tager hensyn til den indirekte maade beregningen foregaar paa.

ALGOL-programmet indeholder de seks blokke:

1. Forside og overskrift.
2. Indlæsning.
3. Trykning af beregningsparametre.
4. Beregning af MODPOL.
5. Beregning af MAINPOL.
6. Trykning af udvidede tabeller og polynomieanalyse.

Beregning af MODPOL sker ved løsning af spec ligninger med spec ubekendte. Koefficienterne i matricen beregnes og den inverteres med proceduren INVERT2.

Ved beregning af MAINPOL benyttes en lokal procedure, FIT, der til en given værdi, deg, af polynomiegraden beregner yn-værdierne som vist ovenfor, behandler dem paa POLY1 og finder middelfejlen.

FIT-proceduren kaldes først uden resultattrykning for stigende værdi af deg, indtil den optimale er fundet. Derefter fikseres deg til denne værdi, og FIT kaldes igen, men nu med resultattrykning.

Polynomieanalysen foregaar med proceduren NOLEQ3, der omtales i afsnit 4.5.1.

3.3.8. Behandling af manglende data. Det er en ulempe ved brugen af proceduren POLY8, at man skal kende funktionsværdierne i alle gitterpunkterne svarende til kombinationer af de benyttede værdier af de uaf-

hængige variable. I program d-176 paa side 56 skulle vi saaledes kende alle 90 funktionsværdier.

Hvis man skal finde et approksimationspolynomium til en funktion af flere variable og tabellen over funktionsværdierne er ufuldstændig, men dog indeholder de fleste af gitterpunkterne, kan man ofte klare sig paa følgende maade. Man indsætter omtrentlige funktionsværdier i de manglende punkter og beregner polynomiet med POLY8. Nu indsætter vi polynomiets værdi i de manglende punkter, gentager beregningen med POLY8, o.s.v. indtil konvergens. Denne viser sig ved at de manglende punkter antager en konstant værdi og ved at middelfejlen kun aftager meget langsomt.

Hvis det originale talmateriale er meget sparsomt, eller man har gættet daarlige startværdier i de manglende punkter, kan konvergensen være meget langsom og metoden kan ikke anbefales.

Vi har i øjeblikket ikke noget specielt ALGOL-program til iterativ beregning af approksimationspolynomier med manglende punkter, men der findes et program, PA-5, i maskinsprog, som løser dette for funktioner af to variable. Det er en variant af program PA-4, der er en forløber i maskinsprog til program PA-7. Kan PA-5 ikke bruges, maa man lave et lille ad hoc program i ALGOL ligesom program d-176 med iterativ tilbageføring af funktionsværdierne.

Skal et polynomium i flere variable tilfredsstille specielle krav med hensyn til funktionen eller de partielle afledede, maa den nødvendige forbehandling af talmaterialet ogsaa kodes specielt.

4. RODBESTEMMELSE ELLER LØSNING AF ULINEÆRE LIGNINGER

Et hyppigt forekommende numerisk problem bestaar i at finde ud af, hvornaar en forelagt funktion antager værdien nul. Har vi funktionen y = F(x), altsaa en funktion af een variabel, kan vi spørge om, for hvilke værdier af x vi faar:

$$F(x) = 0$$

Naar x er fundet, siger vi at vi har fundet en rod i ligningen F(x) = 0.

Man kan ogsaa have flere samtidige ligninger med ligesaa mange ubekendte, f.eks.:

$$F(x1, x2, x3) = 0$$
$$G(x1, x2, x3) = 0$$
$$H(x1, x2, x3) = 0$$

Her skal vi finde et saadant talsæt: x1, x2, x3, at de tre funktioner F, G og H alle bliver nul. Hvis de tre funktioner er lineære udtryk i de uafhængige variable, altsaa:

```
F := a0 + a1*x1 + a2*x2 + a3*x3;
```

og analogt for G og H, har vi tre lineære ligninger med tre ubekendte, og vi kan løse dem som i kapitel 2. Men er funktionerne ikke lineære, f.eks.:

```
F := a0 + a1*x1↑2 + a2*sin(x2) + a3*x1*x3;
```

siger vi, at der foreligger et sæt ulineære ligninger, som skal løses.

I det følgende ser vi først paa funktioner af een variabel og derefter paa flere funktioner af flere variable. Men der er ogsaa en anden maade at klassificere problemet paa. Hvis den foreliggende funktion er særlig simpel, f.eks. et polynomium af lav grad, findes der specielle simple løsningsmetoder. De klassiske metoder for andengradsligninger og tredje-

gradsligninger er vist nedenfor.

Men i de fleste praktisk forekommende tilfælde er funktionerne langtfra simple. Som illustration kan vi tænke os, at de tre ubekendte, x1, x2 og x3 i eksemplet ovenfor er mængderne af kulbrinte, luft og ilt, som skal sammenblandes for at producere en vis mængde syntesegas i en adiabatisk reformer. Hvis vi gætter paa et sæt værdier af de tre ubekendte, kan vi med et passende kompliceret program beregne værdien af F, G og H. Disse kan f.eks. være:

F: Fejl i varmebalancen.
G: Fejl i brint-kvælstofforholdet.
H: Fejl i den producerede gasmængde.

Vi maa da variere paa x1, x2 og x3, indtil F, G og H bliver nul. Ved systematisk variation af de ubekendte og tilhørende beregning af fejlene kan man efterhaanden faa udforsket de tre ubekendte, komplicerede funktioner og saaledes ved gentagne beregninger (iterationer) naa frem til løsningen. Iterationsmetoderne beskrives i afsnit 4.3 for een ubekendt og i afsnit 4.5 for flere ubekendte.

4.1. Andengradsligninger.

Proceduren QUAREQ1 finder de to løsninger i andengradsligningen:

$$A \times x\uparrow 2 + B \times x + C = 0$$

De seks formelle parametre i proceduren er dels de tre koefficienter A, B og C, og dels de to søgte rødder: x1 og x2. Den sidste parameter er etiketten: ERROR, hvortil der hoppes, hvis diskriminanten:

$$D := B\uparrow 2 - 4 \times A \times C;$$

er negativ. I saa fald er x1 og x2 komplekse.

Deklarationen af QUAREQ1 er:

```
procedure QUAREQ1 (A, B, C, x1, x2, ERROR);
value A, B, C;
real A, B, C, x1, x2;
label ERROR;
begin
   real D;
   if A ≠ 0 then go to L1;
   if B = 0 then go to ERROR;
   x1 := x2 := - C/B;
   go to L2;
L1: D := B↑2 - 4×A×C;
   if D < 0 then go to ERROR;
   x1 := if B < 0 then -1 else 1;
   x1 := (-B - x1×sqrt(D))/2/A;
   x2 := C/A/x1;
L2: end of QUAREQ1;
```

4.2. Tredjegradsligninger.

Ogsaa for tredjegradsligninger:

$$A \times x\uparrow 3 + B \times x\uparrow 2 + C \times x + D = 0$$

kan man angive en simpel løsningsmetode uden iteration. Deklarationen for en saadan procedure, CUBEQ1, er:

```
procedure CUBEQ1 (A, B, C, D, x1, x2, x3, comp);
value A, B, C, D;
real A, B, C, D, x1, x2, x3;
boolean comp;
begin
   real a, b, AA, BB, DD, p, q, r, SQ, cosfi, fac, fi3, p3;
   real procedure cubrt (x);
   value x;
   real x;
```

```
    cubrt := sign(x)×(abs(x))↑(1/3);
    p := B/A;
    p3 := p/3;
    q := C/A;
    r := D/A;
    a := q - p↑2/3;
    b := (2×p↑3 - 9×p×q + 27×r)/27;
    DD := b↑2/4 + a↑3/27;
    comp := DD > 0;
    if DD < 0 then go to LA;
    SQ := sqrt (DD);
    AA := cubrt( - b/2 + SQ);
    BB := cubrt(- b/2 - SQ);
    x1 := AA + BB - p3;
    if ¬ comp then x2 := x3 := - (AA + BB)/2 - p3;
    go to LB;
LA: cosfi := - b/2/sqrt(- a↑3/27);
    fi3 := if cosfi = 0 then 0.52359878
    else arctan (sqrt(1-cosfi↑2)/cosfi)/3;
    fac := 2×sqrt(- a/3);
    x1 := fac×cos(fi3) - p3;
    x2 := fac×cos(fi3 + 2.0943951) - p3;
    x3 := fac×cos(fi3 + 4.1887902) - p3;
LB: end of CUBEQ1;
```

De formelle parametre er de fire koefficienter: A, B, C og D, de tre søgte rødder: x1, x2 og x3, samt en boolean: comp. Hvis der er tre reelle rødder, bliver de beregnet og afleveret som x1, x2 og x3. Hvis der kun er een reel rod, bliver den beregnet og gemt i x1, og comp sættes til true. Med tre reelle rødder sættes comp til false.

Der er ikke gjort noget særligt forsøg paa at opnaa god regnenøjagtighed, og proceduren indgaar saa vidt vides ikke i noget rutineprogram.

4.3. Iterationsmetoder for een Ubekendt.

De anførte procedurer for andengradsligninger og tredjegradsligninger har den fordel, at der ikke benyttes iteration. Man faar altid det ønskede svar efter gennemløb af formlerne netop een gang. Til gengæld er formlerne ikke helt simple, især for tredjegradsligninger.

Ogsaa for fjerdegradsligninger er det muligt at angive et formelsæt uden iteration. Men det er endnu mere kompliceret. Her vil man i de fleste tilfælde foretrække en iterativ metode.

Som et typisk eksempel ser vi nu paa talmaterialet i tabellen paa side 30. Hvis vi paa program PA-8 beregner et fjerdegradspolynomium, som udtrykker y som funktion af x, bliver koefficienterne heri:

Potens	Koefficient
0	$6.6517485_{10}2$
1	3.3595950
2	$8.4959053_{10}-3$
3	$-7.1126416_{10}-7$
4	$-2.5486772_{10}-10$

Polynomiet giver varmeindholdet (entalpien) af metan som funktion af temperaturen (i grader Kelvin). Hvis en bestemt gasmængde efterhaanden optager varme, f.eks. ved at strømme igennem rør, der opvarmes udefra, vil temperaturen af gassen stige. Vi kan da faa brug for at beregne den temperatur, x, der svarer til en given entalpi, y0. Vi skal bestemme x saaledes at:

$$F(x) = y0$$

eller x skal findes som rod i ligningen:

$$F(x) - y0 = 0$$

Naar F(x) som her er et fjerdegradspolynomium i x og y0 er et kendt tal, skal vi altsaa løse en fjerdegradsligning. Som eksempel sætter vi y0 = 16580, hvilket svarer til at x ligger imellem 1200 og 1300. Ved

prøve finder vi, at

$$x = 1273.35$$

giver den rigtige værdi af y0. I de følgende afsnit viser vi forskellige metoder til den iterative rodbestemmelse. Her bemærker vi, at en fjerdegradsligning kan have op til 4 forskellige reelle rødder, og det er netop tilfældet her. De fire rødder er:

$$-7066.86$$
$$-1530.67$$
$$1273.35$$
$$4533.47$$

Vi ser, at de tre andre rødder ligger langt udenfor det omraade, for hvilket polynomiet er udregnet, og de maa derfor i dette tilfælde karakteriseres som helt fremmede rødder, som vi slet ikke har interesse i at kende. Det ville derfor være helt urimeligt at bruge en metode til en komplet løsning af en fjerdegradsligning, som gav os alle fire rødder, hvorefter de tre forkerte rødder skulle kasseres. Her er den iterative metode langt at foretrække.

I den nedenstaaende gennemgang af forskellige metoder til iterativ rodbestemmelse følger vi en lidt anden fremgangsmaade end hidtil. Hver procedure bliver forklaret ud fra den brug, som man ønskede at gøre af den paa det tidspunkt, den blev lavet. Paa denne maade faar vi en historisk forklaring paa de strategiske fordele og ulemper, som hver procedure frembyder. Fremstillingen afsluttes med en oversigt over alle strategiske finesser og med en procedure, hvori alle disse er medtaget.

Som regneeksempel bruger vi det ovennævnte entalpipolynomium, idet vi laver en særlig procedure, YPOL, som har deklarationen:

```
real procedure YPOL;
begin
   real z;
   z := 665.17485 + x×(3.359595 + x×(8.4959053₁₀-3 + x×(-7.1126416₁₀-7
        - x×2.5486772₁₀-10))) - y0;
```

```
   trykvr;
   tryk({-ndddd.dddddd}, x, z);
   YPOL := z
end YPOL;
```

Der er to globale variable: x og y0, og værdien af YPOL er fjerdegradspolynomiet for den aktuelle værdi af x minus den aktuelle værdi af y0, som er den entalpiværdi, som vi skal ramme. Hvert kald af YPOL giver ogsaa udskrift af x og polynomieværdien minus y0. Paa denne maade kan vi let følge iterationen.

4.3.1. Proceduren ROOT1. Grundprincippet i de fleste iterative metoder til rodbestemmelse er den saakaldte Newton-Raphsonske metode, der er illustreret paa figur 1. Se ogsaa Frøberg (1962), side 16 ff.

Figur 1 viser en funktion y = F(x), samt et punkt (x1, y1) paa kurven. I Newton-Raphsons metode bruger vi som tilnærmelse til F(x) tangenten i punktet (x1, y1). Kaldes differentialkvotienten i dette punkt for alfa:

```
alfa := dF/dx
```

bliver ligningen for tangenten:

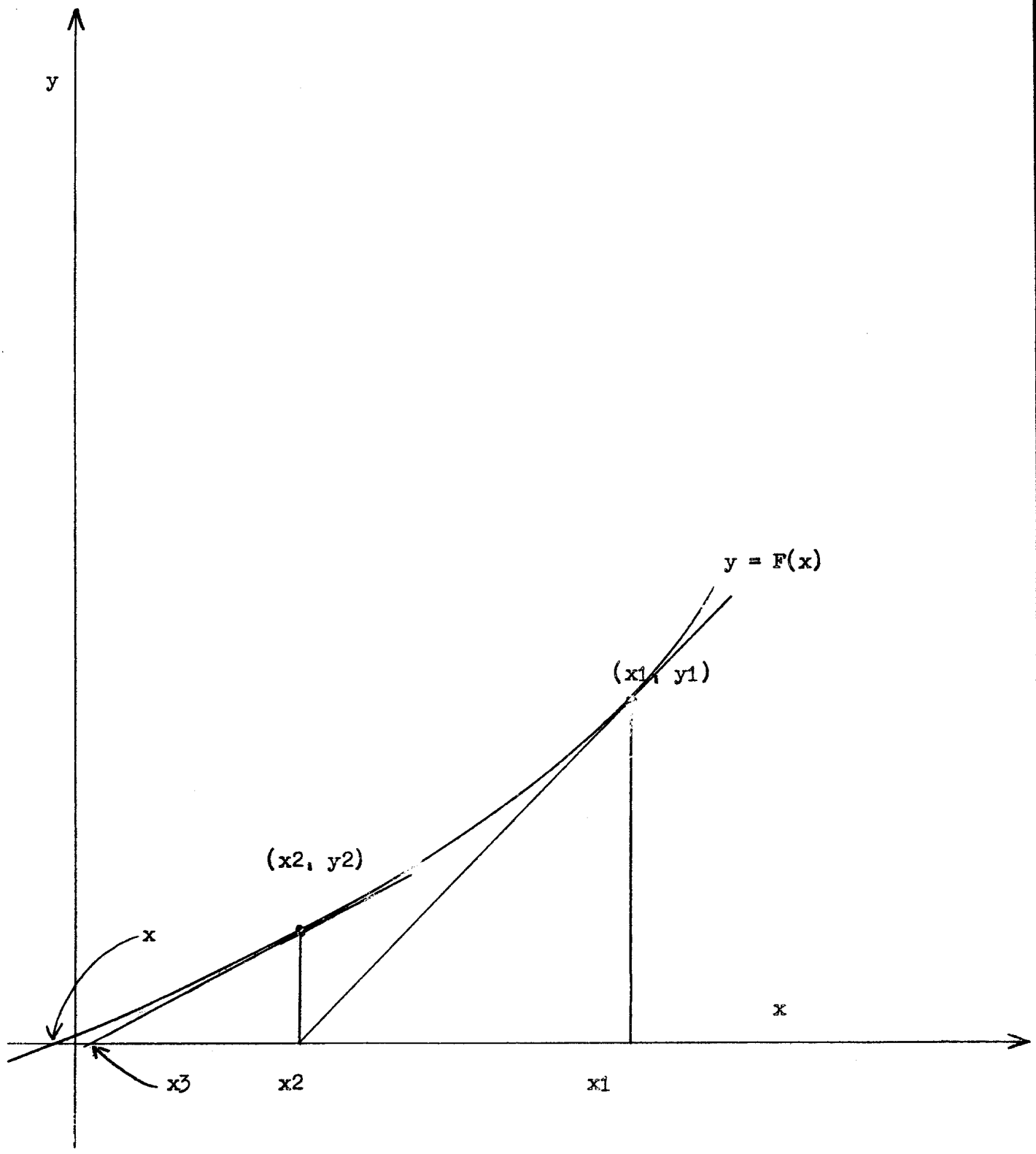
```
y := y1 + alfa*(x-x1);
```

Vi sætter y = 0 og finder x af ligningen. Dette giver den næste tilnærmelse, x2, til den søgte rod:

```
x2 := x := x1 - y1/alfa;
```

Vi kan ogsaa udtrykke resultatet som den tilvækst, delx, man skal addere til x1 for at faa x2:

```
delx := - y1/alfa;
```



Figur 1

Newton-Raphsons Metode

I punktet x2 skal vi derefter beregne værdien y2 = F(x2), samt den nye værdi af alfa i x2. Iterationen fortsættes, indtil delx bliver tilpas lille.

I denne version skal vi for hver ny x-værdi beregne baade funktionsværdien, y, og differentialkvotienten, dy/dx. Det sidste er nogenlunde overkommeligt, hvis funktionen er et polynomium, men hvis F(x) er et eller andet kompliceret udtryk, eventuelt en større procedure eller et programstykke, vil man kun kunne finde dF/dx ved en besværlig numerisk metode.

Da differentialkvotienten ikke behøver at være bestemt med nogen stor nøjagtighed, kan man nøjes med at beregne differenskvotienten:

(y2-y1)/(x2-x1)

og bruge denne som tilnærmelse til alfa. Denne metode kaldes regula falsi metoden og er illustreret paa figur 2. Da vi ikke beregner differentialkvotienten i punktet x1, er det nødvendigt at vælge punktet x2 arbitrært for at komme igang. Tilvæksten, delx, i x fra x2 til x3 findes da som:

```
delx := - y2/alfa
```

hvori vi indsætter differenskvotienten og faar:

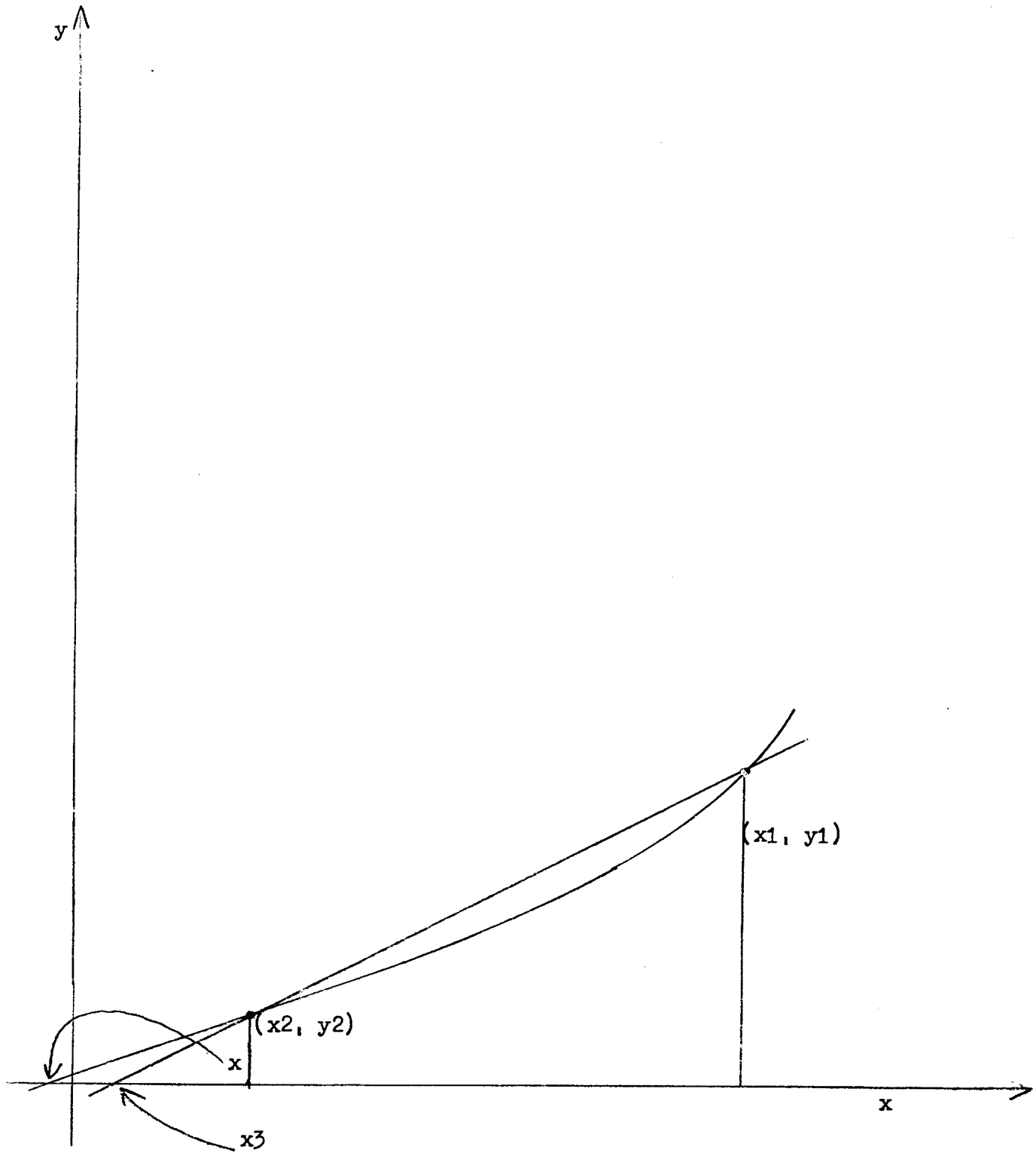
```
delx := - y2/(y2-y1)*(x2-x1)
```

Da x2 - x1 er den forrige værdi af delx, kan vi ogsaa skrive:

```
delx := delx*ynew/(yold-ynew);
```

Her er ynew den sidst beregnede y-værdi (y2) og yold den næstsidst beregnede (y1).

Den her skitserede regula falsi metode er anvendt i proceduren ROOT1, der har deklarationen:



Figur 2

Regula falsi Metoden

```
procedure ROOT1 (y, yold, ynew, x, delx, del0x, epsilon, FOUND);
real y, yold, ynew, x, delx, del0x, epsilon;
label FOUND;
begin
   real A;
   ynew:=y;
   if epsilon - abs(ynew) ≥ 0 then go to FOUND;
   if yold=0 then
   begin
      A := del0x;
      go to L1
   end;
   A := delx×ynew/(yold - ynew);
L1: delx := A;
   x := x + delx;
   yold := ynew
end ROOT1;
```

Som et eksempel paa brugen af ROOT1 viser vi nedenfor program d-180, der finder roden i den ovennævnte procedure YPOL.

```
Program d-180. Prøve af ROOT1.
begin
   real x, y0, yold, ynew, delx;
   comment library ROOT1;
   comment library YPOL;
   trykvr;
   tryktekst(‡<      x              y‡);
   trykvr;
   x := 1000;
   y0 := 16580;
   yold := 0;
B: ROOT1(YPOL, yold, ynew, x, delx, 10, 0.01, F1);
   go to B;
F1: YPOL
end program;
```

Resultatudskriften er:

x	y
1000.000000	-5025.456848
1010.000000	-4852.993896
1291.393414	351.671326
1272.380085	-18.845520
1273.347153	-0.057007
1273.350086	0.000000
1273.350086	0.000000

ROOT1 har følgende parametre:

real y:	Den undersøgte funktion.
real yold:	Bruges af proceduren til at gemme den næstsidst beregnede y-værdi. Før første kald af ROOT1 skal brugeren sætte yold lig med nul.
real ynew:	Bruges af proceduren til at gemme den sidst beregnede y-værdi.
real x:	Den uafhængige variable. Før første kald skal brugeren have indsat en passende startværdi.
real delx:	Bruges af proceduren til at gemme den sidst beregnede tilvækst i x.
real del0x:	Brugeren skal opgive en passende værdi, som proceduren kan bruge første gang, x skal forøges.
real epsilon:	Naar den numeriske værdi af y er blevet mindre end epsilon, anser proceduren roden for fundet.
label FOUND:	Naar roden er fundet, hoppes til denne etikette.

Proceduren er skrevet paa aaben form, d.v.s. vi kommer ud af proceduren for hver ny x-værdi:

```
B:    ROOT1(.........);
      ......
      ......
      ......
      go to B;
```

I den viste udførelse af program d-180 er y-funktionen deklareret som en særlig procedure, men hvis man ikke ønsker dette, kan beregningen af den nye y-værdi programmeres paa pladsen mellem ROOT1-kaldet og returhoppet: go to B.

ROOT1 virker iøvrigt saaledes. Først sættes ynew lig med den nye y-værdi. Hvis den numeriske værdi af ynew er mindre end epsilon, hoppes til etiketten FOUND. Hvis yold = 0, er dette tegn paa, at det er det første kald af ROOT1, og tilvæksten delx skal da blot sættes lig med del0x. Ellers beregnes delx efter den ovennævnte formel:

```
delx := delx×ynew/(yold-ynew);
```

Endelig adderes delx til x, yold sættes lig ynew, og proceduren er færdig i denne omgang.

Proceduren er lavet paa et tidligt tidspunkt, og er derfor skrevet i et noget ubehjælpsomt ALGOL. F.eks. kan sætningerne:

```
    if yold = 0 then
    begin
       A := del0x;
       go to L1
    end;
    A := delx×ynew/(yold-ynew);
L1: delx := A;
```

skrives noget kortere som:

```
    delx := if yold = 0 then del0x
    else
    delx×ynew/(yold-ynew);
```

Herved undgaar vi ogsaa etiketten L1 og hjælpevariablen A.

ROOT1 bør kun opfattes som et noget ufuldkomment første forsøg paa at lave en iterativ rodbestemmelsesprocedure. For skikkelige funktioner, som den ovennævnte YPOL, der ikke er ret meget ulinesr indenfor det normale definitionsomraade, er proceduren udmærket anvendelig og bruges endnu i flere rutineprogrammer. Ulemper ved proceduren omtales senere.

4.3.2. Proceduren ROOT5. I nogle tilfælde kan det være praktisk at have rodbestemmelsesproceduren paa lukket form, saaledes at man først kommer ud af proceduren, naar roden er fundet. Proceduren ROOT5 er lavet paa denne maade, men virker iøvrigt som ROOT1. Deklarationen er:

```
procedure ROOT5 (y, x, del0x, eps);
value del0x, eps;
real y, x, del0x, eps;
begin
   real yold, ynew, delx;
   ynew := y;
   yold := 2×ynew;
   delx := del0x;
L: if eps - abs (ynew) ≥ 0 then go to EX;
   delx := delx×ynew/(yold - ynew);
   x := x + delx;
   yold := ynew;
   ynew := y;
   go to L;
EX: end;
```

Et eksempel paa brugen af ROOT5 er vist i program d-181, der er ganske analogt med d-180:

```
Program d-181. Prøve af ROOT5.
begin
   real x, y0;
   comment library ROOT5;
   comment library YPOL;
```

```
   trykvr;
   trykteksт({<      x              y});
   x := 1000;
   y0 := 16580;
   ROOT5(YPOL, x, 10, 0.01);
   YPOL
end program;
```

Resultatudskriften er:

```
      x              y
 1000.000000 -5025.456848
 1009.999998 -4852.993896
 1291.393410   351.671265
 1272.380085   -18.845520
 1273.347153    -0.057007
 1273.350086     0.000000
 1273.350086     0.000000
```

Parametrene i ROOT5 er: y, x, del0x og eps med samme betydning som i ROOT1. Ogsaa her er resultatet af afprøvningen tilfredsstillende. ROOT5 er indbygget i flere rutineprogrammer.

4.3.3. Procedurerne ROOT6 og ROOT7. Hvis den undersøgte funktion y = F(x) ikke har et pænt og skikkeligt forløb, kan der opstaa vanskeligheder ved rodbestemmelsen. Hvis udtrykket F(x) bliver udefineret, naar x ikke ligger meget tæt ved den søgte rod, er det nødvendigt at indføre en nedre og øvre grænse for x: xmin og xmax, saaledes at vi kun spørger om F(x) for saadanne værdier af x, hvor F(x) er defineret. Denne situation kan f.eks. opstaa i forbindelse med løsning af differentialligninger med topunktsrandbetingelser (se bind 3).

Procedurerne ROOT6 og ROOT7 har været benyttet til løsning af dette problem. Deklarationen for ROOT6 er:

```
procedure ROOT6(y, x, del0x, xmin, xmax, delmax, eps);
value del0x, xmin, xmax, delmax, eps;
real y, x, del0x, xmin, xmax, delmax, eps;
begin
   real yold, ynew, delx;
   ynew := y;
   yold := 2×ynew;
   delx := - del0x×sign(ynew);
L: if ynew > 0 ∧ x < xmax then xmax := x;
   if ynew < 0 ∧ x > xmin then xmin := x;
   if xmax - xmin < eps then go to EX;
   delx := delx×ynew/(yold-ynew);
   if abs(delx) > delmax then delx := sign(delx)×delmax;
   if x + delx ≥ xmax ∨ x + delx ≤ xmin then
   delx := 0.5×(xmax+xmin) - x;
   x := x + delx;
   yold := ynew;
   ynew := y;
   go to L;
EX: end of ROOT-6;
```

I ROOT6 har vi foruden de tidligere parametre: y, x, del0x og eps de nye parametre: xmin, xmax og delmax. Brugeren skal opgive den nedre og øvre grænse: xmin og xmax. Vi forudsætter, at funktionen F(x) er stadig voksende i intervallet, saaledes at x-værdier med en negativ y-værdi er mindre end den søgte rod, og x-værdier med positiv y-værdi er større end roden. De to grænser xmin og xmax er deklareret for value, og for hver ny beregning af x og y undersøger proceduren, om grænserne kan indsnævres. Hvis y er positiv og x er mindre end xmax, sættes xmax lig med x, og hvis y er negativ og x større end xmin, sættes xmin lig med x. Som kriterium for at roden er fundet, benytter vi nu, at xmax - xmin er mindre end eps, idet eps nu betegner tolerancen paa x, ikke paa y, som i ROOT1 og ROOT5.

Som yderligere forsigtighedsforanstaltninger kontrollerer vi først, at den numeriske værdi af tilvæksten, delx, ikke overskrider en tilladelig størrelse, delmax, der er en ny parameter. Gør den det, sættes delx lig med delmax, med bevarelse af fortegnet. Derefter undersøger vi, om den nye x-værdi kommer til at ligge udenfor intervallet fra xmin til xmax.

Gør den det, sættes x lig med middelværdien af xmin og xmax.

Iøvrigt benyttes den sædvanlige regula falsi formel.

Det viser sig nu, at proceduren ROOT6 i nogle tilfælde virker tilfredsstillende, i andre ikke. Forklaringen er den, at for at faa den nødvendige indsnævring af intervallet fra xmin til xmax, og saaledes at begge grænser bevæger sig mod roden, maa man forlange, at delx hele tiden skifter fortegn, idet vi nærmer os roden fra begge sider. For en voksende funktion vil dette kun ske, hvis differentialkvotienten af anden orden er positiv. Er den negativ, vil vi nærme os roden ensidigt, og vi kan ikke faa tilfredsstillet udhopskriteriet. De to strategier: regula falsi formlen og intervalindsnævring ved iagttagelse af funktionens fortegn kan derfor ikke uden videre bruges samtidigt.

I proceduren ROOT7 har vi derfor forladt regula falsi metoden til fordel for halvering af intervallet. Deklarationen er:

```
procedure ROOT7(y, x, del0x, xmin, xmax, delmax, eps, cfirst, clast, cact,
          trouble);
value del0x, xmin, xmax, delmax, eps, cfirst, clast;
boolean trouble;
integer cfirst, clast, cact;
real y, x, del0x, xmin, xmax, delmax, eps;
begin
   integer type, clow, chigh, cnew;
   real delx, ynew, yoldl, yoldh;
   yoldl := -1_10100;  yoldh := -yoldl;
   type := 0;
   trouble := false;
   clow := chigh := cfirst;
LL: ynew := y;
   cnew := cact;
   if ynew < 0 then
   begin
      if abs(cnew-clast) < abs(clow-clast)
      ∨ cnew = clow ∧ ynew > yoldl then
      begin
         clow := cnew;
         yoldl := ynew;
```

```
            xmin := x
         end if improvement;
         type := if type ≤ 1 then 1 else 3
      end if negative
      else
      begin
         if abs(cnew-clast) < abs(chigh-clast)
         ∨ cnew = chigh ∧ ynew < yoldh then
         begin
            chigh := cnew;
            yoldh := ynew;
            xmax := x
         end if improvement;
         type := if type = 0 ∨ type = 2 then 2 else 3
      end if positive;
      delx := if type = 3 then
      0.5*(xmax+xmin) - x
      else
      if type = 1 then del0x else - del0x;
      if type = 3 ∧ xmax - xmin < eps then go to EX;
      if abs(delx) > delmax then delx := sign(delx)*delmax;
      if x + delx ≥ xmax ∨ x + delx ≤ xmin then
      begin
         if type < 3 then trouble := true;
         delx := 0.5*(xmax+xmin) - x
      end if out of range;
      x := x + delx;
      go to LL;
EX: end of ROOT7;
```

Foruden de tidligere parametre: y, x, del0x, xmin, xmax, delmax og eps har vi faaet fire nye parametre:

```
integer cfirst, clast, cact;
boolean trouble;
```

De nye heltalsparametre refererer direkte til det ovennævnte problem

med topunkts-randbetingelser ved løsning af differentialligninger. Lad os antage, at y fremkommer som slutværdien af en funktion, som findes ved løsning af en differentialligning i intervallet fra A til B. Værdien af y er altsaa f(B). Hver ny værdi af x giver en ny startværdi, f(A), og løsningen af differentialligningen giver f(B), som altsaa er y. Hvis x er meget forskellig fra den søgte rod, kan det ske, at differentialligningen ikke kan løses i intervallet fra A til B fordi differentialkvotienten bliver udefineret undervejs. Vi antager, at integrationsvejen fra A til B er inddelt i et vist antal trin, saaledes at A svarer til cfirst og B til clast. For hver ny x-værdi, som prøves, skal y-proceduren give baade y-værdien og nummeret, cact, som viser hvor langt integrationen kunne gaa inden differentialkvotienten blev udefineret.

Ligesom i ROOT6 bliver grænserne xmin og xmax hele tiden reguleret, men nu saaledes at xmin sættes lig x hvis y er negativ og:

1. cact er bedre end den hidtil bedste værdi, clow, af cact for xmin eller
2. cact er lig med clow og y er større end den hidtil største, negative y-værdi (yoldl).

Paa lignende maade reguleres xmax nedad.

Saalænge proceduren kun har set y-værdier med samme fortegn, ændres x med den faste skridtlængde del0x for y negativ og - del0x for y positiv, idet vi ligesom i ROOT6 forudsætter, at y er voksende. Saasnart der er fundet et interval med modsat fortegn af y-værdierne i de to endepunkter, findes næste x ved halvering af intervallet.

I program d-183 demonstrerer vi brugen af ROOT6 og ROOT7 til at finde roden x = 0.707107 i udtrykket:

```
y := x↑2 - 0.5;
```

Vi lader ogsaa programmet beregne en værdi af cact, der er desto større jo nærmere vi er ved roden.

Programmet er:

Program d-183. Prøve af ROOT6 og ROOT7.

```
begin
   boolean trouble;
   integer cact, i;
   real x;
   comment library ROOT6;
   comment library ROOT7;
   real procedure Y;
   begin
      real z;
      z := x↑2 - 0.5;
      cact := 10 - entier(10×abs(x-sqrt(0.5)));
      trykvr;
      tryk({-nd.dddddd}, x);
      tryk({-n.ddd10-dd}, z);
      tryk({-nddd}, cact);
      Y := z
   end Y;
   for i := 6, 7 do
   begin
      trykvr;
      tryktekst({< x          y          cact});
      x := 0.2;
      if i = 6 then
      ROOT6(Y, x, 0.1, 0, 1, 0.2, 1_10-5)
      else
      ROOT7(Y, x, 0.1, 0, 1, 0.2, 1_10-5, 0, 10, cact, trouble);
      Y;
      if i = 7 ∧ trouble then tryktekst({<Fejl})
   end for i
end program;
```

Resultatudskriften er:

x	y	cact
0.200000	-4.600 $_{10}$-1	5
0.300000	-4.100 $_{10}$-1	6
0.500000	-2.500 $_{10}$-1	8
0.700000	-1.000 $_{10}$-2	10
0.708333	1.736 $_{10}$-3	10
0.707101	-8.756 $_{10}$-6	10
0.707107	-7.451 $_{10}$-9	10
0.707107	-1.863 $_{10}$-9	10
0.707720	8.677 $_{10}$-4	10
0.707413	4.337 $_{10}$-4	10
0.707107	1.881 $_{10}$-7	10
0.707107	1.881 $_{10}$-7	10

x	y	cact
0.200000	-4.600 $_{10}$-1	5
0.300000	-4.100 $_{10}$-1	6
0.400000	-3.400 $_{10}$-1	7
0.500000	-2.500 $_{10}$-1	8
0.600000	-1.400 $_{10}$-1	9
0.700000	-1.000 $_{10}$-2	10
0.800000	1.400 $_{10}$-1	10
0.750000	6.250 $_{10}$-2	10
0.725000	2.562 $_{10}$-2	10
0.712500	7.656 $_{10}$-3	10
0.706250	-1.211 $_{10}$-3	10
0.709375	3.213 $_{10}$-3	10
0.707812	9.985 $_{10}$-4	10
0.707031	-1.068 $_{10}$-4	10
0.707422	4.457 $_{10}$-4	10
0.707227	1.694 $_{10}$-4	10
0.707129	3.128 $_{10}$-5	10
0.707080	-3.777 $_{10}$-5	10
0.707104	-3.245 $_{10}$-6	10
0.707117	1.402 $_{10}$-5	10
0.707111	5.385 $_{10}$-6	10
0.707111	5.385 $_{10}$-6	10

Den første serie udskrifter er fra ROOT6 (12 iterationer) og den anden fra ROOT7 (22 iterationer). Som ventet er brugen af ROOT7 langsommere end brug af ROOT6, da halveringen kun giver eet betydende ciffer i totalssystemet ad gangen. Iøvrigt ser man, at ROOT6 har fundet det rigtige resultat allerede efter 8 iterationer:

Iteration:	x	y
7	0.707107	$-7.451_{10}-9$
8	0.707107	$-1.863_{10}-9$

Vi maa have:

xmin: 0.707107
xmax: 0.708333

Dette interval er større end den tilladte tolerance ($1_{10}-5$), men der sker nu det, at vi paa grund af afrundingsfejl i y-værdierne faar en ny x-værdi, som ligger ganske lidt udenfor intervallet fra xmin til xmax. Derfor tages middelværdien 0.707720 af xmin og xmax, og vi faar efterhaanden ogsaa bragt xmax ned. Dette illustrerer iøvrigt en alvorlig risiko ved alle de her nævnte procedurer, som bruger regula falsi metoden i den her benyttede form:

```
delx := delx*ynew/(yold-ynew);
```

Hvis to x-værdier afviger saa lidt fra hinanden, at yold og ynew bliver eksakt ens, faar man division med nul. Selv om det ikke gaar saa galt, kan der blive saa stor afrundingsfejl paa yold - ynew, at differenskvotienten:

```
(ynew-yold)/delx
```

bliver meget forkert. Det giver det paradoksale resultat, at jo nærmere vi er ved roden, desto daarligere bestemt bliver den yderligere korrektion paa x.

4.3.4. Valg af strategi. Vi vil nu diskutere forskellige af de strategiske finesser i rodbestemmelsesprocedurerne og søge at naa frem til en paalidelig formulering.

1. Iterationsprincip. Vi fastholder regula falsi metoden, da den er hurtigere end halveringsmetoden, og da Newton-Raphson med direkte beregning af differentialkvotienten kun sjældent kan bruges. Men vi maa sørge for at undgaa division med nul eller fejl paa grund af daarlig nøjagtighed. Vi skriver derfor tilvæksten som:

```
delx := - ynew/alfa;
```

og beregner kun differenskvotienten alfa:

```
alfa := (ynew-yold)/delx;
```

hvis den numeriske værdi af ynew - yold er større end $1_{10}-4$ gange den numeriske værdi af yold. Faktoren $1_{10}-4$ er valgt arbitrært, men kan maaske vælges mere systematisk.

2. Definitionsomraade. I nogle tilfælde er det vigtigt, at den uafhængige variable, x, holder sig indenfor et bestemt omraade (fra xmin til xmax), men i andre tilfælde er dette ligegyldigt. For ikke at spilde for mange parametre herpaa, nøjes vi med en enkelt:

```
boolean outside;
```

Ønsker man ikke at bruge denne kontrol, sætter man den aktuelle parameter til false. Skal vi kontrollere, at xmin $\leq$ x $\leq$ xmax, kan vi deklarere en separat procedure:

```
boolean procedure outside;
outside := x < xmin ∨ x > xmax;
```

Naar iterationsproceduren har fundet en ny x-værdi ved:

```
x := x + delx;
```

kalder den outside (der er kaldt ved name) til kontrol. Denne metode har

ogsaa vist sig praktisk i forskellige optimeringsprocedurer (se kapitel 5). Paa denne maade kan vi ogsaa let indføre andre betingelser, f.eks. at en anden funktion af x skal ligge i et bestemt interval.

3. Skridtkontrol. Det kan være praktisk at lægge en øvre grænse paa tilvæksten, delx, især ved drilagtige funktioner. Vi kan derfor indføre en real parameter, maxstepfac, som angiver hvor mange gange værdien af delx højst maa være større end førstegangstilvæksten, del0x.

Hvis man kommer udenfor definitionsomraadet eller maxstepfac overskrides, halveres delx, indtil det passer.

4. Konvergenskriterium. Vi har set eksempler paa, at kriteriet for at roden var fundet var det, at y-værdien blev mindre end en opgivet tolerance (ROOT1 og ROOT5). I ROOT6 var vi mere forsigtige og lod et sæt lokale grænser, xmin og xmax, indsnævres, indtil deres afstand var mindre end tolerancen, som derfor refererer til x-værdien. Da en tolerance paa x altid kan omregnes til en tolerance paa y ved at multiplicere med alfa, er det ikke af større principiel betydning, hvilken af de to man vælger. Erfaringen med brug af rodprocedurer og optimeringsprocedurer har vist, at det i praksis er fuldt ud tilfredsstillende at regne tolerancen paa x (hvad der matematisk set ogsaa er det naturligste), samt at regne konvergensen for indtruffet, naar den numeriske værdi af delx bliver mindre end tolerancen. Det sidste er knapt saa tilfredsstillende ud fra et matematisk synspunkt, idet man kan risikere at konvergere mod en x-værdi, der slet ikke er en rod, men blot et lokalt minimum. En rationel løsning herpaa er knyttet til behandlingen af det følgende punkt:

5. Fejlkriterium. Visse funktioner kan være saa umedgørlige, at det vil være rimeligt at tillade, at rodbestemmelsesproceduren giver fortabt og hopper til en fejletikette. Vi maa da give en klar definition af mængden af de funktioner, som proceduren skal kunne behandle, og hvilke den maa give op for.

Naur (1964) har givet et eksempel paa automatisk bedømmelse af en række rodbestemmelsesprocedurer, udarbejdet af studenter. Opgaven var defineret saaledes, at F(x) var givet i det lukkede interval $a \leq x \leq b$. Det var forudsat, at F(a) og F(b) havde modsat fortegn, og var dette ikke tilfældet, skulle proceduren konkludere, at der ingen rod var.

Vi har altsaa en kontinuert funktion defineret i et opgivet lukket interval og med modsat fortegn i de to endepunkter. Der vil derfor altid være mindst een rod. Spørgsmaalet er nu, om det er en realistisk maade

at definere opgaven paa. Erfaringen viser, at man ved praktisk forekommende problemer oftere er i den situation, at man kan give et omtrentligt skøn over rodens placering, men kun med større besvær kan angive et lukket interval indenfor hvilket roden med sikkerhed findes, og hvor der er modsat fortegn af funktionen i de to endepunkter. I stedet for at starte undersøgelsen i de to endepunkter med at beregne F(a) og F(b) og efterhaanden indsnævre afstanden fra a til b, indtil roden er indkredset, er det mere realistisk at starte undersøgelsen i et enkelt punkt:

```
x := xstart;
```

og derefter undersøge et punkt i nærheden:

```
x := xstart + del0x;
```

og saa køre løs med regula falsi metoden ud fra disse to punkter. Hvis F(a) og F(b) skal inddrages i undersøgelsen, risikerer man let, at a og b maa vælges saa langt væk fra roden, at funktionen slet ikke kan beregnes, fordi den antager ekstreme og urealistiske værdier.

Hvis vi fastlægger den strategi, at vi begynder med xstart og xstart + del0x, er det rimeligt at forudsætte, at funktionen er monotont voksende eller aftagende i omraadet heromkring. Med denne forudsætning vil vi altid nærme os til roden med regula falsi metoden. Er forudsætningen ikke opfyldt, d.v.s. at der optræder et lokalt minimum eller maksimum, risikerer vi at konvergere mod dette ekstremumspunkt.

I proceduren ROOT8, der er lavet efter den her skitserede strategi, og som er beskrevet i næste afsnit, gemmer vi fortegnet af den først beregnede differenskvotient og laver fejludhop, hvis der senere optræder en differenskvotient med modsat fortegn. Fejludhop foretages ogsaa, hvis den numeriske værdi af den bedste y-værdi er mange gange (100) større end den numeriske værdi af alfa×delx, fordi vi da maa være konvergeret til et ekstremum, ikke en rod.

6. Andre forholdsregler. I ROOT8 er der yderligere indbygget den simple sikkerhedsforanstaltning, at proceduren hele tiden gemmer den beregnede y-værdi, hvis numeriske værdi er mindst, samt den tilsvarende x-værdi. Den sidste afleveres altid som den fundne rod. Er man meget nervøs for, at proceduren ikke skal konvergere indenfor en rimelig tid,

kan man indbygge en tælling i y-proceduren og foretage fejludhop, hvis antallet af kald overskrider en vis størrelse.

4.3.5. Proceduren ROOT8. Denne er udarbejdet efter den strategi, som blev diskuteret ovenfor. Deklarationen er:

```
procedure ROOT8(y, x, del0x, eps, outside, maxstepfac, ERROR);
value del0x, eps, maxstepfac;
boolean outside;
real y, x, del0x, eps, maxstepfac;
label ERROR;
begin
   boolean first, up, error;
   real xold, xbest, ybest, yold, ynew, delx, alfa, diff;
   xbest := xold := x;
   ybest := ynew := y;
   yold := 2×ynew;
   delx := -del0x×sign(ynew);
   first := true;
   error := false;
   for diff := ynew - yold while
   abs(delx) > eps ∧ yold ≠ 0 ∧ ¬ error do
   begin
      if first ∨ abs(diff) > 1_10-4×abs(yold) then alfa := diff/delx;
      if ¬ first ∧ (alfa < 0 = up) then error := true;
      delx := -ynew/alfa;
      for x := xold + delx while outside
          ∨ abs(delx) > abs(maxstepfac×del0x) do
      delx := 0.5×delx;
      xold := x;
      yold := ynew;
      ynew := y;
      if first then
      begin
         up := (ynew-yold)/delx > 0;
         first := false
```

```
      end if first;
      if abs(ynew) < abs(ybest) then
      begin
         ybest := ynew;
         xbest := xold
      end if better
   end for diff;
   x := xbest;
   if error ∨ abs(ybest) > 100×abs(alfa×delx) then go to ERROR
end ROOT8;
```

Til afprøvning af ROOT8 er lavet program d-186. Det finder roden i 11 forskellige funktioner, hvoraf de 9 første er taget fra Naur (1964) i det ovennævnte studenterprøveprogram, og de to sidste er de samme, som blev afprøvet tidligere i denne bog. Programmet er:

Program d-186. Prøve af ROOT8.

```
begin
   integer i, j;
   real x;
   comment library ROOT8;
   procedure P(xmin, xmax, xstart, eps, root, function);
   value xmin, xmax, xstart, eps, root;
   real xmin, xmax, xstart, eps, root, function;
   begin
      boolean procedure outside;
      outside := x < xmin ∨ x > xmax;
      real procedure Y;
      begin
         Y := function;
         j := j + 1
      end Y;
      i := i + 1;
      j := 0;
      trykvr;
      tryk(‹-nd›, i);
```

```
      x := xstart;
      ROOT8(Y, x, 0.01*(xmax-xmin), eps, outside, 10, ERROR);
      trykml(3);
      go to F1;
ERROR: tryktekst(<< E >>);
F1:   tryk(<-ndddd>, j);
      tryk(<-n.ddddddd10-dd>, trykml(4), root, trykml(4), x, trykml(4), Y)
   end P;
   i := 0;
   trykvr;
   tryktekst(<<Nr.   Iteration   Rod, sand      Rod, ber.          y>);
   trykvr;
   P(0, 2, 0.05, 1_10-6, 0.1, if x<=0.1 then 10*x - 1 else 1_10-20);
   P(-2, 0, -1, 1_10-6, 0, -1+x);
   P(5, 6, 5.5, 1_10-6, 5, 5-x);
   P(-17, -13, -15, 1_10-6, -13, x+13);
   P(0, 20, 10, 1_10-4, 0.95, (x+0.05)↑0.1-1);
   P(0.001, 99.9, 0.5, 1_10-5, 0.01, x+1/x-100.01);
   P(2, 12, 7, 1_10-6, 10, x/10-x↑6/1_108-0.99);
   P(-5, 10, 3, 1_10-6, 1.324718, x↑3-1-x);
   P(-3.2, 20, -1, 1_10-5, -0.4, sin(x)+x/4+0.489418);
   P(0, 1, 0.5, 1_10-5, sqrt(0.5), x↑2-0.5);
   P(0, 2000, 1000, 1_10-2, 1273.35, 665.17485+x*(3.359595+x*
     (8.4959053_10-3+x*(-7.1126416_10-7 -x*2.5486772_10-10)))-16580);
   trykvr
end program;
```

Resultatudskriften er:

Nr.	Iteration	Rod, sand	Rod, ber.	y
1	4	1.0000000 $_{10}$-1	1.0000000 $_{10}$-1	-3.7252903 $_{10}$-9
2 E	16	0.0000000	-6.3137652 $_{10}$-7	-1.0000006
3	10	5.0000000	5.0000000	0.0000000
4	10	-1.3000000 $_{10}$ 1	-1.3000000 $_{10}$ 1	0.0000000
5	12	9.5000000 $_{10}$-1	9.4999994 $_{10}$-1	-1.1175871 $_{10}$-8
6	13	1.0000000 $_{10}$-2	9.9999986 $_{10}$-3	1.3828278 $_{10}$-5
7	9	1.0000000 $_{10}$ 1	9.9999999	-3.7252903 $_{10}$-9
8	11	1.3247180	1.3247180	-3.7252903 $_{10}$-9
9	7	-4.0000000 $_{10}$-1	-3.9999971 $_{10}$-1	0.0000000
10	8	7.0710678 $_{10}$-1	7.0710678 $_{10}$-1	0.0000000
11	7	1.2733500 $_{10}$ 3	1.2733501 $_{10}$ 3	0.0000000

Programmet udskriver antallet af iterationer, den paa forhaand kendte rod (som iøvrigt ikke benyttes), den beregnede rod og den tilsvarende y-værdi. Funktion nr. 2 har givet fejludhop, idet funktionen -1 + x ikke har nogen rod i det opgivne interval fra -2 til 0. Proceduren afleverer x = 0, som den værdi der giver numerisk mindste y. For de øvrige funktioner er det gennemsnitlige antal iterationer 9.

4.4. Serier af Rødder.

I visse tilfælde har man brug for at bestemme roden i en funktion for flere forskellige værdier af en parameter, som indgaar i funktionsudtrykket. Som eksempel tager vi funktionen:

```
y := x↑6 + p×x↑5 - 20×x - 1;
```

For p = 20 har vi roden x = 1. Vi ønsker nu at bestemme roden for p = 20, 21, 22, o.s.v. op til 30. Dersom beregninger af denne art skal udføres manuelt, vil man efterhaanden opbygge en tabel over rødderne og beregne differenserne:

p	Rod	Delta1	Delta2	Delta3
20	1.00000			
		-0.01130		
21	0.98870		+0.00061	
		-0.01069		-0.00005
22	0.97801		+0.00056	
		-0.01013		
23	0.96788			

Hvis vi her forsøger at forudsige roden for $p = 24$ ved at ekstrapolere ud fra differenserne, finder vi:

$$0.96788 + (-0.01013+(+0.00056+(-0.00005))) = 0.95826$$

som ved indsættelse viser sig at ligge meget nær den rigtige rod.

4.4.1. Proceduren ROOT4. Denne ekstrapolation til den næste rod i en serie af rødder ved hjælp af differenserne kan udføres med proceduren ROOT4, hvis deklaration er:

```
procedure ROOT4 (q, n, x0, del0x, xmin, xmax, epsilon, epsy, diffx, x, y);
integer q, n;
boolean epsy;
real x0, del0x, xmin, xmax, epsilon, x, y;
array diffx;
begin
   integer j;
   boolean first;
   real delx, yold, ynew, A;
   procedure incr(z);
   real z;
   begin
      A := z;
L1:   if -, epsy then begin
         if epsilon - abs(A) ≥ 0 ∧ -, first then go to L4 end;
```

```
        if x + A < xmin ∨ x + A > xmax then
        begin
           A := 0.5×A;
           go to L1
        end;
        delx := A;
        x := x + delx
     end incr;
     first := true;
     x := if q=0 then x0 else diffx[0];
     if q≤1 then go to L3;
     delx := 0;
     for j:=2 step 1 until q do
     delx := delx + diffx[j-1];
     incr(delx);
L3:  yold := ynew;
     ynew := y;
     if epsy∧epsilon - abs(ynew)≥0 then go to L4;
     if first then
     begin
        incr(del0x);
        first := false;
        go to L3
     end first;
     delx := if yold ≠ ynew then delx×ynew/(yold-ynew) else -0.5×delx;
     incr(delx);
     go to L3;
L4:  if q=0 then go to L5;
     A := x;
     for j:=1 step 1 until q do
     A := diffx[j-1] := A - diffx[j-1];
     if q<n then diffx[q] := diffx[q-1];
     if q=1 then go to L5;
     for j:=q-1 step -1 until 1 do
     diffx[j] := diffx[j-1];
L5:  diffx[0] := x;
     if q<n then q := q + 1
end ROOT4;
```

Parametrene i ROOT4 er:

integer q:	Et tælleværk, som skal sættes til nul før første kald af ROOT4. Proceduren tæller een frem heri efter hvert kald, indtil den naar n:
integer n:	Den ønskede orden af differenstabellen.
real x0:	Startværdien af x.
real del0x:	Første tilvækst i x.
real xmin:	Nedre grænse for x.
real xmax:	Øvre grænse for x.
real epsilon:	Den tilladelige fejl paa x eller y.
boolean epsy:	Specificeres som sand, hvis epsilon refererer til y og som falsk, hvis epsilon gælder x.
array diffx[0:n]:	Bruges af proceduren til lagring af differenserne.
real x:	Den uafhængige variable. Er lig med roden ved udhoppet fra ROOT4.
real y:	Den opgivne funktion (udtryk), hvis rod skal findes.

Program d-187 viser anvendelsen af ROOT4 til bestemmelse af roden for de 11 værdier af p. Programmet er:

Program d-187. Prøve af ROOT4.

```
begin
   integer p, q, j;
   real x;
   array diffx[0:4];
   comment library ROOT4;
   real procedure y;
   begin
      y := x↑6 + p×x↑5 - 20×x - 1;
      j := j + 1
   end y;
   q := 0;
   trykvr;
   tryktekst(‹< p  Iteration      x                    y›);
```

```
    trykvr;
    for p := 20 step 1 until 30 do
    begin
        trykvr;
        tryk({-nd}, p);
        j := 0;
        ROOT4(q, 4, 0.9, 0.025, -100, 100, 1_10-5, false, diffx, x, y);
        tryk({-ndddd}, j);
        tryk({-n.ddddddd_10-dd}, trykml(4), x, trykml(4), y)
    end for p
end program;
```

Resultatudskriften er:

p	Iteration	x	y
20	6	1.0000040	$3.4344196_{10}-4$
21	5	$9.8869904_{10}-1$	$1.7404556_{10}-5$
22	4	$9.7800693_{10}-1$	$-1.9073486_{10}-4$
23	3	$9.6787753_{10}-1$	$2.8550625_{10}-4$
24	3	$9.5824559_{10}-1$	$8.6784363_{10}-5$
25	3	$9.4907493_{10}-1$	$-1.6885996_{10}-4$
26	3	$9.4033272_{10}-1$	$-1.9788742_{10}-5$
27	3	$9.3197938_{10}-1$	$3.9458275_{10}-5$
28	3	$9.2398415_{10}-1$	$-5.2630901_{10}-5$
29	3	$9.1632312_{10}-1$	$-4.1663647_{10}-5$
30	3	$9.0897147_{10}-1$	$5.5432320_{10}-6$

Programmet bestaar i det væsentlige af en for-sætning styret af p. For hver værdi af p kaldes ROOT4 een gang, og der udskrives en linie med antallet af iterationer, værdien af x og værdien af y. Vi har her sat n = 4, og vi ser, hvorledes det nødvendige antal af iterationer falder fra 6 til 3, efterhaanden som differenstabellen bygges op.

ROOT4 virker iøvrigt saaledes. Der er en lokal procedure, incr(z), der øger x med z. Hvis man derved kommer udenfor intervallet fra xmin til xmax, halveres z, indtil betingelsen er opfyldt.

Som startværdi i rodbestemmelsen bruges første gang x0, og de følgende gange beregnes startværdien ud fra differenstabellen. Den videre beregning sker derefter med det sædvanlige regula falsi udtryk:

```
delx := delx*ynew/(yold-ynew);
```

For at undgaa division med nul, benyttes halvering af den tidligere differens, hvis yold = ynew.

Naar roden er fundet, opbygges de nye differenser inden udhoppet.

Der findes en tidligere version af ROOT4, som kaldes ROOT3, men som er næsten identisk med ROOT4. Et eksempel paa brugen heraf til beregning af en tabel over ammoniakligevægten er vist paa side 18 i Kjær (1963b).

4.5. Løsning af flere ulineære Ligninger.

Vi skal nu udvide de tidligere rodbestemmelsesmetoder til ogsaa at omfatte flere variable. Som eksempel kan vi tage de tre ligninger:

$$F1(x1, x2, x3) = 0$$
$$F2(x1, x2, x3) = 0$$
$$F3(x1, x2, x3) = 0$$

Vi skal finde et talsæt x1, x2, x3, saaledes at de tre udtryk F1, F2 og F3 bliver nul paa nær en tilladelig fejl. Princippet er det samme som ved rodbestemmelser med een ubekendt. Med tre variable begynder vi med et opgivet talsæt:

```
xstart[1], xstart[2], xstart[3]
```

De tre funktioner beregnes i dette punkt samt i visse nabopunkter. Man udtrykker derefter de tre funktioner som lineære funktioner af de tre variable, sætter disse til nul og løser de derved fremkomne lineære ligninger. Den fundne løsning vil være korrekt, hvis de originale funktioner var lineære. Ellers gentages beregningen omkring det nye punkt.

Ligesom ved tilfældet med kun een ubekendt kan vi her skelne mellem

regula falsi metoden, hvor vi beregner differenskvotienter som tilnærmelser til de sande partielle differentialkvotienter, og Newton-Raphson metoden, hvor vi direkte beregner de sande differentialkvotienter. Regula falsi metoden er den mest benyttede og er vist som proceduren NOLEQ3 nedenfor. I de sjældnere tilfælde, hvor differentialkvotienterne kan beregnes analytisk, kan man bruge proceduren NOLEQ4, der arbejder med Newton-Raphson metoden.

4.5.1. Proceduren NOLEQ3. Vi giver først deklarationen af NOLEQ3:

```
procedure NOLEQ3(var, count, cmax, outside, xstart, del0x, xact, epsx,
        yact, yold, y0, inveps, maxstepfac, FOUND, ERROR);
value var, cmax, inveps, maxstepfac;
boolean outside;
integer var, count, cmax;
real inveps, maxstepfac;
array xstart, del0x, xact, epsx, yact, yold, y0;
label FOUND, ERROR;
begin
   boolean found;
   integer corcount, i, j;
   real R, S;
   array corx[1:var];
   corcount := count - (count-1)*(var+1)*(var+1);
   for i := 1 step 1 until var do xact[i] := xstart[i];
   if count ≠ 0 then
   begin
      for i := 1 step 1 until var do
      if corcount = 1 then y0[i] := - yact[i]
      else
      yold[i, corcount-1] := (yact[i] + y0[i])/del0x[corcount-1];
      if corcount < var + 1 then
      begin
         j := 1;
L1:      for i := 1 step 1 until var do xact[i] := xstart[i];
         xact[corcount] := xact[corcount] + del0x[corcount]/j;
```

```
        if outside then
        begin
           j := 2×j;
           go to L1
        end if outside
     end if corcount < var + 1
     else
     begin
        INVERT2(var, yold, inveps, ERROR);
        for i := 1 step 1 until var do
        begin
           R := 0;
           for j := 1 step 1 until var do
           R := R + yold[i, j]×y0[j];
           corx[i] := R
        end for i;
        found := true;
        S := 0;
        for i := 1 step 1 until var do
        begin
           R := abs(corx[i]/del0x[i]);
           if R > S then S := R;
           if abs(corx[i]) > epsx[i] then found := false
        end for i;
        if found then go to FOUND;
        if S > maxstepfac then
        for i := 1 step 1 until var do
        corx[i] := corx[i]/S×maxstepfac;
L2:     for i := 1 step 1 until var do
        xact[i] := xstart[i] + corx[i];
        if outside then
        begin
           for i := 1 step 1 until var do
           corx[i] := 0.5×corx[i];
           go to L2
        end if outside;
        for i := 1 step 1 until var do
```

```
            xstart[i] := xact[i]
        end if corcount = var + 1
    end count ≠ 0;
    count := count + 1;
    if count > cmax then go to ERROR
end NOLEQ-3;
```

Parametrene i NOLEQ3 er:

integer var:	Antallet af variable (ubekendte).
integer count:	Et tælleværk, som brugeren skal sætte til nul før første kald af NOLEQ3. Proceduren tæller 1 frem i count ved hvert kald.
integer cmax:	Maksimumsværdien af count. Hvis count > cmax, laver proceduren fejludhop til etiketten ERROR.
boolean outside:	Dette vil normalt være en boolean procedure. Den kaldes (ved name) af NOLEQ3, hver gang et nyt sæt af aktuelle x-værdier, xact[1:var], er beregnet. Ligger dette sæt udenfor et tilladt omraade, skal outside sættes til true, ellers til false.
array xstart[1:var]:	Startværdier af x.
array del0x[1:var]:	Tilvækster i x.
array xact[1:var]:	Det aktuelle sæt x-værdier. Det sættes altid af proceduren.
array epsx[1:var]:	Den tilladelige fejl i x-værdierne.
array yact[1:var]:	De aktuelle funktionsværdier svarende til xact. Efter hvert kald af NOLEQ3 skal hovedprogrammet beregne yact.
array yold[1:var, 1:var]:	Benyttes af proceduren til lagring af funktionsværdierne fra hvert kald.
array y0[1:var]:	Bruges af proceduren til lagring af funktionsværdierne svarende til xstart (med modsat fortegn).
real inveps:	Det mindste tilladelige diagonalelement ved matrixinversionen.

real maxstepfac:	Proceduren kontrollerer, at ændringen i variabel nr. i ikke overskrider maxstepfac *del0x[i] for alle værdier af i: $1 \leq i \leq$ var. Er dette tilfældet, reduceres alle tilvækster i samme forhold, saaledes at betingelsen netop er opfyldt.
label FOUND:	Proceduren hopper til denne etikette, naar x-værdierne er fundet med den givne tolerance, epsx. Løsningen indsættes baade i xact og i xstart.
label ERROR:	Fejludhopsetikette, som proceduren hopper til, hvis et diagonalelement bliver mindre end inveps, eller count > cmax.

Proceduren benytter den globale procedure INVERT2 til matrixinversion.

Vi illustrerer nu procedurens virkemaade for tre ligninger med tre ubekendte.

Ved første kald (count = 0) indsættes blot startværdierne af xact:

```
for i := 1 step 1 until var do xact[i] := xstart[i];
```

Hovedprogrammet skal derefter beregne de tilsvarende yact-værdier.

I næste kald (count = 1) gemmes yact-værdierne i y0 (med modsat fortegn):

```
for i := 1 step 1 until var do y0[i] := - yact[i];
```

Derefter sætter proceduren de nye værdier af xact:

```
xact[1] := xstart[1] + del0x[1];
xact[2] := xstart[2];
xact[3] := xstart[3];
```

Hovedprogrammet skal derefter beregne de tilsvarende yact-værdier.

I næste kald (count = 2) gemmer proceduren yact-værdierne i den første søjle af yold-matricen:

```
for i := 1 step 1 until var do
yold[i, 1] := (yact[i] + y0[i])/del0x[1];
```

Dette er differenskvotienterne af de tre funktioner med hensyn til den første variable.

Nu sættes det nye sæt x-værdier:

```
xact[1] := xstart[1];
xact[2] := xstart[2] + del0x[2];
xact[3] := xstart[3];
```

og efter udhop og nyt kald (count = 3) gemmes de nye differenskvotienter i den anden søjle af yold.

Endelig faar vi for count = 4 gemt differenskvotienterne for x-værdierne:

```
xact[1] := xstart[1];
xact[2] := xstart[2];
xact[3] := xstart[3] + del0x[3];
```

i den tredje søjle af yold. Elementet yold[i, j] er altsaa differenskvotienten af funktion nr. i med hensyn til den variable nr. j. Først nu begynder de egentlige beregninger. For kortheds skyld skriver vi differentialkvotienterne som:

```
a[i, j] := yold[i, j];
```

og vi udtrykker den søgte løsning ved startværdien og en korrektion:

```
x[1] := xstart[1] + corx[1];
x[2] := xstart[2] + corx[2];
x[3] := xstart[3] + corx[3];
```

De lineære tilnærmelser til de tre funktioner kan nu skrives som:

```
F[1] := a[1,1]*x[1] + a[1,2]*x[2] + a[1,3]*x[3] + k[1];
F[2] := a[2,1]*x[1] + a[2,2]*x[2] + a[2,3]*x[3] + k[2];
F[3] := a[3,1]*x[1] + a[3,2]*x[2] + a[3,3]*x[3] + k[3];
```

Paa matrixform kan vi skrive det:

$$F = ax + k$$

Sætter vi heri vektoren x = xstart + corx, faar vi:

$$F = a(xstart+corx) + k$$

For x = xstart er F = - y0:

$$-y0 = axstart + k$$

og for x = xstart + corx er F = 0:

$$0 = a(xstart+corx) + k$$

Dette giver:

$$y0 = a\ corx$$

eller skrevet helt ud:

```
a[1,1]*corx[1] + a[1,2]*corx[2] + a[1,3]*corx[3] = y0[1]
a[2,1]*corx[1] + a[2,2]*corx[2] + a[2,3]*corx[3] = y0[2]
a[3,1]*corx[1] + a[3,2]*corx[2] + a[3,3]*corx[3] = y0[3]
```

Dette er et simpelt, lineært ligningssystem (se side 12). Vi inverterer a-matricen (yold) og multiplicerer den inverterede matrix med højresiden, y0. Herved findes talsættet corx[1:var].

Inden corx-vektoren adderes til xstart-vektoren, undersøger proceduren, om alle corx-elementerne er mindre end eller lig med det tilsvarende epsx-element (numerisk). Er dette tilfældet, hoppes straks til etiketten: FOUND. Er dette ikke tilfældet, findes størsteværdien af:

```
abs(corx[i]/del0x[i])
```

for alle i: 1 ≤ i ≤ var. Hvis størsteværdien, S, er større end maxstepfac, bliver alle corx-værdierne reduceret:

```
for i := 1 step 1 until var do
corx[i] := corx[i]/S×maxstepfac;
```

Derefter indsættes de nye værdier af xact:

```
for i := 1 step 1 until var do
xact[i] := xstart[i] + corx[i];
```

Proceduren kalder derefter parametren outside, og hvis denne er sand, halveres alle corx-elementer, og xact beregnes igen, outside kaldes igen, o.s.v. Saa snart outside bliver falsk, sættes ogsaa xstart lig med xact:

```
for i := 1 step 1 until var do xstart[i] := xact[i];
```

Proceduren forlades derefter, og hovedprogrammet maa da beregne de nye yact-værdier, der (med modsat fortegn) gemmes som y0, og vi begynder forfra med en ny cyklus, hvor hver variabel faar en tilvækst til bestemmelse af differenskvotienten.

4.5.2. Proceduren POL. Til demonstration af brugen af NOLEQ3 og af de senere optimeringsprocedurer kan vi med fordel bruge en lille procedure, POL, der simulerer en ammoniakkonverterberegning.

Paa side 43-47 i Kjær (1963b) er omtalt beregningen af ammoniakkonvertere af quench-typen, d.v.s. hvor katalysatoren køles ved tilsætning af kold gas paa forskellige punkter ned gennem katalysatorlaget. For en bestemt konverter af denne type (altsaa med fikserede dimensioner) og for fastholdte driftsbetingelser, vil resultatet af konverterberegningen være en funktion af de to variable:

tinlet: Indgangstemperaturen til det øverste lag katalysator (gr.C).

ginlet: Den relative gasmængde, som strømmer ind i det øverste lag katalysator (procent).

Resultatet af beregningen udtrykkes som de to funktioner:

PROD: Ammoniakproduktionen (metr.t/24 h).

LNEC: Den nødvendige højde af den nedre varmeveksler (meter).

For en bestemt konverter af denne type og for bestemte driftsbetingelser blev udført 48 beregninger af PROD og LNEC, nemlig for 6 værdier af tinlet og 8 værdier af ginlet. Resultatet heraf er vist i tabellerne nedenfor:

Ammoniakproduktion, PROD

tinlet:	400	410	420	430	440	450
ginlet:						
66	67.49	68.80	70.00	71.12	71.88	72.35
68	68.69	70.03	71.27	72.06	72.55	72.78
70	69.88	71.25	72.10	72.64	72.85	72.72
72	71.03	72.02	72.64	72.85	72.72	72.32
74	71.81	72.53	72.81	72.68	72.28	71.64
76	72.32	72.70	72.63	72.25	71.59	70.69
78	72.54	72.55	72.20	71.56	70.64	69.54
80	72.42	72.15	71.53	70.62	69.50	68.23

Nødvendig højde af varmeveksler, LNEC

tinlet:	400	410	420	430	440	450
ginlet:						
66	1.92	2.13	2.41	2.81	2.96	3.17
68	1.92	2.12	2.38	2.48	2.61	2.79
70	1.93	2.12	2.18	2.28	2.41	2.58
72	1.93	1.98	2.05	2.15	2.29	2.46
74	1.83	1.88	1.97	2.08	2.22	2.39
76	1.75	1.82	1.91	2.03	2.17	2.35
78	1.70	1.78	1.88	2.00	2.15	2.33
80	1.67	1.76	1.86	1.99	2.14	2.32

De to funktioner blev derefter behandlet paa polynomieapproksimationsprogrammet PA-7 (eller rettere paa dets forløber i maskinsprog, PA-4). Den tilsvarende beregning er vist som typisk beregning for PA-7 i appendikset. De to sæt polynomiekoefficienter blev udtrykt som en real procedure POL med de tre parametre:

integer type: For type = 1 faas PROD og for type = 2 faas LNEC.
real x1: Den aktuelle værdi af tinlet.
real x2: Den aktuelle værdi af ginlet.

Deklarationen af POL er:

```
real procedure POL(type, x1, x2);
value type, x1, x2;
integer type;
real x1, x2;
begin
   integer j;
   real RES;
   procedure P(a0, a1, a2, a3, a4);
   value a0, a1, a2, a3, a4;
   real a0, a1, a2, a3, a4;
   begin
      RES := RES + ((((a4×x1 + a3)×x1 + a2)×x1 + a1)×x1 + a0)×x2↑j;
```

```
   j := j + 1
end P;
RES := 0;
j := 0;
x1 := x1 - 425;
x2 := x2 - 73;
if type = 1 then
begin
   P( 7.28366518E 1,  4.94289484E-3, -1.90311550E-3,  9.74322842E-6,
      2.37557410E-7);
   P( 4.24103338E-3, -1.56682775E-2, -5.43938744E-6,  9.82280472E-7,
      8.36939706E-8);
   P(-4.66653700E-2,  1.14471391E-4,  9.23091754E-5,  1.43909563E-7,
     -1.31535297E-7);
   P( 6.12106929E-4,  3.12216484E-5,  5.34078535E-6, -1.22135790E-8,
     -1.04237200E-8);
   P( 3.48684193E-4,  5.81336386E-6, -5.41804759E-6, -2.65185593E-8,
      8.68393465E-9);
   P( 1.14714568E-6, -7.55820787E-9, -1.10374974E-7,  6.64947167E-10,
      1.75072212E-10);
   P(-4.59104682E-6, -1.39502023E-7,  7.82265943E-8,  3.78765070E-10,
     -1.24104660E-10)
end
else
begin
   P( 2.05517761   ,  1.09969770E-2,  1.27238970E-4, -4.95970713E-7,
      4.80753033E-8);
   P(-3.57374946E-2,  9.07821752E-4, -8.34082894E-5, -1.23463257E-6,
      1.28303912E-7);
   P( 3.43945688E-3, -2.85644784E-4,  2.50597903E-5,  8.09620557E-7,
     -5.17148586E-8);
   P(-8.95571458E-4, -4.42799781E-5,  8.70536738E-6,  2.15024151E-8,
     -1.07247342E-8);
   P( 1.35239950E-4,  2.00152652E-5, -2.33346434E-6, -3.62393101E-8,
      3.69149646E-9);
   P( 1.23541288E-5, -3.86864207E-8, -1.25283778E-7,  4.15479619E-10,
      1.48571028E-10);
```

```
      P(-2.35409855(10) -6, -1.95448903(10) -7,  3.46996145(10) -8,  3.63699191(10)-10,
        -5.19929076(10)-11);
   end;
   POL := RES
end POL;
```

De to polynomier beregnes næsten paa samme maade som eksemplet vist paa side 48.

Til demonstration af NOLEQ3 vil vi bestemme det sæt værdier af tinlet og ginlet for hvilket:

```
PROD = 71.5
LNEC =  2
```

Det sker ved at beregne yact-værdierne som:

```
yact[1] := POL(1, xact[1], xact[2]) - 71.5;
yact[2] := POL(2, xact[1], xact[2]) -  2;
```

idet xact[1] er tinlet og xact[2] er ginlet. Programmet er:

Program d-188. Prøve af NOLEQ3.

```
begin
   boolean fin;
   integer count;
   real PROD, LNEC;
   array xstart, del0x, xact, epsx, yact, y0[1:2], yold[1:2, 1:2];
   comment library INVERT2;
   comment library POL;
   comment library NOLEQ3;
   xstart[1] := 440;
   xstart[2] := 68;
   del0x[1] := 5;
   del0x[2] := 2;
   epsx[1] := 1;
   epsx[2] := 0.2;
   count := 0;
```

```
   trykteskst(⟨<
  t-inlet   g-inlet      PROD         LNEC
⟩);
   fin := false;
H1: NOLEQ3(2, count, 50, false, xstart, del0x, xact, epsx, yact, yold, y0,
          1₁₀-8, 4, F1, E1);
   go to G2;
F1: trykvr;
   trykteskst(⟨<FOUND⟩);
   go to G1;
E1: trykvr;
   trykteskst(⟨<ERROR⟩);
G1: fin := true;
G2: trykvr;
   if (count-1):3×3 = count - 1 then trykvr;
   tryk(⟨-nddd.dddd⟩, xact[1], xact[2]);
   PROD := POL(1, xact[1], xact[2]);
   yact[1] := PROD - 71.5;
   LNEC := POL(2, xact[1], xact[2]);
   yact[2] := LNEC - 2;
   tryk(⟨-nddd.dddd00⟩, PROD, LNEC);
   if ¬ fin then go to H1;
   trykvr
end program;
```

Resultatudskriften er:

```
  t-inlet   g-inlet      PROD         LNEC


 440.0000   68.0000   72.538228    2.599180
 445.0000   68.0000   72.669138    2.669936
 440.0000   70.0000   72.853761    2.411905

 420.0000   68.1038   71.298672    2.348836
 425.0000   68.1038   71.759266    2.428249
 420.0000   70.1038   72.145484    2.178333
```

```
  411.0457    70.5275    71.587531    2.092726
  416.0457    70.5275    72.014447    2.123248
  411.0457    72.5275    72.266506    1.958033

  406.2680    71.4713    71.500424    1.998335
  411.2680    71.4713    71.969902    2.030972
  406.2680    73.4713    72.175319    1.877078
FOUND
  406.2680    71.4713    71.500424    1.998335
```

Løsningen findes efter 13 iterationer til tinlet = 406 og ginlet = 71.5. Det bemærkes, at der meget vel kan være flere løsninger. Hvilken løsning proceduren finder, afhænger af startværdien. Dette forhold gælder naturligvis ogsaa, hvis der er flere rødder i en funktion af een variabel.

4.5.3. Proceduren NOLEQ4. Her bruger vi Newton-Raphsons metode i stedet for regula falsi metoden, som i NOLEQ3. Deklarationen er:

```
procedure NOLEQ4(var, count, cmax, outside, xstart, del0x, xact, epsx,
          yact, yold, y0, inveps, maxstepfac, r, s, FUNC, DERIV, FOUND,
          ERROR);
value var, cmax, inveps, maxstepfac;
boolean outside;
integer var, count, cmax, r, s;
real inveps, maxstepfac, FUNC, DERIV;
array xstart, del0x, xact, epsx, yact, yold, y0;
label FOUND, ERROR;
begin
   boolean found;
   integer i, j;
   real R, S;
   array corx[1:var];
   if count = 0 then
   begin
      for i := 1 step 1 until var do xact[i] := xstart[i];
      for r := 1 step 1 until var do
```

```
      y0[r] := - FUNC
   end
   else
   begin
      for r := 1 step 1 until var do
      for s := 1 step 1 until var do
      yold[r, s] := DERIV;
      INVERT2(var, yold, inveps, ERROR);
      for i := 1 step 1 until var do
      begin
         R := 0;
         for j := 1 step 1 until var do
         R := R + yold[i, j]×y0[j];
         corx[i] := R
      end for i;
      found := true;
      S := 0;
      for i := 1 step 1 until var do
      begin
         R := abs(corx[i]/del0x[i]);
         if R > S then S := R;
         if abs(corx[i]) > epsx[i] then found := false
      end for i;
      if found then go to FOUND;
      if S > maxstepfac then
      for i := 1 step 1 until var do
      corx[i] := corx[i]/S×maxstepfac;
L2:   for i := 1 step 1 until var do
      xact[i] := xstart[i] + corx[i];
      if outside then
      begin
L3:      for i := 1 step 1 until var do
         corx[i] := 0.5×corx[i];
         go to L2
      end if outside;
      R := S := 0;
      for r := 1 step 1 until var do
```

```
      begin
         yact[r] := - FUNC;
         R := R + abs(y0[r]);
         S := S + abs(yact[r])
      end for r;
      if S > R then go to L3;
      for i := 1 step 1 until var do
      begin
         y0[i] := yact[i];
         xstart[i] := xact[i]
      end for i
   end count ≠ 0;
   count := count + 1;
   if count > cmax then go to ERROR
end NOLEQ-4;
```

Sammenlignet med NOLEQ3 er parametrene de samme, men fire nye er kommet til:

`integer r, s:` Disse variable sættes af NOLEQ4, naar de to parametre FUNC og DERIV kaldes.

`real FUNC:` Dette vil normalt være en real procedure. Den kaldes af NOLEQ4 i for-sætningen:

```
for r := 1 step 1 until var do y0[r] := - FUNC;
```

FUNC skal derfor give funktionsværdien af funktion nr. r og for de aktuelle værdier, xact, af x.

`real DERIV:` Dette vil ogsaa normalt være en real procedure. Den kaldes af NOLEQ4 i den dobbelte for-sætning:

```
for r := 1 step 1 until var do
for s := 1 step 1 until var do
yold[r, s] := DERIV;
```

DERIV skal give den partielle differentialkvotient af funktion nr. r med hensyn til den variable nr. s. De sande differentialkvotienter indgaar

altsaa i stedet for differenskvotienterne i NOLEQ3.

Hvert kald af NOLEQ4 giver en komplet cyklus, d.v.s. beregning af var funktionsværdier og var↑2 differentialkvotienter. Tælleren, count, vokser med 1 for hvert kald paa een cyklus, altsaa var + 1 gange langsommere end count i NOLEQ3.

En sammenligning af NOLEQ3 og NOLEQ4 er vist i følgende program:

Program d-189. Prøve af NOLEQ3 og NOLEQ4.

```
begin
   boolean fin;
   integer count, type, r, s;
   array xstart, del0x, xact, epsx, yact, y0[1:2], yold[1:2, 1:2];
   comment library INVERT2;
   comment library NOLEQ3;
   comment library NOLEQ4;
   real procedure FUNC;
   FUNC := if r = 1 then
   xact[1]↑2 + xact[2]↑2 - 2
   else
   1/xact[1]↑2 + xact[2]↑2 - 2;
   real procedure DERIV;
   DERIV := if r = 1 ∨ r = 2 ∧ s = 2 then 2×xact[s] else -2/xact[1]↑3;
   for type := 3, 4 do
   begin
      xstart[1] := 2;
      xstart[2] := 3;
      del0x[1] := del0x[2] := 0.5;
      epsx[1] := epsx[2] := 1_10-4;
      count := 0;
      trykvr;
      tryk(‹-n›, type);
      tryktekst(‹<   x1              x2              y1              y2›);
      trykvr;
      fin := false;
H1:   if type = 3 then
      NOLEQ3(2, count, 50, xact[1] ≤ 0 ∨ xact[2] ≤ 0, xstart, del0x, xact,
             epsx, yact, yold, y0, 1_10-8, 2, F1, E1)
```

```
      else
      NOLEQ4(2, count, 50, xact[1] ≤ 0 ∨ xact[2] ≤ 0, xstart, del0x, xact,
            epsx, yact, yold, y0, 1₁₀-8, 2, r, s, FUNC, DERIV, F1, E1);
      go to G2;
F1:   trykvr;
      tryktekst({<FOUND});
      go to G1;
E1:   trykvr;
      tryktekst({<ERROR});
G1:   fin := true;
G2:   for r := 1, 2 do yact[r] := FUNC;
      if (count-1):3×3 = count - 1 ∨ fin ∨ type = 4 then
      tryk({-n.dddddd₁₀-dd}, trykvr, xact[1], trykml(2), xact[2], trykml(2),
         yact[1], trykml(2), yact[2]);
      if -, fin then go to H1
   end for type
end program;
```

Resultatudskriften er:

```
 3   x1                x2                y1                y2

 2.000000          3.000000          1.100000 10 1     7.250000
 1.295622          2.000000          3.678636          2.595722
 9.999367 10-1     1.385644          9.198833 10-1     9.201366 10-1
 1.000007          1.104392          2.196943 10-1     2.196671 10-1
 9.999993 10-1     1.023294          4.712931 10-2     4.713225 10-2
 1.000000          1.004786          9.595938 10-3     9.595625 10-3
 1.000000          1.000963          1.926482 10-3     1.926526 10-3
 1.000000          1.000193          3.855824 10-4     3.855824 10-4
 1.000000          1.000039          7.712841 10-5     7.712841 10-5
FOUND
 1.000000          1.000039          7.712841 10-5     7.712841 10-5
```

```
4   x1              x2              y1              y2

 2.000000        3.000000        1.100000 10 1   7.250000
 1.291339        2.000000        3.667555        2.599680
 9.872263 10-1   1.279467        6.116519 10-1   6.630814 10-1
 9.999153 10-1   1.030650        6.207076 10-2   6.240970 10-2
 1.000000        1.000456        9.117275 10-4   9.117350 10-4
 1.000000        1.000000        2.086163 10-7   2.086163 10-7
FOUND
 1.000000        1.000000        2.086163 10-7   2.086163 10-7
```

Programmet løser de to ligninger:

$$x1\uparrow 2 + x2\uparrow 2 = 2$$
$$1/x1\uparrow 2 + x2\uparrow 2 = 2$$

som har løsningen x1 = x2 = 1 for positive værdier af x1 og x2. Resultatet udskrives for hver cyklus, først for NOLEQ3 med 9 cykler, og derefter for NOLEQ4 med 6 cykler.

5. OPTIMERING AF FUNKTIONER

Vi har nu set, hvorledes man undersøger en forelagt funktion for forekomsten af nulpunkter. En anden vigtig praktisk opgave er at finde maksimumsværdien af en funktion, d.v.s. et punkt, hvor funktionsværdien er større end eller lig med funktionsværdien i alle andre punkter i det anvendte definitionsomraade.

Vi tager som eksempel funktionen:

$$F(x1, x2, x3)$$

som er en funktion af de tre variable x1, x2 og x3. Hvis F er et simpelt udtryk i de tre variable, kan vi maaske beregne formlerne for de tre partielle differentialkvotienter med hensyn til de tre variable:

$$\frac{dF}{dx1} \quad , \quad \frac{dF}{dx2} \quad \text{og} \quad \frac{dF}{dx3}$$

Under forudsætning af at maksimet ikke ligger paa randen af definitionsomraadet, skal de tre differentialkvotienter være nul i maksimet. Problemet er derfor reduceret til at løse de tre (normalt ulinesre) ligninger:

$$dFdx1(x1, x2, x3) = 0$$
$$dFdx2(x1, x2, x3) = 0$$
$$dFdx3(x1, x2, x3) = 0$$

Dette kan gøres med NOLEQ3 eller NOLEQ4.

Hvis vi ikke søger funktionens maksimum, men dens minimum, kan vi blot erstatte F med -F og søge maksimum herfor.

Ligesom ved rodbestemmelser vil vi anvende den strategi at begynde med et sæt startværdier, xstart[1:var], undersøge funktionen i dette punkt og nogle nabopunkter og iterativt nærme os den rigtige løsning. Man risikerer da at havne i et lokalt maksimum, d.v.s. et punkt hvor de partielle differentialkvotienter er nul, men hvor funktionsværdien ikke er den største i det foreliggende definitionsomraade. Hvis funktionen har flere

lokale maksima indenfor omraadet, vil vore metoder føre os hen mod et af disse punkter, men ikke nødvendigvis det med den absolut største funktionsværdi. Er der risiko for, at noget saadant kan ske, kan man f.eks. foretage en orienterende undersøgelse af funktionen i enkelte punkter spredt over hele omraadet, inden iterationen startes.

Vi diskuterer tre forskellige optimeringsmetoder. De adskiller sig ved den maade paa hvilken funktionen udforskes i nabopunkterne.

I den første metode (OPT1A) undersøges funktionen kun i diskrete punkter med en bestemt afstand. Man kan her let tage sidebetingelser med ind i overvejelserne og ogsaa klare det tilfælde, at maksimet ligger i randen af definitionsomraadet.

I den anden metode (OPT3) undersøges funktionen i lige saa mange nabopunkter, som der er uafhængige variable. Funktionen tilnærmes derefter med et lineært udtryk, og vi bevæger os i den retning, hvor stigningen er stærkest.

I den tredje metode (OPT4) undersøges funktionen i var×(var+2)/2 nabopunkter, og der beregnes et andengradspolynomium, som gaar eksakt igennem disse punkter. Proceduren finder de partielle differentialkvotienter ud fra polynomiet og finder maksimet ved løsning af de tilsvarende lineære ligninger, som vist ovenfor.

Endelig omtales proceduren OPT5, der virker som OPT4, men med tromlelagring af matricen, samt proceduren OPT7, der ogsaa er en udvidelse af OPT4. Her kan man tilfredsstille sidebetingelser, baade for funktionen selv og for andre funktioner.

5.1. Diskrete Metoder.

5.1.1. Proceduren OPT1A. Denne procedure finder først og fremmest maksimum af en funktion af een variabel:

```
y1 := f1(x);
```

idet den varierer x med en fast skridtlængde, delx:

x
x + delx
x + 2×delx
x + 3×delx
o.s.v.

Denne øgning af x fortsættes, saalænge y1-værdierne ogsaa vokser. Saa snart de begynder at falde igen, vælges den næstsidste x-værdi som den optimale. Dette lyder næsten for simpelt til at det kan betale sig at have en speciel procedure hertil, og OPT1A indeholder da ogsaa flere ekstra finesser.

Hvis maksimumværdien ligger lavere end startværdien af x, faar man følgende rækkefølge:

x
x + delx
x - delx
x - 2×delx
x - 3×delx
o.s.v.

Der opgives et tilladeligt interval fra xmin til xmax. Proceduren vil ikke lade x vokse eller aftage ud over intervallet, men vælger den bedste løsning.

Endelig vil proceduren ogsaa tage hensyn til en anden funktion af den samme variable:

```
y2 := f2(x);
```

Det forlanges, at y2 skal være positiv eller nul.

Proceduren vil altsaa lade x variere opad eller nedad, indtil den finder den x-værdi, som giver den største værdi af y1 og for hvilken samtidigt $y2 \geq 0$. De værdier af x, for hvilke y2 er negativ, lades ude af betragtning.

Det forudsættes, at de to funktioner, f1(x) og f2(x), højst har eet maximum og intet minimum i det indre af det opgivne interval.

Proceduren OPT1A har især været brugt (og bruges stadig) til ammoniak-

konverterberegninger, hvor man ønsker at variere katalysatortemperaturen indtil ammoniakproduktionen bliver maksimum, og saaledes at overskuddet af den nedre varmeveksler ikke er negativt.

Deklarationen af OPT1A er:

```
procedure OPT1A(q, x, y1, y2, num, xmin, delx, xmax, xold, y1old, y2old,
        numold, xopt, y1opt, y2opt, numopt, FOUND, ERROR);
value y1, y2, num, xmin, xmax;
integer q, num, numopt;
real x, y1, y2, xmin, delx, xmax, xopt, y1opt, y2opt;
array xold, y1old, y2old;
integer array numold;
label FOUND, ERROR;
begin
   integer row, col, r;
   switch TABLE2 := ER, UP, DN, S1, S2;
   switch TABLE3 := ER, S1, S2, S3, D1, U3, DE, UE;
   integer procedure NEG(z);
   value z;
   real z;
   NEG := if z < 0 then 1 else 0;
   procedure TA(n);
   value n;
   integer n;
   if r = row then
   begin
      n := n:(8/col);
      n := n - n:8×8 + 1;
      go to if q = 1 then TABLE2[n] else TABLE3[n]
   end
   else r := r + 1;
   procedure MOVE(n) to:(m);
   value n, m;
   integer n, m;
   begin
      boolean zero;
      zero := n = 0;
```

```
        xold[m] := if zero then x else xold[n];
        y1old[m] := if zero then y1 else y1old[n];
        y2old[m] := if zero then y2 else y2old[n];
        numold[m] := if zero then num else numold[n]
    end MOVE;
    procedure SELECT(n);
    value n;
    integer n;
    begin
        xopt := xold[n];
        y1opt := y1old[n];
        y2opt := y2old[n];
        numopt := numold[n];
        go to FOUND
    end SELECT;
    if q = 0 then
    begin
        MOVE(0, 1);
        delx := abs(delx);
UP:     x := x + delx
    end
    else
    if q = 1 then
    begin
        MOVE(0, 2);
        row := NEG(y2old[1] - y2old[2]) + 2×NEG(y1old[1] - y1old[2]);
        col := NEG(y2old[2]) + 2×NEG(y2old[1]);
        r := 0;
        TA(2 + 2×8 +       2×512);
        TA(2 +     4×64 + 1×512);
        TA(1 + 3×8 +       2×512);
        TA(1 +     1×64 + 1×512);
ER:     go to ERROR;
DN:     MOVE(2, 3);
        MOVE(1, 2);
        r := if q = 1 then 2 else 3×sign(delx);
        delx := - abs(delx);
```

```
      if r < 0 then go to UP;
      x := x - r*abs(delx)
   end if q = 1
   else
   begin
      MOVE(0, 2 + sign(delx));
      row := NEG(y2old[2] - y2old[3]) + 2*NEG(y2old[1] - y2old[2])
          + 4*NEG(y1old[2] - y1old[3]) + 8*NEG(y1old[1] - y1old[2]);
      col := NEG(y2old[3]) + 2*NEG(y2old[2]) + 4*NEG(y2old[1]);
      r := 0;
      TA(4 + 4*8 + 4*512 +                              6*2097152);
      r := 2;
      TA(4 + 4*8      +2*4096 + 2*32768                          );
      TA(4 +           2*4096 +         3*262144 + 7*2097152);
      r := 8;
      TA(2 + 2*8 + 1*512                          + 6*2097152);
      r := 10;
      TA(2 + 2*8 +     2*4096 + 2*32768                          );
      TA(2 +           2*4096 +         3*262144 + 7*2097152);
      TA(5 + 2*8 + 1*512                          + 6*2097152);
      r := 14;
      TA(5 + 2*8 +     5*4096 + 2*32768                          );
      TA(5 +           5*4096 +         5*262144 + 7*2097152);
      go to ERROR;
S1:   SELECT(1);
S2:   SELECT(2);
S3:   SELECT(3);
D1:   go to if xold[1] - abs(delx) ≥ xmin then DN
      else S1;
DE:   go to if xold[1] - abs(delx) ≥ xmin then DN
      else ERROR;
U3:   if delx < 0 then go to ERROR;
      go to if xold[3] + delx ≤ xmax then L1
      else S3;
UE:   if delx < 0 then go to ERROR;
      if xold[3] + delx > xmax then go to ERROR;
```

```
L1:   MOVE(2, 1);
      MOVE(3, 2);
      go to UP
   end q > 1;
   q := q + 1
end OPT1A;
```

De formelle parametre er:

integer q: Et tælleværk, som brugeren maa sætte til nul før første kald. Proceduren tæller 1 frem her efter hvert kald.

real x: Den aktuelle værdi af den uafhængige variable. Startværdien maa være indsat her før første kald. Proceduren indsætter selv nye værdier her i hvert kald.

real y1, y2: De to funktionsværdier svarende til x. Før første kald maa hovedprogrammet have indsat de to værdier, som svarer til startværdien af x. Efter hvert kald maa hovedprogrammet ligeledes beregne y-værdierne svarende til det nye x.

integer num: Et identifikationsnummer svarende til det aktuelle talsæt: x, y1 og y2. Maa leveres af hovedprogrammet.

real xmin: Nedre grænse for x.

real delx: Den faste tilvækst i x. Da proceduren skifter fortegn paa delx, hvis x skal formindskes, skal den aktuelle parameter være en variabel, ikke en talværdi.

real xmax: Øvre grænse for x.

array xold, y1old, y2old[1:3]: Bruges af proceduren til lagring af gamle resultater. Kun data fra de tre sidste kald gemmes.

integer array numold[1:3]: Bruges ogsaa af proceduren til lagring af gamle identifikationsnumre.

real xopt: Her afleverer proceduren den optimale x-værdi.

real y1opt, y2opt: De tilsvarende værdier af y1 og y2 som proceduren ogsaa afleverer.

integer numopt: Identifikationsnummeret svarende til xopt. Afleveres af proceduren.

label FOUND: Hertil hoppes, naar xopt er fundet.

label ERROR: Hertil hoppes, hvis xopt ikke kan findes. Dette sker, hvis y2 er negativ for alle x, eller hvis talmaterialet ikke opfylder be-

tingelsen, at der højst er eet maksimum og intet minimum paa de to funktioner.

Proceduren virker saaledes:

I første kald (q=0) gemmes talsættet x, y1 og y2. Fortegnet af delx sættes til plus, og x øges:

```
x := x + delx;
```

Hovedprogrammet skal da beregne de nye værdier af y1 og y2.

I næste kald (q=1) gemmes det nye talsæt ogsaa, og proceduren skal nu paa basis af disse seks tal:

```
xold[1], y1old[1], y2old[1]
xold[2], y1old[2], y2old[2]
```

træffe et valg mellem følgende fem muligheder:

UP: x skal øges yderligere
DN: x skal nu formindskes
S1: Vælg første sæt som optimalt
S2: Vælg andet sæt som optimalt
ER: Forudsætningerne er ikke opfyldt, og der skal hoppes til ERROR

Proceduren benytter følgende beslutningstabel:

Beslutningstabel for 2 datasæt.

y1old[1] og y1old[2]	y2old[1] og y2old[2]	row	Fortegn af y2old			
			1 2	1 2	1 2	1 2
			+ +	+ -	- +	- -
			col=0	col=1	col=2	col=3
1 2	1 2	0	DN	DN	ER	DN
1 2	2 1	1	DN	ER	S2	UP
2 1	1 2	2	UP	S1	ER	DN
2 1	2 1	3	UP	ER	UP	UP

De fire rækker i tabellen svarer til de fire kombinationer af voksende eller faldende værdi af y1 og y2. De fire søjler i tabellen svarer til de fire fortegnskombinationer af y2old[1] og y2old[2]. Proceduren beregner paa simpel vis rækkenummeret, row, og søjlenummeret, col, ud fra fortegnene af de fire størrelser:

```
y1old[1] - y1old[2]
y2old[1] - y2old[2]
y2old[1]
y2old[2]
```

Nu kunne man have anbragt en switch med 16 etiketter, men da der i realiteten kun er fem forskellige etiketter, kan det betale sig at skrive det lidt anderledes. Dette er især vigtigt, naar vi kommer til den store beslutningstabel, baseret paa 3 sæt data og indeholdende 128 muligheder, men heraf kun 8 forskellige.

Vi giver hver etikette et nummer:

```
ER: 0
UP: 1
DN: 2
S1: 3
S2: 4
```

Etikettetabellen:

```
DN   DN   ER   DN
DN   ER   S2   UP
UP   S1   ER   DN
UP   ER   UP   UP
```

skrives derefter som:

```
2    2    0    2
2    0    4    1
1    3    0    2
1    0    1    1
```

Ved at multiplicere tallene i søjle nr. s (s = 0, 1, 2, 3) med 8↑s, faar vi da

```
2    2×8    0×64    2×512
2    0×8    4×64    1×512
1    3×8    0×64    2×512
1    0×8    1×64    1×512
```

Vi kan nu addere tallene i hver række, saaledes at vi faar:

```
1042
 770
1049
 577
```

I stedet for de oprindelige 16 tal eller etiketter har vi nu kun 4 tal, eet for hver række. Proceduren indeholder fire kald af en lokal procedure, TA(n), nemlig:

```
TA(1042);
TA( 770);
TA(1049);
TA( 577);
```

Først beregnes rækkenummeret, row, og søjlenummeret, col, ud fra fortegnene. Derefter sættes tælleren, r, til nul, og vi begynder de fire TA-kald. I hvert kald undersøges det om r = row, og hvis dette er tilfældet, beregnes nummeret paa switch-elementet som:

```
n := n:(8↑col);
n := n - n:8×8 + 1;
```

og der hoppes til TABLE2[n] eller TABLE3[n], alt eftersom det er beslutningstabellen for to datasæt eller tre datasæt, vi arbejder med. Er r ≠ row, øges r med 1 og vi gaar til næste TA-kald.

Bemærk, hvorledes vi ikke skriver TA(1042) men:

```
TA(2 + 2×8 + 0×64 + 2×512);
```

og derved overlader det til oversætterprogrammet at foretage udregningen.

Hvis vi naar videre til tredje kald (q = 2), bliver beslutningstabellen som vist paa næste side.

Her skal valget træffes mellem 8 forskellige muligheder:

S1:	Vælg første sæt som optimalt
S2:	- andet - - -
S3:	- tredje - - -
D1:	Formindsk x eller vælg første sæt
U3:	Øg x eller vælg tredje sæt
DE:	Formindsk x eller hop til ERROR
UE:	Øg x eller hop til ERROR
ER:	Hop til ERROR

I de to tilfælde D1 og U3 skal valget af sæt 1 eller 3 bruges, hvis en yderligere ændring af x bringer os udenfor intervallet fra xmin til xmax. Det samme gælder fejlhopsmuligheden i de to tilfælde DE og UE.

Valget af de 8 muligheder sker paa basis af fortegnet af de 7 størrelser:

```
y1old[1] - y1old[2]
y1old[2] - y1old[3]
y2old[1] - y2old[2]
y2old[2] - y2old[3]
y2old[1]
y2old[2]
y2old[3]
```

Heraf beregnes rækkenummer og søjlenummer i tabellen. Denne er iøvrigt kondenseret til eet tal per række, ligesom for den lille tabel.

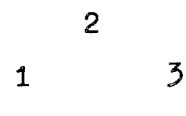
For at lette oversigten skriver vi kombinationen:

```
y1old1[1] < y1old1[2]
y1old1[2] > y1old1[3]
```

som:

Beslutningstabel for 3 datasæt.

y1old[1] y1old[2] y1old[3]	y2old[1] y2old[2] y2old[3]	row	Fortegn af y2old 123 +++ col=0	123 ++- col=1	123 +-+ col=2	123 +-- col=3	123 -++ col=4	123 -+- col=5	123 --+ col=6	123 --- col=7
	1 2 3	0	D1	D1	ER	D1	ER	ER	ER	DE
1	1 3 2	1	ER	ER	ER	ER	ER	ER	ER	ER
2										
3	2 1 3	2	D1	D1	ER	ER	S2	S2	ER	ER
	3 2 1	3	S2	ER	ER	ER	S2	ER	S3	UE
	1 2 3	4	ER	ER	ER	ER	ER	ER	ER	ER
1 3	1 3 2	5	ER	ER	ER	ER	ER	ER	ER	ER
2										
	2 1 3	6	ER	ER	ER	ER	ER	ER	ER	ER
	3 2 1	7	ER	ER	ER	ER	ER	ER	ER	ER
	1 2 3	8	S2	S2	ER	S1	ER	ER	ER	DE
2	1 3 2	9	ER	ER	ER	ER	ER	ER	ER	ER
1 3										
	2 1 3	10	S2	S2	ER	ER	S2	S2	ER	ER
	3 2 1	11	S2	ER	ER	ER	S2	ER	S3	UE
	1 2 3	12	U3	S2	ER	S1	ER	ER	ER	DE
3	1 3 2	13	ER	ER	ER	ER	ER	ER	ER	ER
2										
	2 1 3	14	U3	S2	ER	ER	U3	S2	ER	ER
1	3 2 1	15	U3	ER	ER	ER	U3	ER	U3	UE



Bemærk, at alle kombinationer, som indeholder:

er forbudte, da de implicerer et minimum. De tilsvarende rækker i tabellen er helt sprunget over.

Vi giver to eksempler paa brugen af OPT1A. Først:

Program d-190. Prøve af OPT1A med een variabel.

```
begin
   integer q, numopt;
   real x, y1, y2, xopt, y1opt, y2opt, ginlet, delx;
   array xold, y1old, y2old[1:3];
   integer array numold[1:3];
   comment library OPT1A;
   comment library POL;
   for ginlet := 70, 74, 78 do
   begin
      trykvr;
      tryk({-ndd}, trykteksT({<ginlet =}), ginlet);
      trykvr;
      trykteksT({<tinlet  Prod.  Excess});
      q := 0;
      x := 400;
      delx := 10;
AA:   trykvr;
      tryk({-nddd}, x);
      y1 := POL(1, x, ginlet);
      y2 := 2.1 - POL(2, x, ginlet);
      tryk({-nddd.dd}, y1, y2);
      OPT1A(q, x, y1, y2, q+1, 300, delx, 500, xold, y1old, y2old, numold,
            xopt, y1opt, y2opt, numopt, F1, E1);
      go to AA;
```

```
F1:   trykvr;
      tryktekst({<FOUND});
      go to H1;
E1:   trykvr;
      tryktekst({<ERROR});
H1:   trykvr;
      tryk({-nddd}, xopt);
      tryk({-nddd.dd}, y1opt, y2opt);
      trykvr;
      trykvr
   end for ginlet
end program;
```

Resultatudskriften er:

```
ginlet =  70
tinlet  Prod.   Excess
  400   69.89    0.16
  410   71.24   -0.01
FOUND
  400   69.89    0.16


ginlet =  74
tinlet  Prod.   Excess
  400   71.83    0.26
  410   72.52    0.23
  420   72.80    0.13
  430   72.70    0.02
FOUND
  420   72.80    0.13
```

```
ginlet =  78
tinlet  Prod.   Excess
  400   72.54    0.40
  410   72.54    0.32
  390   72.21    0.45
FOUND
  400   72.54    0.40
```

Her har vi simuleret optimeringen af driften af en ammoniakkonverter ved hjælp af proceduren **POL**. For tre forskellige værdier af ginlet varieres tinlet i spring paa 10 grader til beregning af den maksimale ammoniakproduktion. Samtidigt skal overskuddet af den nedre varmeveksler være positivt.

OPT1A kan ogsaa bruges til optimering af funktioner af to variable. Dette er vist i nedenstaaende program:

```
Program d-191. Prøve af OPT1A med 2 variable.
begin
   integer q1, q2, numopt1, numopt2, num;
   real tinlet, ginlet, y1, y2, xopt1, xopt2, y1opt1, y1opt2, y2opt1,
        y2opt2, delt, delg;
   array xold1, xold2, y1old1, y1old2, y2old1, y2old2[1:3];
   integer array numold1, numold2[1:3];
   comment library OPT1A;
   comment library POL;
   num := 0;
   trykvr;
   tryktekst({<Nr. tinlet ginlet  Prod.   Excess});
   q1 := 0;
   tinlet := 400;
   delt := 10;
   ginlet := 70;
A1: q2 := 0;
   delg := 2;
A2: num := num + 1;
   trykvr;
   tryk({-nd}, num);
```

```
   y1 := POL(1, tinlet, ginlet);
   y2 := 2.1 - POL(2, tinlet, ginlet);
   tryk({-nddd}, tinlet, trykml(2), ginlet);
   tryk({-nddd.dd}, trykml(1), y1, y2);
   OPT1A(q2, ginlet, y1, y2, num, 66, delg, 80, xold2, y1old2, y2old2,
        numold2, xopt2, y1opt2, y2opt2, numopt2, F2, E2);
   go to A2;
E2: F2: OPT1A(q1, tinlet, y1opt2, y2opt2, numopt2, 380, delt, 460, xold1,
        y1old1, y2old1, numold1, xopt1, y1opt1, y2opt1, numopt1, F1, E1);
   ginlet := xopt2 - 2;
   go to A1;
F1: trykvr;
   tryktekst({<FOUND});
   go to H1;
E1: trykvr;
   tryktekst({<ERROR});
H1: trykvr;
   tryk({-nd}, numopt1);
   trykvr
end program;
```

Resultatudskriften er:

Nr.	tinlet	ginlet	Prod.	Excess
1	400	70	69.89	0.16
2	400	72	71.01	0.18
3	400	74	71.83	0.26
4	400	76	72.31	0.36
5	400	78	72.54	0.40
6	400	80	72.42	0.43
7	410	76	72.71	0.28
8	410	78	72.54	0.32
9	410	74	72.52	0.23
10	420	74	72.80	0.13
11	420	76	72.63	0.19
12	420	72	72.64	0.05
13	430	72	72.84	-0.05
14	430	74	72.70	0.02

```
FOUND
 10
```

Her finder vi optimum af ammoniakproduktionen ligesom før, men lader nu OPT1A variere baade tinlet og ginlet. For hver værdi af tinlet varieres ginlet med OPT1A, og det fundne optimum bruges i et andet kald af OPT1A, hvor tinlet varieres. Det er naturligvis en ret langsom metode, da der skal beregnes temmelig mange punkter.

5.2. Lineær Optimering (steepest ascent).

5.2.1. Proceduren OPT3. Hvis det ikke er tilstrækkeligt nøjagtigt at lade de uafhængige variable variere i spring af fast længde, maa man anvende metoder, som tillader en kontinuerlig variation. Som første eksempel herpaa giver vi proceduren OPT3, hvis deklaration er:

```
procedure OPT3(N, count, cycount, cymax, y, ymax, y0, y1, y2, delx, DELX,
        XMAX, SSQ, eps, x, dif, MORE, ERROR, FOUND);
integer N, count, cycount, cymax;
real y, ymax, y0, y1, y2, delx, DELX, XMAX, SSQ, eps;
array x, dif;
label MORE, ERROR, FOUND;
begin
   integer i, q;
   real R, A2, X;
   if count = 0 then
   begin
      y0 := ymax := y;
      SSQ := 0;
      go to L1
   end;
   q := count - N;
   if q>0 then go to L3;
   dif[count] := (y-y0)/delx;
   x[count] := x[count]-delx;
```

```
   SSQ := SSQ + dif[count]↑2;
   if q = 0 then go to L2;
L1: count := count + 1;
   x[count] := x[count] + delx;
   go to MORE;
L2: R := DELX/sqrt(SSQ);
   for i := 1 step 1 until N do dif[i] := R×dif[i];
L3: if q = 2 then go to L4;
   count := count + 1;
   if q = 1 then y1 := y;
   for i := 1 step 1 until N do x[i] := x[i] + dif[i];
   go to MORE;
L4: y2 := y;
   A2 := (y2-2×y1+y0)/DELX↑2;
   X := if A2<0 then
   -(4×y1-3×y0-y2)/2/DELX/A2
   else XMAX×sign(y2-y0);
   if abs(X) > XMAX then X := XMAX×sign(y2-y0);
   for i := 1 step 1 until N do
   begin
      x[i] := x[i]-2×dif[i];
      dif[i] := dif[i]×X/DELX
   end;
   if abs(X)< eps then go to FOUND;
   count := 0;
   cycount := cycount + 1;
   if cycount = cymax then go to ERROR;
   for i := 1 step 1 until N do x[i] := x[i] + dif[i];
   go to MORE
end of OPT-3;
```

Parametrene i OPT3 er:

integer N: Antallet af uafhængige variable.

integer count: En tæller, som brugeren skal sætte til nul før første kald. Proceduren tæller 1 frem her for hvert kald, men sætter count tilbage til nul, hver gang en ny cyklus paa N + 3 kald begynder.

integer cycount: Dette er ogsaa et tælleværk, hvor proceduren tæller 1 frem for hver cyklus. Skal nulstilles af brugeren før første kald.

integer cymax: Hvis cycount bliver lig med cymax, foretager proceduren et fejludhop til etiketten ERROR.

real y: Dette er et udtryk eller en real procedure, som skal give værdien af den undersøgte funktion, svarende til de aktuelle værdier af de uafhængige variable, x.

real ymax, y0, y1, y2: Benyttes af proceduren til lagring af mellemresultater. Værdien af y i begyndelsen af hver cyklus gemmes i ymax, som derfor normalt vil ende med at indeholde det søgte maksimum af y.

real delx: Tilvæksten i x, som benyttes ved den første lineære udforskning af funktionen. Der benyttes samme tilvækst i alle de uafhængige variable.

real DELX: Proceduren foretager to faste skridt langs retningen med stærkest stigning, først med tilvæksten DELX og derefter med tilvæksten 2×DELX.

real XMAX: Det tredje skridt langs denne retning maa højst have længden XMAX.

real SSQ: Bruges af proceduren til lagring af mellemresultater.

real eps: Naar det tredje skridt langs retningen med stærkest stigning bliver numerisk mindre end eps, anses maksimet for fundet, og der foretages et udhop til FOUND.

array x[1:N]: Talsættet med de uafhængige variable. Før første kald skal startværdierne være til raadighed her. Proceduren styrer den videre variation af x. Naar maksimet er fundet, staar løsningen i x.

array dif[1:N]: Benyttes af proceduren til lagring af tilvækster i x.

label MORE: Dette er det normale udhop. Hovedprogrammet maa da beregne den nye y-værdi svarende til det nye sæt x-værdier. Derefter kaldes OPT3 igen.

label ERROR: Hertil hoppes, hvis cycount = cymax.

label FOUND: Hertil hoppes, naar løsningen er fundet.

Proceduren virker paa følgende maade. Undersøgelsen er opdelt i to faser: først findes differenskvotienterne med hensyn til de N variable, og derefter foretages tre skridt i retningen af den stærkeste stigning (steepest ascent).

For N = 3 vil første fase bestaa af 3 kald (og basisberegningen) med følgende værdier af x:

count	x[1]	x[2]	x[3]
0	x[1]	x[2]	x[3]
1	x[1] + delx	x[2]	x[3]
2	x[1]	x[2] + delx	x[3]
3	x[1]	x[2]	x[3] + delx

I de tre kald for count = 1, 2 og 3 vil proceduren gemme differenskvotienterne i talsættet dif:

```
dif[count] := (y-y0)/delx;
```

Her er y den aktuelle y-værdi og y0 er y-værdien svarende til basisberegningen. Summen af kvadraterne:

```
dif[1]↑2 + dif[2]↑2 + .....
```

gemmes i cellen for **SSQ**.

I det sidste af de N kald vil proceduren korrigere dif-værdierne ved at dividere med sqrt(**SSQ**) og multiplicere med trinlængden, **DELX**:

```
for i := 1 step 1 until N do
dif[i] := dif[i]×DELX/sqrt(SSQ);
```

Derefter begynder anden fase, som altid omfatter tre kald (ogsaa for N ≠ 3), og hvor de første to kald bruger x-værdierne:

count	x[1]	x[2]	x[3]
N + 1	x[1]+dif[1]	x[2]+dif[2]	x[3]+dif[3]
N + 2	x[1]+2×dif[1]	x[2]+2×dif[2]	x[3]+2×dif[3]

Proceduren gemmer de to y-værdier svarende til disse to kald som y1 og y2.

Ud fra den maade, som dif-værdierne blev beregnet paa, ser vi, at afstanden i rummet fra basispunktet:

P: x[1], x[2], x[3]

til punktet:

Q: x[1]+dif[1], x[2]+dif[2], x[3]+dif[3]

netop er DELX. Komposanterne langs de tre koordinatakser er proportionale med funktionens gradient (differentialkvotient) i den retning, og vi bevæger os derfor fra punktet P til punktet Q langs retningen af den stærkeste stigning. Det næste punkt:

R: x[1]+2×dif[1], x[2]+2×dif[2], x[3]+2×dif[3]

ligger længere ude i samme retning, saaledes at afstanden PQ er lig afstanden QR.

Beregningens første fase giver os den retning, i hvilken funktionen vokser stærkest. Den anden fase skal nu fortælle os, hvor langt vi skal gaa ud i denne retning. Vi har de tre punkter, P, Q og R i den rigtige retning, og vi skriver nu funktionen, y, som et andengradspolynomium:

```
y := A×X↑2 + B×X + C;
```

X er afstanden fra basispunktet, P, og ud ad den rigtige retning. De tre konstanter, A, B og C, findes ved ved at indsætte de tre punkter:

	P	Q	R
X:	0	DELX	2×DELX
y:	y0	y1	y2

Dette giver:

```
A := (y2-2×y1+y0)/2/DELX↑2;
B := (4×y1-3×y0-y2)/(2×DELX);
```

Hvis A < 0 har parablen sit toppunkt ved

```
X := - B/(2×A);
```

eller:

```
X := - (4×y1-3×y0-y2)/(2×DELX)/(2×A);
```

Programmet beregner dette X. Hvis X er lille:

```
abs(X) < eps
```

betyder det, at den parabolske ekstrapolation langs den rigtige retning fører os til et punkt, der er tæt ved basispunktet, P. Proceduren anser da maksimet for fundet og hopper til FOUND. Ellers indsættes det nye punkt svarende til X som basispunkt og en ny cyklus med N + 3 kald begyndes. Da de hidtidige værdier af dif svarer til afstanden DELX, omregnes de:

```
for i := 1 step 1 until N do
dif[i] := dif[i]×X/DELX;
```

saaledes at de nu svarer til X. Hvis abs(X) > XMAX, sættes X ned, men med bevarelse af det rigtige fortegn. Endelig sættes de nye x-værdier:

```
for i := 1 step 1 until N do
x[i] := x[i] + dif[i];
```

og næste cyklus begynder.

Et simpelt eksempel paa brugen af OPT3 er:

Program d-192. Prøve af OPT3.

```
begin
   integer count, cycount;
   real y, ymax, y0, y1, y2, SSQ;
   array x, dif[1:2];
   comment library OPT3;
   count := cycount := 0;
   x[1] := 1;
   x[2] := 2;
   trykvr;
   tryktekst(‡<   x[1]        x[2]        y‡);
```

```
MORE:
    trykvr;
    if count = 0 then trykvr;
    y := 10 - 5×(x[1] - 5)↑2 - (x[2] - 5)↑2;
    tryk(‹-ndd.dddddd›, x[1], x[2], y);
    OPT3(2, count, cycount, 100, y, ymax, y0, y1, y2, 0.1, 0.2, 2, SSQ,
         1₁₀-4, x, dif, MORE, ERROR, FOUND);
FOUND:
    trykvr;
    trykteкst(‹<FOUND›);
    go to OUT;
ERROR:
    trykvr;
    trykteкst(‹<ERROR›);
OUT:
end program;
```

Resultatudskriften er:

x[1]	x[2]	y
1.000000	2.000000	-79.000000
1.100000	2.000000	-75.050000
1.000000	2.100000	-78.410000
1.197806	2.029546	-71.107011
1.395611	2.059091	-63.607037
2.978056	2.295457	-17.755844
3.078056	2.295457	-15.783900
2.978056	2.395457	-17.224935
3.171179	2.347451	-13.758944
3.364302	2.399446	-10.140417

4.909287	2.815404	5.186397
5.009287	2.815404	5.227110
4.909287	2.915404	5.613316
4.928274	3.014501	6.032070
4.947261	3.213598	6.794859
5.099154	4.806371	9.913351
5.199154	4.806371	9.764197
5.099154	4.906371	9.942076
4.902763	4.844195	9.928449
4.706372	4.882018	9.554992
4.993327	4.826753	9.969763
5.093327	4.826753	9.926436
4.993327	4.926753	9.994412
4.819491	4.925652	9.831554
4.645654	5.024551	9.371594
4.981080	4.833720	9.970561
5.081080	4.833720	9.939481
4.981080	4.933720	9.993817
4.820946	4.953542	9.837541
4.660813	5.073364	9.419378
4.975716	4.837733	9.970721
5.075716	4.837733	9.945005
4.975716	4.937733	9.993174
4.825060	4.969273	9.846036
4.674404	5.100813	9.459772
4.972201	4.840803	9.970792
5.072201	4.840803	9.948592
4.972201	4.940803	9.992632
4.829624	4.981060	9.854501
4.687047	5.121316	9.495585

4.969250	4.843706	9.970844
5.069250	4.843706	9.951594
4.969250	4.943706	9.992103
4.835007	4.991959	9.863822
4.700764	5.140211	9.532630
4.966218	4.847054	9.970901
5.066218	4.847054	9.954683
4.966218	4.947054	9.991491
4.842460	5.004165	9.875889
4.718702	5.161276	9.578348
4.962398	4.851903	9.970998
5.062398	4.851903	9.958600
4.962398	4.951903	9.990617
4.855556	5.020974	9.895240
4.748715	5.190044	9.648161
4.956222	4.861676	9.971284
5.056222	4.861676	9.965062
4.956222	4.961676	9.988949
4.889775	5.050316	9.936721
4.823328	5.238955	9.786836
4.942914	4.899459	9.973597
5.042914	4.899459	9.980684
4.942914	4.999459	9.983706
5.057721	5.063225	9.979344
5.172527	5.226991	9.799646
5.003875	4.986417	9.999740
5.103875	4.986417	9.945865
5.003875	5.086417	9.992457
4.805678	4.959623	9.809565
4.607481	4.932828	9.225132

```
5.000380   4.985945   9.999802
5.100380   4.985945   9.949422
5.000380   5.085945   9.992613
4.802386   4.957692   9.802953
4.604391   4.929439   9.212489
FOUND
```

Her har vi ønsket at finde maksimum af funktionen:

```
y := 10 - 5*(x1-5)↑2 - (x2-5)↑2;
```

Dette ligger i punktet (5,5), og niveaukurverne danner ellipser omkring dette punkt. Vi starter i punktet (1,1). Udskriften viser, at vi ret hurtigt kommer tæt paa det søgte maksimum, men saa gaar beregningen meget langsomt. Dette skyldes, at den lineære undersøgelsesmetode ikke er velegnet som tilnærmelse nær ved maksimet, hvor de lineære komponenter af funktionen er forsvindende, fordi differentialkvotienterne er nær ved nul. Her er en kvadratisk optimering mere velegnet, som vist i næste afsnit.

Der findes en speciel disciplin indenfor de numeriske metoder, som kaldes lineær programmering, og hvor man skal finde maksimum af en funktion, der selv er lineær eller kan tilnærmes med et lineært udtryk. Samtidigt forlanges det, at andre foreliggende lineære funktioner ikke antager negative værdier. Det ligner altsaa problemstillingen ved brugen af OPT1A, men er rent lineært. Procedurer til lineær programmering har endnu ikke været taget i brug ved GIER-installationen hos Haldor Topsøe.

5.3. Kvadratisk Optimering.

5.3.1. Proceduren OPT4. Ved kvadratisk optimering af en funktion af N variable tilnærmer man funktionen med et andengradspolynomium i de N variable, og løser de N lineære ligninger, som udsiger at de N førsteordens differentialkvotienter skal være nul. Proceduren OPT4 udfører en saadan beregning, og den arbejder iterativt paa samme maade som OPT3 og NOLEQ3. Deklarationen er:

```
procedure OPT4(count, cmax, var, xstart, del0x, xact, xold, epsx,
yact, yold, inveps, maxstepfac, FOUND, ERROR);
value cmax, var, inveps, maxstepfac;
integer count, cmax, var;
real yact, inveps, maxstepfac;
array xstart, del0x, xact, xold, epsx, yold;
label FOUND, ERROR;
begin
   integer corcount, fac, i, j, k, m, n, p;
   real R, S, T;
   array COEF, VEC, RES[1:(var+1)×(var+2):2],
         MATRIX[1:(var+1)×(var+2):2, 1:(var+1)×(var+2):2];
   procedure MULT(n, VEC, RES);
   value n;
   integer n;
   array VEC, RES;
   for i := 1 step 1 until n do
   begin
      R := 0;
      for j := 1 step 1 until n do
      R := R + MATRIX[i, j]×VEC[j];
      RES[i] := R
   end MULT;
   procedure FIT;
   begin
      for i := 1 step 1 until fac do
      begin
         MATRIX[i,1] := 1;
         k := 2×var + 1;
         for j := 1 step 1 until var do
         begin
            R := MATRIX[i,j+1] := xold[i,j];
            MATRIX[i,1+var+j] := R↑2;
            if j ≠ var then
            for m := j + 1 step 1 until var do
            begin
               k := k + 1;
```

```
                MATRIX[i,k] := R*xold[i,m]
            end for m
        end for j;
    end for i;
    INVERT2(fac, MATRIX, inveps, ERROR);
    MULT(fac, yold, COEF)
end FIT;
fac := (var+1)*(var+2):2;
corcount := count - (count - 1):fac*fac;
if count = 0 then
begin
    for i := 1 step 1 until var do
    xact[i] := xstart[i];
    go to OUT
end if count = 0;
yold[corcount] := yact;
if corcount < fac then
begin
    for i := 1 step 1 until var do
    xact[i] := xold[1,i];
    i := corcount;
    R := 1;
    if corcount <= 2*var then
    begin
        if corcount > var then
        begin
            i := i - var;
            R := -1
        end;
        xact[i] := xact[i] + R*del0x[i]
    end
    else
    begin
        p := 2*var;
        for i := 1 step 1 until var - 1 do
        for j := i + 1 step 1 until var do
```

```
            begin
               p := p + 1;
               if p = corcount then go to L2
            end;
L2:         xact[i] := xact[i] + del0x[i];
            xact[j] := xact[j] + del0x[j]
         end;
         go to OUT
      end if corcount < fac;
      FIT;
      for i := 1 step 1 until var do
      begin
         VEC[i] := - COEF[1+i];
         MATRIX[i,i] := 2×COEF[1+var+i]
      end for i;
      k := 1 + 2×var;
      for i := 1 step 1 until var - 1 do
      for j := i + 1 step 1 until var do
      begin
         k := k + 1;
         MATRIX[i,j] := MATRIX[j,i] := COEF[k]
      end for i and j;
      INVERT2(var, MATRIX, inveps, ERROR);
      MULT(var, VEC, xact);
      for i := 1 step 1 until var do
      if abs(xact[i]-xold[1,i]) > abs(epsx[i])
      then go to OUT;
      go to FOUND;
OUT:  count := count + 1;
      if count > cmax then go to ERROR;
      corcount := count  - (count - 1)÷fac×fac;
      if corcount = 1 ∧ count > 1 then
      begin
          S := 0;
          for i := 1 step 1 until var do
          begin
             R := abs((xact[i] - xold[1,i])/del0x[i]);
```

```
            if R > S then
            begin
               n := i;
               S := R
            end if larger
         end for i;
         if S > maxstepfac then
         for i := 1 step 1 until var do
         xact[i] := xold[1,i] + (xact[i] - xold[1,i])/S*maxstepfac
      end if;
      for i := 1 step 1 until var do
      xold[corcount,i] := xact[i]
   end OPT4;
```

Proceduren indeholder parametrene:

integer count: En tæller, som brugeren skal sætte til nul før første kald, og som proceduren øger med 1 i hvert kald.

integer cmax: Hvis count bliver større end cmax, laver OPT4 et fejludhop til etiketten: ERROR.

integer var: Antallet af uafhængige variable. Det maa gerne være 1.

array xstart[1:var]: Startværdier af de uafhængige variable.

array del0x[1:var]: Tilvækster i de uafhængige variable. De to talsæt xstart og del0x ændres ikke af proceduren.

array xact[1:var]: Efter hvert kald af OPT4 vil proceduren have indsat nye værdier af de uafhængige variable i dette talsæt. Brugeren behøver ikke at indsætte noget her før første kald.

array xold[1:fac, 1:var]: Bruges af proceduren til lagring af gamle sæt af x-værdier. Værdien af fac er (var+1)*(var+2):2 eller:

var:	1	2	3	4	5	6	7	8	9	10
fac:	3	6	10	15	21	28	36	45	55	66

Da proceduren opererer med en matrix af dimensionen [1:fac, 1:fac], er var = 5 normalt det højeste, der kan behandles med OPT4. Med flere variable maa man bruge OPT5.

array epsx[1:var]: Den tilladelige fejl ved beregning af maksimumpunktet. Naar den beregnede ændring i xact[i] er mindre end epsx[i] for alle variable, er beregningen slut, og proceduren hopper til etiketten: FOUND.

real yact: Efter hvert kald af OPT4 maa hovedprogrammet beregne funktionsværdien, yact, svarende til de aktuelle x-værdier, xact.

array yold[1:fac]: Bruges af proceduren til lagring af gamle yact-værdier.

real inveps: Det mindste tilladelige diagonalelement ved matrixinversionen. Denne parameter bruges i proceduren INVERT2. Hvis et diagonalelement bliver mindre end inveps, sker der fejludhop til ERROR.

real maxstepfac: Proceduren kontrollerer, at ændringen i den variable nr. i ikke overskrider maxstepfac×del0x[i] for alle i. Hvis ændringen er for stor, bliver alle ændringer reduceret tilsvarende.

label FOUND: Hertil hoppes, naar maksimet er fundet.

label ERROR: Dette bruges som fejludhop fra INVERT2, eller hvis count > cmax.

Proceduren opererer i cykler med fac kald i hver cyklus. Med tre variable er den første cyklus:

count før kaldet	Værdier af de variable efter kaldet			count efter kaldet
	xact[1]	xact[2]	xact[3]	
0	x1	x2	x3	1
1	x1 + d1	x2	x3	2
2	x1	x2 + d2	x3	3
3	x1	x2	x3 + d3	4
4	x1 - d1	x2	x3	5
5	x1	x2 - d2	x3	6
6	x1	x2	x3 - d3	7
7	x1 + d1	x2 + d2	x3	8
8	x1 + d1	x2	x3 + d3	9
9	x1	x2 + d2	x3 + d3	10

Vi har her for nemheds skyld skrevet xact[1] som x1 og del0x[1] som d1, og analogt for de andre variable.

I kald nr. fac + 1 har proceduren fac værdier af yact og de tilsvarende værdier af xact lagret i yold og xold. Den udtrykker nu funktionen yact som et andengradspolynomium i de variable. For tre variable kan det skrives som:

```
y := c0 + c1×x1    + c2×x2    + c3×x3
        + c4×x1↑2  + c5×x2↑2  + c6×x3↑2
        + c7×x1×x2 + c8×x1×x3 + c9×x2×x3;
```

De ukendte koefficienter, c0, c1, o.s.v., der i proceduren betegnes som array COEF[1:fac], beregnes nu ved løsning af de fac lineære ligninger, der fremkommer ved indsætning af sammenhørende værdier af x og y. Dette udføres af den lokale procedure, FIT, der opstiller matricen: array MATRIX [1:fac, 1:fac], inverterer den med INVERT2, der skal være til stede som en global procedure, og endelig finder COEF-vektoren ved multiplikation af MATRIX og yold-vektoren. Matrixmultiplikationen sker med en lokal procedure: MULT.

Det næste skridt efter beregning af koefficienterne bliver opstillingen af differentialkvotienterne ud fra polynomiet. For tre variable finder vi:

```
dy/dx1 = c1 + 2×c4×x1 +   c7×x2 +   c8×x3
dy/dx2 = c2 +   c7×x1 + 2×c5×x2 +   c9×x3
dy/dx3 = c3 +   c8×x1 +   c9×x2 + 2×c6×x3
```

I maksimumpunktet er de tre differentialkvotienter nul. Vi finder dette punkt ved løsning af de tre lineære ligninger, idet vi først inverterer matricen:

```
2×c4     c7     c8
 c7    2×c5     c9
 c8     c9    2×c6
```

med INVERT2 og multiplicerer den inverterede matrix med vektoren -c1, -c2, -c3.

Det nye sæt x-værdier, som fremkommer paa denne maade sammenlignes med basisberegningens x-værdier, der staar som første række i xold. Hvis ændringen er mindre end epsx[1] for alle variable, hoppes til FOUND. Ellers kontrolleres med maxstepfac, og der startes en ny cyklus.

Proceduren søger altsaa efter et punkt, hvor differentialkvotienterne er nul, men den undersøger ikke nærmere om der virkeligt er et maksimum her. Den kan ligesaa godt konvergere mod et minimum eller et saddelpunkt. Hvis der er flere maksima, minima eller saddelpunkter indenfor et givet omraade, kan man ikke paa forhaand vide, hvilket af disse, den finder.

Her er et eksempel paa brugen af OPT4:

Program d-142. Prøve af OPT4 med 2 variable.

```
begin
   integer count, third;
   boolean fin;
   real yact;
   array xstart, del0x, xact, epsx[1:2], xold[1:6, 1:2], yold[1:6];
   comment library OPT4;
   comment library INVERT2;
   for third := 0, 1 do
   begin
      xstart[1] := 1;
      xstart[2] := 2;
      del0x[1] := del0x[2] := 0.1;
      epsx[1] := epsx[2] := 1_10-4;
      count := 0;
      tryktekst({<
    x[1]        x[2]        yact

});
H1:   fin := false;
      OPT4(count, 1000, 2, xstart, del0x, xact, xold, epsx, yact, yold,
           1_10-8, 10, F1, E1);
      go to G2;
F1:   trykvr;
      tryktekst({<FOUND});
      go to G1;
```

```
E1:   trykvr;
      tryktekst({<ERROR});
G1:   fin := true;
G2:   trykvr;
      if (count-1):6×6 = count - 1 then trykvr;
      yact := 10 - 5×(xact[1] - 5)↑2 - (xact[2] - 5)↑2 -
           third×0.5×(xact[2] - 5)↑3;
      tryk({-nddd.dddddd}, xact[1], xact[2], yact);
      if -, fin then go to H1;
      trykvr;
      trykvr
   end for third
end of program;
```

Resultatudskriften er:

x[1]	x[2]	yact
1.000000	2.000000	-79.000000
1.100000	2.000000	-75.050000
1.000000	2.100000	-78.410000
0.900000	2.000000	-83.050000
1.000000	1.900000	-79.610000
1.100000	2.100000	-74.460000
2.000000	2.750071	-40.062180
2.100000	2.750071	-37.112180
2.000000	2.850071	-39.622195
1.900000	2.750071	-43.112180
2.000000	2.650071	-40.522166
2.100000	2.850071	-36.672194

3.000000	3.500154	-12.249538
3.100000	3.500154	-10.299538
3.000000	3.600154	-11.959569
2.900000	3.500154	-14.299538
3.000000	3.400154	-12.559507
3.100000	3.600154	-10.009569
4.000000	4.250108	4.437662
4.100000	4.250108	5.387662
4.000000	4.350108	4.577640
3.900000	4.250108	3.387662
4.000000	4.150108	4.277683
4.100000	4.350108	5.527640
4.999988	4.999943	10.000000
5.099988	4.999943	9.950012
4.999988	5.099943	9.990011
4.899988	4.999943	9.949988
4.999988	4.899943	9.989989
5.099988	5.099943	9.940023
4.999967	4.999830	10.000000
5.099967	4.999830	9.950033
4.999967	5.099830	9.990034
4.899967	4.999830	9.949967
4.999967	4.899830	9.989966
5.099967	5.099830	9.940066

FOUND

4.999968	4.999872	10.000000

x[1]	x[2]	yact
1.000000	2.000000	-65.500000
1.100000	2.000000	-61.550000
1.000000	2.100000	-66.215499
0.900000	2.000000	-69.550000
1.000000	1.900000	-64.714499
1.100000	2.100000	-62.265500
2.000000	2.268016	-32.268335
2.100000	2.268016	-29.318335
2.000000	2.368016	-32.811019
1.900000	2.268016	-35.318335
2.000000	2.168016	-31.663692
2.100000	2.368016	-29.861018
3.000000	2.576621	-8.756796
3.100000	2.576621	-6.806796
3.000000	2.676621	-9.127184
2.900000	2.576621	-10.806795
3.000000	2.476621	-8.333706
3.100000	2.676621	-7.177184
4.000000	2.953014	5.098440
4.100000	2.953014	6.048440
4.000000	3.053014	4.899519
3.900000	2.953014	4.048440
4.000000	2.853014	5.338770
4.100000	3.053014	5.849519
4.999985	3.483422	9.444063
5.099985	3.483422	9.394078
4.999985	3.583422	9.414625
4.899985	3.483422	9.394047
4.999985	3.383422	9.498997
5.099985	3.583422	9.364641

```
 4.999962    3.649021    9.407722
 5.099962    3.649021    9.357759
 4.999962    3.749021    9.413910
 4.899962    3.649021    9.357684
 4.999962    3.549021    9.422062
 5.099962    3.749021    9.363948

 4.999973    3.669008    9.407413
 5.099973    3.669008    9.357440
 4.999973    3.769008    9.417345
 4.899973    3.669008    9.357385
 4.999973    3.569008    9.417410
 5.099973    3.769008    9.367373

 4.999969    3.669315    9.407414
 5.099969    3.669315    9.357445
 4.999969    3.769315    9.417403
 4.899969    3.669315    9.357384
 4.999969    3.569315    9.417346
 5.099969    3.769315    9.367434
FOUND
 4.999971    3.669314    9.407414
```

Programmet finder først maksimum af funktionen:

```
y := 10 - 5*(x1-5)↑2 - (x2-5)↑2;
```

som ogsaa blev brugt til afprøvning af OPT3. Vi ser, at konvergensen er hurtigere med OPT4, og den ville have været endnu hurtigere, hvis vi havde brugt en større værdi af maxstepfac.

I det andet eksempel har vi modificeret funktionen lidt:

```
y := 10 - 5*(x1-5)↑2 - (x2-5)↑2 - 0.5*(x2-5)↑3;
```

Her finder proceduren ikke maksimumspunktet (5,5), men bliver hængende ved saddelpunktet ved (5, 3.6667) i stedet for.

5.3.2. Brug af OPT4 med Lagrangemultiplikator. Vi viser nu, hvorledes programeksemplet side 155 kan regnes paa en lidt anden maade med OPT4.

Vi simulerer beregningen med testproceduren **POL**, og skal finde maksimum af ammoniakproduktionen:

```
PRODUCTION := POL(1, tinlet, ginlet);
```

idet vi samtidigt forlanger, at den nødvendige højde af varmeveksleren bliver nøjagtigt 2 meter. Den anden funktion:

```
EXCESS := POL(2, tinlet, ginlet) - 2;
```

skal altsaa være nul. De to uafhængige variable er tinlet og ginlet (se side 129).

Optimering af en funktion med samtidigt krav om at een eller flere andre funktioner af de samme variable skal være nul kan løses ved hjælp af de saakaldte Lagrangemultiplikatorer. I stedet for de to funktioner: **PRODUCTION** og **EXCESS** danner vi en enkelt ny funktion:

```
yact := PRODUCTION + lambda×EXCESS;
```

Den nye variable, lambda, er den ubestemte Lagrangemultiplikator, der skal tilfredsstille betingelsen:

$$dyact/dlambda = 0$$

Endvidere har vi:

$$dyact/dtinlet = 0$$
$$dyact/dginlet = 0$$

og vi kan derfor løse problemet med OPT4 ved at optimere yact som en funktion af de tre variable: tinlet, ginlet og lambda. Her udnytter vi, at OPT4 ikke leder efter et sandt maksimum, men blot et punkt, hvor differentialkvotienterne er nul.

Hvis der er yderligere funktioner, som ogsaa skal være nul, maa de inkluderes i yact efter multiplikation med lambda2, lambda3, o.s.v.

Programmet er:

```
Program d-143.  Prøve af OPT4 med 3 variable.
begin
   integer count, i;
   boolean fin;
   real yact, PRODUCTION, EXCHANGER EXCESS;
   array xstart, del0x, xact, epsx, x, corr1, corr2[1:3], xold[1:10, 1:3],
         yold[1:10];
   comment library OPT4;
   comment library INVERT2;
   comment library POL;
   xstart[1] := 430;
   xstart[2] := 70;
   xstart[3] := -1;
   del0x[1] := 10;
   del0x[2] := 2;
   del0x[3] := 0.1;
   epsx[1] := 1;
   epsx[2] := 0.1;
   epsx[3] := 0.01;
   corr1[1] := 1/200;
   corr1[2] := 1/60;
   corr1[3] := 1;
   corr2[1] := -350;
   corr2[2] := -40;
   corr2[3] := 0;
   for i := 1 step 1 until 3 do
   begin
      xstart[i] := (xstart[i] + corr2[i])×corr1[i];
      del0x[i] := del0x[i]×corr1[i];
      epsx[i] := epsx[i]×corr1[i]
   end for i;
   count := 0;
   trykteksт(‡<
    x[1]      x[2]      x[3]    PRODUCTION   EXCHANGER    yact
  t-inlet   g-inlet   lambda                  EXCESS

‡);
```

```
H1: fin := false;
   OPT4(count, 100, 3, xstart, del0x, xact, xold, epsx, yact, yold, 1_10-8,
       4, F1, E1);
   go to G2;
F1: trykvr;
   tryktekst({<FOUND});
   go to G1;
E1: trykvr;
   tryktekst({<ERROR});
G1: fin := true;
G2: trykvr;
   if (count-1):10×10 = count - 1 then trykvr;
   for i := 1 step 1 until 3 do
   begin
      x[i] := xact[i]/corr1[i] - corr2[i];
      tryk({-nddd.dddd}, x[i])
   end for i;
   PRODUCTION := POL(1, x[1], x[2]);
   EXCHANGER EXCESS := POL(2, x[1], x[2]) - 2;
   yact := PRODUCTION + x[3]×EXCHANGER EXCESS;
   tryk({-nddd.dddd00}, PRODUCTION, EXCHANGER EXCESS, yact);
   if -, fin then go to H1
end of program;
```

Resultatudskriften er:

x[1] t-inlet	x[2] g-inlet	x[3] lambda	PRODUCTION	EXCHANGER EXCESS	yact
430.0000	70.0000	-1.0000	72.636262	0.272274	72.363988
440.0000	70.0000	-1.0000	72.853761	0.411905	72.441856
430.0000	72.0000	-1.0000	72.844714	0.150442	72.694272
430.0000	70.0000	-0.9000	72.636262	0.272274	72.391215
420.0000	70.0000	-1.0000	72.109605	0.186624	71.922981
430.0000	68.0000	-1.0000	72.082274	0.502946	71.579329
430.0000	70.0000	-1.1000	72.636262	0.272274	72.336761
440.0000	72.0000	-1.0000	72.719478	0.293042	72.426436
440.0000	70.0000	-0.9000	72.853761	0.411905	72.483047
430.0000	72.0000	-0.9000	72.844714	0.150442	72.709316
424.8474	72.7221	-0.6000	72.830394	0.063761	72.792137
434.8474	72.7221	-0.6000	72.750742	0.185743	72.639296
424.8474	74.7221	-0.6000	72.714892	-0.001202	72.715614
424.8474	72.7221	-0.5000	72.830394	0.063761	72.798513
414.8474	72.7221	-0.6000	72.534347	-0.027609	72.550912
424.8474	70.7221	-0.6000	72.580002	0.166281	72.480233
424.8474	72.7221	-0.7000	72.830394	0.063761	72.785761
434.8474	74.7221	-0.6000	72.350346	0.123702	72.276124
434.8474	72.7221	-0.5000	72.750742	0.185743	72.657871
424.8474	74.7221	-0.5000	72.714892	-0.001202	72.715493
422.3198	73.7077	-0.6204	72.819316	0.001167	72.818592
432.3198	73.7077	-0.6204	72.677222	0.119799	72.602895
422.3198	75.7077	-0.6204	72.620732	-0.055861	72.655390
422.3198	73.7077	-0.5204	72.819316	0.001167	72.818709
412.3198	73.7077	-0.6204	72.579050	-0.096112	72.638680
422.3198	71.7077	-0.6204	72.672102	0.085665	72.618953
422.3198	73.7077	-0.7204	72.819316	0.001167	72.818476
432.3198	75.7077	-0.6204	72.201788	0.065155	72.161364
432.3198	73.7077	-0.5204	72.677222	0.119799	72.614875
422.3198	75.7077	-0.5204	72.620732	-0.055861	72.649804

422.3758	73.7354	-0.4179	72.818978	0.000855	72.818620
432.3758	73.7354	-0.4179	72.670791	0.119663	72.620777
422.3758	75.7354	-0.4179	72.614346	-0.055918	72.637717
422.3758	73.7354	-0.3179	72.818978	0.000855	72.818706
412.3758	73.7354	-0.4179	72.585650	-0.096840	72.626124
422.3758	71.7354	-0.4179	72.678612	0.084634	72.643239
422.3758	73.7354	-0.5179	72.818978	0.000855	72.818535
432.3758	75.7354	-0.4179	72.190272	0.065269	72.162993
432.3758	73.7354	-0.3179	72.670791	0.119663	72.632744
422.3758	75.7354	-0.3179	72.614346	-0.055918	72.632125
FOUND					
422.3439	73.7246	-0.4181	72.819087	0.000864	72.818726

Der kræves altsaa fire cykler eller 40 kald af OPT4 for at finde maksimet. Vi skal senere (side 198) vise, hvorledes man kan komme ned paa 18 kald ved at bruge proceduren OPT7.

Bemærk iøvrigt, hvorledes vi har indført en lineær transformation af de uafhængige variable. Dette kan være nyttigt, hvis disse er af forskellig størrelsesorden.

Det er en ulempe ved den her viste brug af Lagrangemultiplikator, at man skal have en startværdi af lambda, som er af nogenlunde rigtig størrelsesorden og fortegn. Dette kan være svært at fremskaffe i praksis, da lambda ikke har nogen umiddelbar fysisk betydning. Man kan fortolke lambda som en slags bødekoefficient, hvormed man kan omregne en afvigelse i EXCESS fra nul til en ækivalent værdi af produktionen, men dette er næppe nogen større hjælp.

5.3.3. Proceduren OPT5. Med proceduren OPT5 kan man behandle et større antal variable end det er muligt med OPT4, idet talsættene lagres paa tromlen. Beregningerne er iøvrigt de samme. Matrixinversionen udføres med INVERT3 i stedet for INVERT2.

Deklarationen er:

```
procedure OPT5(count, cmax, var, drum, yact, inveps, maxstepfac, FOUND,
          ERROR);
value cmax, var, drum, inveps, maxstepfac;
integer count, cmax, var, drum;
real yact, inveps, maxstepfac;
label FOUND, ERROR;
begin
   integer corcount, fac, dr1, dr2, dr3, dr4, dr5, dr6, i,
           j, k, m, n, p;
   real R, S;
   array x, dx[1:var], ELEM[1:1], ROW, COEF[1:(var+1)×(var+2):2];
   procedure DRUM(place, a, from);
   value place, from;
   integer place;
   array a;
   boolean from;
   begin
      tromleplads := place;
      if from then fratromle(a) else tiltromle(a)
   end DRUM;
   procedure MULT(n, vec, respla);
   value n, respla;
   integer n, respla;
   array vec;
   begin
      integer i, j;
      array row, res[1:fac];
      for i := 1 step 1 until n do
      begin
         R := 0;
         DRUM(dr4 + i×fac, row, true);
         for j := 1 step 1 until n do R := R + row[j]×vec[j];
         res[i] := R
      end for i;
      DRUM(respla, res, false)
   end MULT;
   procedure FIT;
```

```
   begin
      ROW[1] := 1;
      for i := 1 step 1 until fac do
      begin
         DRUM(dr2+i×var, x, true);
         k := 1 + 2×var;
         for j := 1 step 1 until var do
         begin
            R := ROW[1+j] := x[j];
            ROW[1+var+j] := R↑2;
            if j ≠ var then
            for m := j+1 step 1 until var do
            begin
               k := k+1;
               ROW[k] := R×x[m]
            end for m
         end for j;
         DRUM(dr4+i×fac, ROW, false)
      end for i;
      INVERT3(fac, dr4, fac, inveps, ERROR);
      DRUM(dr4, COEF, true);
      MULT(fac, COEF, dr6)
   end FIT;
   fac := (var+1)×(var+2):2;
   dr1 := drum + 3×var;
   dr2 := dr1 + var;
   dr3 := dr2 + fac×var;
   dr4 := dr3 + fac;
   dr5 := dr4 + fac;
   dr6 := dr5 + fac↑2;
   corcount := count - (count-1):fac×fac;
   if count = 0 then
   begin
      DRUM(drum+var, x, true);
L1:   DRUM(dr1, x, false);
      go to OUT
   end if first call;
```

```
      DRUM(dr4, ROW, true);
      ROW[corcount] := yact;
      DRUM(dr4, ROW, false);
      if corcount < fac then
      begin
         DRUM(dr2 + var, x, true);
         DRUM(drum + 2×var, dx, true);
         i := corcount;
         R := 1;
         if corcount ≤ 2×var then
         begin
            if corcount > var then
            begin
               i := i - var;
               R := -1
            end;
            x[i] := x[i] + R×dx[i]
         end
         else
         begin
            p := 2×var;
            for i := 1 step 1 until var - 1 do
            for j := i + 1 step 1 until var do
            begin
               p := p + 1;
               if p = corcount then go to L2
            end;
L2:         x[i] := x[i] + dx[i];
            x[j] := x[j] + dx[j]
         end;
         go to L1
      end if corcount < fac;
      FIT;
      DRUM(dr6, COEF, true);
      k := 1 + 2×var;
      for i := 1 step 1 until var do
```

```
    begin
       DRUM(dr4 + i×fac, ROW, true);
       ROW[i] := 2×COEF[1 + var + i];
       for j := i + 1 step 1 until var do
       begin
          k := k + 1;
          ELEM[1] := ROW[j] := COEF[k];
          DRUM(dr4 + j×fac + i - fac, ELEM, false)
       end for j;
       DRUM(dr4 + i×fac, ROW, false)
    end for i;
    for i := 1 step 1 until var do
    x[i] := - COEF[1+i];
    INVERT3(var, dr4, fac, inveps, ERROR);
    MULT(var, x, dr4);
    DRUM(dr4, ROW, true);
    for i := 1 step 1 until var do
    x[i] := ROW[i];
    DRUM(dr1, x, false);
    DRUM(dr2 + var, dx, true);
    for i := 1 step 1 until var do
    x[i] := x[i] - dx[i];
    DRUM(dr2, dx, true);
    for i := 1 step 1 until var do
    if abs(x[i]) > abs(dx[i]) then go to OUT;
    go to FOUND;
OUT: count := count + 1;
    if count > cmax then go to ERROR;
    corcount := count - (count-1):fac×fac;
    if corcount = 1 ∧ count > 1 then
    begin
       S := 0;
       DRUM(drum + 2×var, dx, true);
       for i := 1 step 1 until var do
       begin
          R := abs(x[i]/dx[i]);
          if R > S then
```

```
            begin
               n := i;
               S := R
            end if larger
         end for i;
         if S > maxstepfac then
         begin
            DRUM(dr2 + var, dx, true);
            for i := 1 step 1 until var do
            x[i] := dx[i] + x[i]/S×maxstepfac;
            DRUM(dr1, x, false)
         end if larger
      end if check;
      DRUM(dr1, x, true);
      DRUM(dr2 + corcount×var, x, false)
end OPT5;
```

De simple parametre er ens i OPT4 og OPT5. Men i OPT5 er alle array-parametre erstattet af en enkelt integer parameter: drum. Tromleplads for de tromlelagrede talsæt fremgaar af følgende oversigt:

Array	Dimensioner	Tromleplads for sidste element
xstart	[1:var]	drum + var
del0x	[1:var]	drum + 2×var
xact	[1:var]	drum + 3×var = dr1
epsx	[1:var]	drum + 4×var = dr2
xold	[1:fac, 1:var]	dr2 + fac×var = dr3
yold	[1:fac]	dr3 + fac = dr4
matr	[1:fac, 1:fac]	dr4 + fac↑2 = dr5
coef	[1:fac]	dr5 + fac = dr6

De seks første af disse talsæt er de samme som de formelle parametre i OPT4. De to sidste talsæt: matr og coef, svarer til de lokale talsæt MATRIX og COEF i OPT4. De seks variable dr1 - dr6 er lokale for OPT5.

5.3.4. Proceduren OPT7. Proceduren OPT7 udfører de samme beregninger som OPT4, men der er indført enkelte nye finesser:

1. Foruden hovedfunktionen af var variable kan proceduren samtidigt behandle andre funktioner af de samme variable. Proceduren finder det punkt, hvor hovedfunktionen har maksimum (eller minimum eller saddelpunkt), idet de andre funktioner (fejlfunktionerne) samtidigt er nul.

2. Proceduren kan køres paa en lidt anden maade, saaledes at hovedfunktionen indstilles nøjagtigt paa nul ved at variere paa den første uafhængige variable (xact[1]), og saaledes at de partielle differentialkvotienter af hovedfunktionen med hensyn til de andre variable (xact[2], xact[3], o.s.v.) bliver nul.

Deklarationen af OPT7 er:

```
procedure OPT7(count, cmax, var, error, out, fixmax, xstart, del0x,
      xact, xold, epsx, yact, yold, inveps, maxstepfac, FOUND, ERROR);
value cmax, var, error, fixmax, inveps, maxstepfac;
boolean out, fixmax;
integer count, cmax, var, error;
real inveps, maxstepfac;
array xstart, del0x, xact, xold, epsx, yact, yold;
label FOUND, ERROR;
begin
   integer corcount, count2, count3, fac, fac2, i, j, k, m, n, p;
   real MIN, R, S, T;
   boolean good;
   array VEC, RES[1:(var+1)×(var+2)÷2], MATRIX[1:(var+1)×(var+2)÷2,
   1:(var+1)×(var+2)÷2], COEF[1:(var+1)×(var+2)÷2, 1:1 + error];
   procedure MULT(n, VEC, RES);
   value n;
   integer n;
   real VEC, RES;
   for i := 1 step 1 until n do
   begin
      R := 0;
      for j := 1 step 1 until n do
      R := R + MATRIX[i, j]×VEC;
      RES := R
```

```
end MULT;
procedure FIT;
begin
   for i := 1 step 1 until fac do
   begin
      MATRIX[i,1] := 1;
      k := 2×var + 1;
      for j := 1 step 1 until var do
      begin
         R := MATRIX[i,j+1] := xold[i,j];
         MATRIX[i,1+var+j] := R↑2;
         if j ≠ var then
         for m := j + 1 step 1 until var do
         begin
            k := k + 1;
            MATRIX[i,k] := R×xold[i,m]
         end for m
      end for j
   end for i;
   INVERT2(fac, MATRIX, inveps, ERROR);
   for p := 1 step 1 until 1 + error do
   MULT(fac, yold[j, p], COEF[i, p])
end FIT;
real procedure POLYN(n);
value n;
integer n;
begin
   R := COEF[1, n];
   for i := 1 step 1 until var do
   begin
      T := xact[i];
      R := R + T×COEF[1+i, n] + T↑2×COEF[1+var+i, n]
   end for i;
   k := 1 + 2×var;
   for m := 1 step 1 until var -1 do
   for j := m + 1 step 1 until var do
   begin
```

```
      k := k + 1;
      R := R + COEF[k, n]×xact[m]×xact[j]
   end for m, j;
   POLYN := R
end POLYN;
real procedure DERIV(n);
value n;
integer n;
begin
   R := COEF[1+n, 1] + 2×xact[n]×COEF[1+var+n, 1];
   k := 1 + 2×var;
   for m := 1 step 1 until var - 1 do
   for j := m+1 step 1 until var do
   begin
      k := k + 1;
      if m = n ∨ j = n then
      R := R + COEF[k, 1]×xact[if m = n then j else m]
   end for m, j;
   DERIV := R
end DERIV;
fac := (var+1)×(var+2):2;
corcount := count - (count - 1):fac×fac;
fac2 := (var-error+1)×(var-error+2):2;
if count = 0 then
begin
   for i := 1 step 1 until var do
   xact[i] := xstart[i];
   go to OUT
end if count = 0;
for i := 1 step 1 until 1 + error do
yold[corcount, i] := yact[i];
if corcount < fac then
begin
   for i := 1 step 1 until var do
   xact[i] := xold[1,i];
   i := corcount;
   R := 1;
```

```
        if corcount ≤ 2×var then
        begin
           if corcount > var then
           begin
              i := i - var;
              R := -1
           end;
           xact[i] := xact[i] + R×del0x[i]
        end
        else
        begin
           p := 2×var;
           for i := 1 step 1 until var - 1 do
           for j := i + 1 step 1 until var do
           begin
              p := p + 1;
              if p = corcount then go to L2
           end;
L2:        xact[i] := xact[i] + del0x[i];
           xact[j] := xact[j] + del0x[j]
        end;
        go to OUT
     end if corcount < fac;
     FIT;
     count2 := 0;
     good := false;
     if error > 0 then
     begin
        array Xst, delX, Xact, epsX, Xbest, Yact, YO[1:error],
              Yold[1:error, 1:error], Xold[1:fac2, 1:var-error];
        boolean procedure outside;
        begin
           outside := false;
           for i := 1 step 1 until error do
           if abs((Xact[i] - Xst[i])/delX[i]) > 4
           then outside := true
        end outside;
```

```
        if error < var then
L4:     OPT7(count2, cmax, var-error, 0, out, fixmax, xstart, del0x, xact,
          Xold, epsx, yact, yold, inveps, maxstepfac, F2, ERROR);
F4:     count3 := 0;
        for i := 1 step 1 until error do
        begin
           n := var - error + i;
           Xst[i] := Xbest[i] := xstart[n];
           delX[i] := del0x[n]/2;
           epsX[i] := epsx[n]
        end for i;
        MIN := 1_10100;
L3:     NOLEQ3(error, count3, cmax:2, outside, Xst, delX, Xact, epsX,
              Yact, Yold, Y0, inveps, maxstepfac, F3, E1);
        for i := 1 step 1 until error do xact[var-error+i] := Xact[i];
        S := 0;
        for p := 1 step 1 until error do
        begin
           Yact[p] := R := POLYN(1+p);
           S := S + R↑2
        end for p;
        if S < MIN then
        begin
           MIN := S;
           for i := 1 step 1 until error do Xbest[i] := Xact[i]
        end if better;
        go to L3;
F2:     good := true;
        go to F4;
E1:     for i := 1 step 1 until error do Xact[i] := Xbest[i];
F3:     for i := 1 step 1 until error do xact[var-error+i] := Xact[i];
        yact[1] := POLYN(1);
        if error < var ∧ ¬ good then go to L4
      end if error > 0
      else
      if ¬ fixmax then
```

```
      begin
         for i := 1 step 1 until var do
         begin
            VEC[i] := - COEF[1+i, 1];
            MATRIX[i,i] := 2×COEF[1+var+i, 1]
         end for i;
         k := 1 + 2×var;
         for i := 1 step 1 until var - 1 do
         for j := i + 1 step 1 until var do
         begin
            k := k + 1;
            MATRIX[i,j] := MATRIX[j,i] := COEF[k,1]
         end for i, j;
         INVERT2(var, MATRIX, inveps, ERROR);
         MULT(var, VEC[j], xact[i])
      end normal optimization
      else
      begin
         array Xbest, Yact, YO[1:var], Yold[1:var, 1:var];
         boolean procedure outside;
         begin
            outside := false;
            for i := 1 step 1 until var do
            if abs((xact[i] - xstart[i])/del0x[i]) > 4 then outside := true
         end outside;
F4:      count3 := 0;
         for i := 1 step 1 until var do Xbest[i] := xstart[i];
         MIN := 1_10100;
L3:      NOLEQ3(var, count3, cmax:2, outside, xstart, del0x, xact, epsx,
                  Yact, Yold, YO, inveps, maxstepfac, F3, E1);
         R := Yact[1] := POLYN(1);
         S := R↑2;
         for p := 2 step 1 until var do
         begin
            Yact[p] := R := DERIV(p);
            S := S + R↑2
         end for p;
```

```
          if S < MIN then
          begin
             MIN := S;
             for i := 1 step 1 until var do Xbest[i] := xact[i]
          end if better;
          go to L3;
E1:       for i := 1 step 1 until var do xact[i] := Xbest[i];
F3:    end if fixmax;
       for i := 1 step 1 until var do
       if abs(xact[i]-xold[1,i]) > abs(epsx[i]) then go to OUT;
       go to FOUND;
OUT:   count := count + 1;
       if count > cmax then go to ERROR;
       corcount := count  - (count - 1)*fac*fac;
       if corcount = 1 ∧ count > 1 then
       begin
           S := 0;
           for i := 1 step 1 until var do
           begin
              R := abs((xact[i] - xold[1,i])/del0x[i]);
              if R > S then
              begin
                 n := i;
                 S := R
              end if larger
           end for i;
           if S > maxstepfac then
           for i := 1 step 1 until var do
           xact[i] := xold[1,i] + (xact[i] - xold[1,i])/S*maxstepfac;
           S := 1;
           for S := 0.5*S while out do
           for i := 1 step 1 until var do
           xact[i] := xold[1, i] + (xact[i] - xold[1, i])*S
        end if;
       for i := 1 step 1 until var do
       xold[corcount,i] := xact[i]
    end OPT7;
```

De fleste formelle parametre i OPT7 er de samme som i OPT4 (se side 170). Parametrene er:

integer count, cmax, var: Som i OPT4.

integer error: Antallet af fejlfunktioner, som skal indstilles paa nul. For error = 0 behandles ingen fejlfunktioner.

boolean out: Dette er normalt en boolean procedure og den kaldes ved name. Den maa give værdien sand, hvis det aktuelle sæt af de uafhængige variable, xact, er udenfor et tilladt omraade. Ellers skal den sættes til falsk.

boolean fixmax: Hvis denne specificeres som falsk, udføres en normal optimering. Sættes den til sand, vil proceduren variere xact[1] indtil hovedfunktionen bliver nul og saaledes at differentialkvotienterne med hensyn til de andre variable bliver nul.

array xstart, del0x, xact, epsx[1:var]: Som i OPT4.

array xold[1:fac, 1:var]: Som i OPT4.

array yact[1:1 + error]: I OPT4 var yact en simpel real. Efter hvert kald af OPT7 skal hovedprogrammet beregne yact[1] som værdien af hovedfunktionen, der skal optimeres eller indstilles paa nul. Værdierne af yact[2], yact[3], o.s.v. skal ogsaa beregnes som værdien af fejlfunktionerne. Alle funktionsværdier skal svare til det aktuelle sæt af xact.

array yold[1:fac, 1:1+error]: Som i OPT4, men har nu en dimension mere.

real inveps, maxstepfac: Som i OPT4.

label FOUND, ERROR: Som i OPT4.

Proceduren skal have adgang til de to globale procedurer INVERT2 og NOLEQ3.

Der benyttes følgende lokale procedurer:

MULT udfører multiplikation af en matrix og en vektor.

FIT finder koefficienterne i de polynomier, som tilnærmer de 1 + error funktioner: yact.

POLYN(n) finder polynomieværdien af funktion nr. n og for et givet sæt af xact.

DERIV(n) finder differentialkvotienten af hovedfunktionen med hensyn til den variable nr. n og for et givet sæt af xact.

Beregningen foregaar saaledes:

Ligesom for OPT4 regnes i cykler paa fac kald af proceduren. Efter hver cyklus finder proceduren FIT koefficienterne i et andengradspolynomium, som tilnærmer de 1 + error funktioner. Koefficienterne lagres som det lokale talsæt COEF[1:fac, 1:1+error].

Hvis error > 0, skal proceduren søge at reducere fejlfunktionerne til nul. Hvis antallet af fejlfunktioner er mindre end antallet af variable (error < var) foretages et rekursivt kald af OPT7 idet vi nu arbejder med var - error variable og ingen fejlfunktioner. I det oprindelige talsæt xact[1:var] skal de første var - error elementer indstilles til optimum, idet vi samtidigt regulerer de sidste elementer: xact[var-error+1: var], saaledes at fejlfunktionerne er nul. Denne sidste nulindstilling sker med NOLEQ3. I det rekursive kald af OPT7 med dertil hørende kald af NOLEQ3 beregnes alle funktionsværdier ud fra polynomierne. Hvis error = var, kaldes OPT7 ikke, kun NOLEQ3.

Vi eliminerer altsaa de sidste error variable ud fra fejlbetingelserne og faar en almindelig optimering af de resterende var-error variable.

Hvis error = 0 og fixmax er false er beregningen den samme som i OPT4.

Hvis error = 0 og fixmax er true, løser proceduren et sæt af var ulinære ligninger med NOLEQ3. Af de var funktioner, der skal reduceres til nul, er den første hovedfunktionen og de øvrige differentialkvotienterne af hovedfunktionen med hensyn til de variable nr. 2, 3, o.s.v. Baade hovedfunktionen og differentialkvotienterne findes af polynomierne.

I de rekursive kald af OPT7 og i kaldene af NOLEQ3 er værdien af cmax sat til halvdelen af den originale værdi af cmax. I disse kald benyttes ogsaa en speciel boolean outside, som tillader variation af xact noget ud over det omraade, hvor polynomiet er opstaaet. Da polynomierne kun er tilnærmelser til de originale funktioner, holdes der i NOLEQ3-kaldene kontrol med at den bedst mulige løsning bliver indsat, hvis der ikke er nogen helt rigtig løsning.

Til afprøvning af OPT7 anvendtes følgende program:

Program d-172. Prøve af OPT7.

```
begin
   integer count, i, type;
   boolean fin;
   real PRODUCTION, EXCHANGER EXCESS;
```

```
    array xstart, del0x, xact, epsx, x, corr1, corr2[1:2], yact[1:2],
          xold[1:6, 1:2], yold[1:6, 1:2];
    comment library OPT7;
    comment library INVERT2;
    comment library POL;
    comment library NOLEQ3;
    for type := 1, 2 do
    begin
       xstart[1] := if type = 1 then 420 else 405;
       xstart[2] := if type = 1 then 72 else 77;
       del0x[1] := 2;
       del0x[2] := 2;
       epsx[1] := 1;
       epsx[2] := 0.1;
       corr1[1] := 1/200;
       corr1[2] := 1/60;
       corr2[1] := -350;
       corr2[2] := -40;
       for i := 1 step 1 until 2 do
       begin
          xstart[i] := (xstart[i] + corr2[i])×corr1[i];
          del0x[i] := del0x[i]×corr1[i];
          epsx[i] := epsx[i]×corr1[i]
       end for i;
       count := 0;
       tryktekst(‹
        x[1]      x[2]   PRODUCTION   EXCHANGER
  t-inlet   g-inlet                    EXCESS

);
H1:    fin := false;
       OPT7(count, 20, 2, if type = 1 then 1 else 0, false, type = 2,
            xstart, del0x, xact, xold, epsx, yact, yold, 1_10-8, 4, F1, E1);
       go to G2;
F1:    trykvr;
       tryktekst(‹FOUND›);
       go to G1;
```

```
E1:   trykvr;
      tryktekst(«<ERROR>);
G1:   fin := true;
G2:   trykvr;
      if (count-1):6x6 = count - 1 then trykvr;
      for i := 1 step 1 until 2 do
      begin
         x[i] := xact[i]/corr1[i] - corr2[i];
         tryk(«-nddd.dddd», x[i])
      end for i;
      yact[1] := PRODUCTION := POL(1, x[1], x[2]) -
          (if type = 2 then 72.6 else 0);
      yact[2] := EXCHANGER EXCESS := POL(2, x[1], x[2]) - 2;
      tryk(«-nddd.dddd00», PRODUCTION, EXCHANGER EXCESS);
      if -, fin then go to H1;
      trykvr; trykvr
   end for type
end of program;
```

Resultatudskriften var:

x[1] t-inlet	x[2] g-inlet	PRODUCTION	EXCHANGER EXCESS
420.0000	72.0000	72.635491	0.051393
422.0000	72.0000	72.706795	0.067741
420.0000	74.0000	72.801409	-0.033774
418.0000	72.0000	72.548643	0.036520
420.0000	70.0000	72.109605	0.186624
422.0000	74.0000	72.810282	-0.011943

421.9087	73.5933	72.818035	0.000568
423.9087	73.5933	72.825367	0.022652
421.9087	75.5933	72.650475	-0.057541
419.9087	73.5933	72.795063	-0.020690
421.9087	71.5933	72.637085	0.088680
423.9087	75.5933	72.594550	-0.035543
422.1792	73.7002	72.818817	-0.000129
424.1792	73.7002	72.820667	0.022199
422.1792	75.7002	72.625896	-0.057200
420.1792	73.7002	72.801462	-0.021678
422.1792	71.7002	72.665882	0.084920
424.1792	75.7002	72.565145	-0.035159
FOUND			
422.1711	73.7031	72.818733	-0.000317

x[1]	x[2]	PRODUCTION	EXCHANGER
t-inlet	g-inlet		EXCESS
405.0000	77.0000	0.029361	-0.238513
407.0000	77.0000	0.060956	-0.221195
405.0000	79.0000	-0.130737	-0.286179
403.0000	77.0000	-0.023496	-0.256536
405.0000	75.0000	-0.159498	-0.190883
407.0000	79.0000	-0.174221	-0.272559
403.8427	77.3314	0.005476	-0.256155
405.8427	77.3314	0.035590	-0.238758
403.8427	79.3314	-0.150079	-0.297132
401.8427	77.3314	-0.044752	-0.274159
403.8427	75.3314	-0.164918	-0.207545
405.8427	79.3314	-0.199459	-0.285480
FOUND			
403.8427	77.3314	0.005476	-0.256155

Prøveprogrammet arbejder med proceduren POL, der simulerer beregning af ammoniakproduktion og den nødvendige højde af varmeveksleren i en quench-konverter. De to uafhængige variable er tinlet og ginlet. Afprøvningen bestaar af to dele, og der anvendes to variable i begge dele.

I første gennemregning bruges:

```
error := 1
fixmax := false;
```

og vi faar da nøjagtig det samme problem, som blev løst med OPT4 i programmet side 179. Vi finder den maksimale ammoniakproduktion med en varmeveksler paa 2 meter. Her bruges kun 18 kald af OPT7 mod 40 kald af OPT4.

I anden gennemregning bruges:

```
error := 0;
fixmax := true;
```

Her søger vi et punkt, hvor ammoniakproduktionen er nøjagtigt 72.6 t/24 hr og hvor differentialkvotienten af ammoniakproduktionen med hensyn til ginlet er nul. Proceduren varierer tinlet, indtil produktionen passer. Varmevekslerens højde benyttes ikke. Her kræves 12 kald af OPT7.

6. REFERENCER

Ascher, M. og Forsythe, G.E.: Journ. ACM, 5, 9-21 (1958).

Cadwell, J.H.: Comp. Journ. 3, 266 - 269 (1960).

Fisher, R.A.: Statistical Methods for Research Workers (1948).

Frøberg, C.-E.: Lærobok i numerisk analys. Stockholm (1962).

Kjær, J.: Calculation of Ammonia Converters on an Electronic Digital computer. Akademisk Forlag, København (1963b).

Lapidus, L.: Digital Computation for Chemical Engineers (1962).

Naur, P.: Comm. ACM. 6, 40 (1963).

Naur, P.: Automatic grading of students ALGOL programming. BIT, 4, 177 - 188 (1964).

Schwarz, H.R.: Comm. ACM, 5, 94 (1962).

7. STIKORDSREGISTER

8. APPENDIKS

Dette appendiks omfatter oplysninger om de fire programmer:

LINEAR EQUATIONS-4

PA-6

PA-7

PA-8

8.1. Inputspecifikationer.

8.1.1. Inputspecifikationer til LINEAR EQUATIONS-4

0. Cover Data:

Ff
..... Calculation No.
[..... File No.
[..... Cover Page Text
[..... - - -
[..... - - -
e Stop e

1. Headline Data:

[..... Section Headline Tex
[..... - - -
[..... - - -
e Stop e

2. Calculation Parameters:

		Built in values:
(0).....	Number of unknowns, N($1 \leq N \leq 88$)	1
(1).....	Number of right-hand sides, NRS($0 \leq NRS$)	1
(2).....	Save original matrix: 0: no, 1: yes	1
(3).....	Print original matrix: 0: no, 1: yes	1
(4).....	Print inverse matrix: 0: no, 1: yes	0
(5).....	Minimum inversion pivot	1_{10}-12

e Stop e

3. Coefficient Matrix:

Punch row-wise. Action characters d and q can be used within a single row only.

a[1,1]	a[1,2]	a[1,3]	a[1,4] etc.		
.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....

e Stop e

4. Right Hand Sides:

a[1,N+1]	a[2,N+1]	a[3,N+1]	etc.	
.....	.....	.....	.....	.....
e	Stop e			

This data group must be specified NRS times, each ending with a stop-e. Use no stop e when NRS = 0.

z Final Stop

8.1.2. Inputspecifikationer til PA-6

0. Cover Data:

Ff

..... Calculation No.

[..... File No.

[..... Cover Page Text

[..... - - -

[..... - - -

e Stop e

1. Headline Data:

[..... Section Headline Text

[..... - - -

[..... - - -

e Stop e

2. Calculation Parameters:

(0)..... Number of variables

(1)..... Number of observations

(2)..... Correction type for y (1)

(3)..... - - - x1 (1)

(4)..... - - - x2 (1)

etc.

e Stop e

(1) 0: No correction

1: Use ln(y) or ln(x)

3. Data Material: (punch row-wise).

Identifi-cation number	value of y	value of x1	value of x2	value of x3	etc
.....	.....	.....	.....	.....	
.....	.....	.....	.....	.....	
.....	.....	.....	.....	.....	
.....	.....	.....	.....	.....	
.....	.....	.....	.....	.....	
.....	.....	.....	.....	.....	
.....	.....	.....	.....	.....	
.....	.....	.....	.....	.....	
etc.					

e Stop e

z Final Stop

8.1.3. Inputspecifikationer til PA-7

0. Cover Data:

Ff

..... Calculation No.

[..... File No.

[..... Cover Page Text

[..... - - -

[..... - - -

e Stop e

1. Headline Data:

[..... Section Headline Text

[..... - - -

[..... - - -

e Stop e

2. General Data:

		Built-in value:
(0)	V, number of variables	0
(1)	F, number of functions	0
(2)	Print polynomial coefficients: 0: no, 1: yes	0
(3)	Print original z-table: 0: no, 1: yes	0
(4)	Correction type for z-function (1)	0
(5)	Maximum sum of degrees in all variables	0
(6)	Calculate extended table: 0: no, 1: yes	0
(7)	Polynomial function analysis type (2)	0
	Data for analysis:	
(8)	Maximum number of iterations	0
(9)	Minimum pivot in inversions	0
(10)	Maximum step factor	0
(11)	Desired value of function 1	0
(12)	- - - - 2	0
(13)	- - - - 3	0
(14)	- - - - 4	0
(15)	- - - - 5	0
(16)	- - - - 6	0

e Stop e

(1) 0: No correction
1: Take natural logarithm

(2) 0: No analysis
1: Find maximum of first function
2: Find maximum of first function with specified values of remaining functions
3: Find specified values of all functions
4: Vary first variable to fixed value of first function and maximize for other variables

3. Data for Variables and Degrees:

	Number of values	Maximum polynomial degree	Correction type (1)	Variable No.
(0-2)	.....	.....	.....	1
(3-5)	.....	.....	.....	2
(6-8)	.....	.....	.....	3
(9-11)	.....	.....	.....	4
(12-14)	.....	.....	.....	5
(15-17)	.....	.....	.....	6
(18-20)	.....	.....	.....	7
(21-23)	.....	.....	.....	8

e Stop e

(1)

Correction type:	Normalize:	Take nat. log.	Take reciprocal
0	no	no	no
1	yes	no	no
2	no	yes	no
3	yes	yes	no
4	no	no	yes
5	yes	no	yes
6	no	yes	yes
7	yes	yes	yes

4. Values of Independent Variables:

....	....	....	....	....
....	....	....	....	....
....	....	....	....	....
....	....	....	....	....
....	....	....	....	....
....	....	....	....	....
....	....	....	....	....
....	....	....	....	....

e Stop e

This data group must be given V times, once for each variable. Each group ends with a stop-e.

5. Values of the Function:

....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....
....	....	....	....	....	....	....	....

.....	.....	.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....	.....	.....

e Stop e

This group must be given F times, once for each function. Each group ends with a stop-e.

6. Data for Extended Table Calculation:

Skip this data group and the stop-e, if no table calculation has been specified.

	Number of values	Start value	Increment	Variable no.
(0-2)	.....	.....	.....	1
(3-5)	.....	.....	.....	2
(6-8)	.....	.....	.....	3
(9-11)	.....	.....	.....	4
(12-14)	.....	.....	.....	5
(15-17)	.....	.....	.....	6
(18-20)	.....	.....	.....	7
(21-23)	.....	.....	.....	8

e Stop e

7. Data for Polynomial Analysis:

Skip this data group and the stop-e if there is no analysis.

	Data for independent variables: Start value	Increment	Permissible error	Variable no.
(0-2)	.....	.....	.....	1
(3-5)	.....	.....	.....	2
(6-8)	.....	.....	.....	3
(9-11)	.....	.....	.....	4
(12-14)	.....	.....	.....	5
(15-17)	.....	.....	.....	6
(18-20)	.....	.....	.....	7
(21-23)	.....	.....	.....	8

e Stop e

z Final Stop

8.1.4. Inputspecifikationer til PA-8

0. Cover Data:

Ff
..... Calculation No.
[..... File No.

	Cover	Page	Text
[	-	-	-
[	-	-	-

e Stop e

1. Headline Data:

	Section	Headline	Text
[	-	-	-
[	-	-	-

e Stop e

2. Calculation Parameters:

(0) ..... Number of functions (F)
(1) ..... Number of normal points (obs)
(2) ..... Number of special points (spec)
(3) ..... Minimum value of polynomial degree
(4) ..... Maximum value of polynomial degree
(5) ..... Use specified weights, 0: no, 1: yes
(6) ..... Use automatic weighing, 0: no, 1: yes
(7) ..... Print extended tables, 0: no, 1: yes
(8) ..... Polynomial analysis type (1)
(9) ..... Correction type for x-values (2)
(10) ..... Correction type for y-values (2)
(11) ..... Specified value of first function

Data for polynomial analysis:

(12) ..... Start value of x
(13) ..... Increment in x
(14) ..... Permissible error in x

e Stop e

(1) Polynomial analysis type:

0: No analysis
1: Find maximum of first function
2: Find specified value of first function

(2) Correction types:

Correction type	Normalise (for x only)	Take natural logarithm	Take reciprocal
0	no	no	no
1	yes	no	no
2	no	yes	no
3	yes	yes	no
4	no	no	yes
5	yes	no	yes
6	no	yes	yes
7	yes	yes	yes

3. Data for Special Points:

Skip this data group and the stop-e if there are no special points.

	x	y	Type (1)
(0-2)	.....	.....	.....
(3-5)	.....	.....	.....
(6-8)	.....	.....	.....
(9-11)	.....	.....	.....
(12-14)	.....	.....	.....
(15-17)	.....	.....	.....
(18-20)	.....	.....	.....
(21-23)	.....	.....	.....
(24-26)	.....	.....	.....

e Stop e

(1) Type:

0: y is the function itself
1: y - dy/dx
2: y - d2y/dx2
etc.

4. Data for Normal Points:

Each line contains one x-value and as many y-values as there are functions. All y-values in a line refer to the same x-value.

x y1 y2 y3 .....

....
....
....
....
....
....
....

e Stop e

5. Weights:

Skip this data group and the stop-e if there is no specified weighing.

....
....
....
....

e Stop e

6. Data for Extended Tables:

Skip this data group and the stop-e if there is no printing of extended tables.

(0) ..... Number of groups
(1) ..... Print first-order derivative (0: no, 1: yes)
(2) ..... Print second-order derivative (0: no, 1: yes)

Group data:

	Number of points	Start value	Increment
(3-5)	.....	.....	.....
(6-8)	.....	.....	.....
(9-11)	.....	.....	.....
(12-14)	.....	.....	.....
(15-17)	.....	.....	.....
(18-20)	.....	.....	.....

e Stop e

z Final stop

8.2. Typiske Beregninger.

8.2.1. Typisk beregning fra LINEAR EQUATIONS-4

August 28, 1964

File No. 358

Calculation Example with 10 Equations

GIER Calculation No. 7931

Solution of Linear Equations and Matrix Inversion with Full Pivoting

GIER Program LINEAR EQUATIONS-4

SECTION 1

ORIGINAL MATRIX

1.99670000	6.60280000	9.28070000	8.11669999	7.23060000
4.83180000	3.01849999	3.63860000	9.44109997	1.30290000
8.92029998	2.46529999	7.50309999	1.64130000	7.86119999
1.22160000	9.29169998	2.47189999	7.71619999	5.53269999
4.68179999	9.41959998	7.36029999	3.02339999	1.54850000
1.10960000	5.66839999	8.31879997	7.76679999	1.16670000
1.51550000	9.04389998	4.21159999	6.21599999	2.77680000
1.82370000	7.75799999	3.88250000	8.32709998	1.28320000
8.28089997	8.79119998	8.04139999	4.04730000	4.65539999
8.56709999	9.79479998	6.74120000	4.75649999	8.83519998
3.46230000	5.98969999	7.45909999	6.22039999	5.91890000
7.41790000	2.63829999	6.34789999	7.03950000	7.65079999
7.16909999	5.84249999	5.69909999	3.31340000	1.92640000
1.32930000	9.77289999	9.03070000	3.78529999	7.43329999
9.13569999	3.42770000	2.62080000	2.77949999	2.66249999
1.12940000	6.30779999	1.99290000	7.22839999	7.76069999
4.72129999	1.69850000	3.70840000	1.94670000	5.73649999
7.52119999	8.97739998	7.31250000	8.22819999	7.08620000
3.91060000	2.48289999	2.07139999	6.01830000	5.38270000
2.08300000	4.14129999	9.94259998	3.26899999	2.73779999

INVERSE MATRIX

$1.03533961 \times 10^{-1}$	$-6.00653441 \times 10^{-2}$	$5.19040971 \times 10^{-2}$	$-1.29472740 \times 10^{-1}$	$1.08234609 \times 10^{-1}$
$-1.70290944 \times 10^{-1}$	$-6.94285622 \times 10^{-2}$	$1.41161563 \times 10^{-1}$	$-1.54627289 \times 10^{-2}$	$6.56520927 \times 10^{-2}$
$-2.52734920 \times 10^{-1}$	$1.10676402 \times 10^{-1}$	$7.20544839 \times 10^{-2}$	$1.59765346 \times 10^{-1}$	$5.37563493 \times 10^{-2}$
$1.81770193 \times 10^{-1}$	$-1.71746983 \times 10^{-1}$	$-5.67300755 \times 10^{-2}$	$-1.18988724 \times 10^{-1}$	$4.46805703 \times 10^{-2}$
$1.85703582 \times 10^{-1}$	$-2.53008457 \times 10^{-2}$	$-6.42246718 \times 10^{-3}$	$-1.26254088 \times 10^{-1}$	$-2.54423412 \times 10^{-2}$
$-6.12874683 \times 10^{-2}$	$1.76139846 \times 10^{-1}$	$-4.22175047 \times 10^{-2}$	$1.16684386 \times 10^{-2}$	$-1.10720421 \times 10^{-1}$
$2.42558018 \times 10^{-1}$	$-1.50351189 \times 10^{-1}$	$-1.26800572 \times 10^{-1}$	$-4.34164392 \times 10^{-2}$	$1.19746072 \times 10^{-2}$
$-1.57713824 \times 10^{-1}$	$1.37859113 \times 10^{-1}$	$9.98111335 \times 10^{-2}$	$-6.12979534 \times 10^{-3}$	$2.35199033 \times 10^{-2}$
$-2.63946226 \times 10^{-1}$	$2.37708051 \times 10^{-1}$	$2.49932776 \times 10^{-3}$	$1.43174351 \times 10^{-1}$	$6.64316118 \times 10^{-4}$
$2.16296837 \times 10^{-1}$	$-1.97230904 \times 10^{-1}$	$-1.27271794 \times 10^{-1}$	$-9.65331967 \times 10^{-2}$	$1.16599428 \times 10^{-1}$
$1.15542953 \times 10^{-1}$	$-1.13948032 \times 10^{-1}$	$7.11270433 \times 10^{-4}$	$-8.00486081 \times 10^{-2}$	$1.23172237 \times 10^{-1}$
$-1.30103447 \times 10^{-1}$	$-5.75063851 \times 10^{-2}$	$3.72698698 \times 10^{-2}$	$9.06995556 \times 10^{-2}$	$-5.68386528 \times 10^{-3}$
$5.89487182 \times 10^{-2}$	$-6.54200232 \times 10^{-3}$	$-7.66884629 \times 10^{-2}$	$4.83166671 \times 10^{-2}$	$2.05153653 \times 10^{-2}$
$-1.18882471 \times 10^{-1}$	$9.70502063 \times 10^{-2}$	$-3.89182691 \times 10^{-2}$	$5.45349115 \times 10^{-2}$	$-3.31155735 \times 10^{-2}$
$-3.67762842 \times 10^{-2}$	$-1.51479180 \times 10^{-2}$	$6.61704862 \times 10^{-2}$	$-3.70720963 \times 10^{-2}$	$-3.89783899 \times 10^{-2}$
$2.54070482 \times 10^{-2}$	$1.88406479 \times 10^{-2}$	$-3.31904168 \times 10^{-2}$	$3.14909058 \times 10^{-2}$	$5.35006041 \times 10^{-2}$
$2.89497306 \times 10^{-2}$	$-3.47714556 \times 10^{-2}$	$5.83249941 \times 10^{-2}$	$-3.22499027 \times 10^{-3}$	$-8.85443853 \times 10^{-2}$
$2.10163324 \times 10^{-3}$	$-2.33411330 \times 10^{-2}$	$7.22737880 \times 10^{-2}$	$8.39823561 \times 10^{-2}$	$-4.58503877 \times 10^{-2}$
$-1.51530219 \times 10^{-1}$	$6.08416975 \times 10^{-2}$	$-5.52884466 \times 10^{-2}$	$7.58159608 \times 10^{-2}$	$-8.91975891 \times 10^{-2}$
$2.30233255 \times 10^{-1}$	$6.27832739 \times 10^{-2}$	$-2.26730694 \times 10^{-2}$	$-4.17037676 \times 10^{-2}$	$-5.13437631 \times 10^{-2}$

SOLUTION OF LINEAR EQUATIONS

INPUT VECTOR		SOLUTION VECTOR	
5.54603268_{10}	1	$9.99950171_{10^{-1}}$	
5.46253660_{10}	1	1.00001651	
5.00638425_{10}	1	1.00002691	
4.68382567_{10}	1	$9.99968998_{10^{-1}}$	
7.25111082_{10}	1	1.00002033	
6.01448972_{10}	1	$9.99976475_{10^{-1}}$	
5.53021239_{10}	1	1.00000940	
4.50452880_{10}	1	$9.99994174_{10^{-1}}$	
5.69368895_{10}	1	$9.99978781_{10^{-1}}$	
4.20394286_{10}	1	1.00004740	

8.2.2. Typisk beregning fra PA-6

July 22, 1964

File No. 358

Calculation Example

GIER Calculation No. 7509

Linear Approximation to a Function of Several Variables

GIER Program PA-6

SECTION 1

Approximation to Ammonia Equilibrium
x1: Temperature (deg.C), x2: Pressure (atm.abs.)

DATA MATERIAL

Obs. No.	Ident. No.	y(obs)	y(calc)	x1	x2
1	1	43.920	43.715	410.00	280.00
2	2	38.180	38.332	440.00	300.00
3	3	33.200	33.374	440.00	240.00
4	4	30.570	30.337	460.00	260.00
5	5	39.680	39.717	420.00	260.00
6	6	38.950	39.025	430.00	280.00

Mean error: 228.76 $_{10}$-3

CALCULATED COEFFICIENTS

Expression for y:

y := y0 + x1×c1 + x2×c2;

Variable No.	Coefficient	Coefficient Error
1	-2.34508 $_{10}$-1	5.98566 $_{10}$-3
2	8.26349 $_{10}$-2	4.99711 $_{10}$-3

y0: 1.16726 $_{10}$ 2

SECTION 2

Example from: Lapidus: Digital Computation
for Chemical Engineers, pag. 354

LIST OF CORRECTIONS

The following variables are internally converted into natural logarithms:

y x1 x2 x3 x4

DATA MATERIAL

Obs. No.	Ident. No.	y(obs)	y(calc)	x1	x2	x3
1	1	469.00	476.86	636.00 $_{10}$-3 256.00	309.00	833.00
2	2	913.00	876.38	636.00 $_{10}$-3 555.00	309.00	868.00
3	3	1.1200 $_{10}$ 3	1.1590 $_{10}$ 3	641.00 $_{10}$-3 786.00	309.00	800.00
4	4	234.00	262.43	285.00 $_{10}$-3 255.00	309.00	800.00
5	5	487.00	483.81	285.00 $_{10}$-3 555.00	309.00	800.00
6	6	709.00	673.02	283.00 $_{10}$-3 850.00	309.00	767.00
7	7	581.00	535.45	518.00 $_{10}$-3 254.00	683.00	795.00
8	8	650.00	612.96	521.00 $_{10}$-3 300.00	683.00	795.00
9	9	885.00	831.96	524.00 $_{10}$-3 440.00	683.00	795.00
10	10	672.00	697.57	455.00 $_{10}$-3 338.00	1.0120 $_{10}$ 3	867.00
11	11	986.00	1.0381 $_{10}$ 3	451.00 $_{10}$-3 565.00	1.0120 $_{10}$ 3	867.00
12	12	1.3100 $_{10}$ 3	1.3885 $_{10}$ 3	455.00 $_{10}$-3 811.00	1.0120 $_{10}$ 3	867.00
13	13	1.1900 $_{10}$ 3	1.2578 $_{10}$ 3	944.00 $_{10}$-3 343.00	1.1300 $_{10}$ 3	1.6080 $_{10}$ 3
14	14	1.8900 $_{10}$ 3	1.9273 $_{10}$ 3	974.00 $_{10}$-3 573.00	1.1300 $_{10}$ 3	1.6080 $_{10}$ 3
15	15	2.4600 $_{10}$ 3	2.5614 $_{10}$ 3	985.00 $_{10}$-3 814.00	1.1300 $_{10}$ 3	1.6080 $_{10}$ 3
16	16	915.00	901.53	602.00 $_{10}$-3 343.00	1.1300 $_{10}$ 3	1.6730 $_{10}$ 3
17	17	1.2600 $_{10}$ 3	1.1839 $_{10}$ 3	602.00 $_{10}$-3 485.00	1.1300 $_{10}$ 3	1.6730 $_{10}$ 3
18	18	1.6900 $_{10}$ 3	1.6090 $_{10}$ 3	617.00 $_{10}$-3 700.00	1.1300 $_{10}$ 3	1.6730 $_{10}$ 3

Mean error: 61.792

CALCULATED COEFFICIENTS

Expression for y:

y := exp(y0 + ln(x1)×c1 + ln(x2)×c2 + ln(x3)×c3 + ln(x4)×c4);

Variable No.	Coefficient	Coefficient Error
1	7.40198 $_{10}$-1	5.46501 $_{10}$-2
2	3.45383 $_{10}$-1	3.72990 $_{10}$-2
3	-5.70411 $_{10}$-4	7.44151 $_{10}$-2
4	7.86533 $_{10}$-1	3.62497 $_{10}$-2

y0: 1.64374 $_{10}$-1

8.2.3. Typisk beregning fra PA-7

December 2, 1964

File No. 358

Calculation Example

GIER Calculation No. 9041

Polynomial Approximation to One or More Functions of Several Variables

GIER Program PA-7

SECTION 1

Approximation to Ammonia Equilibrium Data

GENERAL DATA

Number of variables	3
Number of functions	1
Print polynomial coefficients, 0: no, 1: yes	1
Print original z-table, 0: no, 1: yes	1
Maximum sum of polynomial degrees	6
Calculate extended table, 0: no, 1: yes	1

DATA FOR VARIABLES AND DEGREES

Variable no.	Number of values	Maximum polynomial degree	Correction type	Normalize	Take nat. log.	Take reciprocal
1	5	3	1	yes	no	no
2	6	3	1	yes	no	no
3	3	2	1	yes	no	no

TABLES OF ORIGINAL FUNCTION VALUES

PART 1

x3 0.00000

INPUT z, CALCULATED z, AND ERRORS

x2 \ x1	2.00000 $_{10}$ 2	2.20000 $_{10}$ 2	2.40000 $_{10}$ 2	2.60000 $_{10}$ 2	2.80000 $_{10}$ 2
4.00000 $_{10}$ 2	3.88210 $_{10}$ 1	4.09274 $_{10}$ 1	4.29013 $_{10}$ 1	4.47590 $_{10}$ 1	4.65139 $_{10}$ 1
	3.88211 $_{10}$ 1	4.09273 $_{10}$ 1	4.29016 $_{10}$ 1	4.47589 $_{10}$ 1	4.65140 $_{10}$ 1
	9.85861 $_{10}$-5	-1.24335 $_{10}$-4	2.99096 $_{10}$-4	-1.07169 $_{10}$-4	8.05855 $_{10}$-5
4.08000 $_{10}$ 2	3.68153 $_{10}$ 1	3.88940 $_{10}$ 1	4.08475 $_{10}$ 1	4.26906 $_{10}$ 1	4.44357 $_{10}$ 1
	3.68152 $_{10}$ 1	3.88937 $_{10}$ 1	4.08475 $_{10}$ 1	4.26903 $_{10}$ 1	4.44356 $_{10}$ 1
	-1.27673 $_{10}$-4	-2.77042 $_{10}$-4	4.33922 $_{10}$-5	-3.11494 $_{10}$-4	-8.67844 $_{10}$-5
4.16000 $_{10}$ 2	3.48732 $_{10}$ 1	3.69184 $_{10}$ 1	3.88457 $_{10}$ 1	4.06688 $_{10}$ 1	4.23990 $_{10}$ 1
	3.48733 $_{10}$ 1	3.69184 $_{10}$ 1	3.88460 $_{10}$ 1	4.06688 $_{10}$ 1	4.23991 $_{10}$ 1
	1.10507 $_{10}$-4	-3.25441 $_{10}$-5	3.25561 $_{10}$-4	-2.63453 $_{10}$-5	1.01447 $_{10}$-4
4.24000 $_{10}$ 2	3.29982 $_{10}$ 1	3.50047 $_{10}$ 1	3.69007 $_{10}$ 1	3.86987 $_{10}$ 1	4.04090 $_{10}$ 1
	3.29984 $_{10}$ 1	3.50047 $_{10}$ 1	3.69010 $_{10}$ 1	3.86987 $_{10}$ 1	4.04091 $_{10}$ 1
	1.79291 $_{10}$-4	-4.22001 $_{10}$-5	2.72632 $_{10}$-4	-2.12193 $_{10}$-5	1.31369 $_{10}$-4
4.32000 $_{10}$ 2	3.11934 $_{10}$ 1	3.31563 $_{10}$ 1	3.50163 $_{10}$ 1	3.67846 $_{10}$ 1	3.84705 $_{10}$ 1
	3.11932 $_{10}$ 1	3.31560 $_{10}$ 1	3.50163 $_{10}$ 1	3.67843 $_{10}$ 1	3.84704 $_{10}$ 1
	-1.55091 $_{10}$-4	-2.57611 $_{10}$-4	1.16825 $_{10}$-5	-2.66433 $_{10}$-4	-1.10984 $_{10}$-4

4.40000 $_{10}$ 2	2.94607 $_{10}$ 1	3.13760 $_{10}$ 1	3.31958 $_{10}$ 1	3.49301 $_{10}$ 1	3.65875 $_{10}$ 1
	2.94608 $_{10}$ 1	3.13760 $_{10}$ 1	3.31960 $_{10}$ 1	3.49301 $_{10}$ 1	3.65876 $_{10}$ 1
	7.34925 $_{10}$-5	-3.02792 $_{10}$-5	1.69516 $_{10}$-4	-3.18289 $_{10}$-5	6.10352 $_{10}$-5

PART 2

x3 1.00000 $_{10}$ 1

INPUT z, CALCULATED z, AND ERRORS

x2 \ x1	2.00000 $_{10}$ 2	2.20000 $_{10}$ 2	2.40000 $_{10}$ 2	2.60000 $_{10}$ 2	2.80000 $_{10}$ 2
4.00000 $_{10}$ 2	3.16245 $_{10}$ 1	3.33405 $_{10}$ 1	3.49477 $_{10}$ 1	3.64593 $_{10}$ 1	3.78863 $_{10}$ 1
	3.16246 $_{10}$ 1	3.33404 $_{10}$ 1	3.49479 $_{10}$ 1	3.64592 $_{10}$ 1	3.78864 $_{10}$ 1
	5.95450 $_{10}$-5	-7.12872 $_{10}$-5	2.37346 $_{10}$-4	-9.31025 $_{10}$-5	5.88894 $_{10}$-5
4.08000 $_{10}$ 2	2.99900 $_{10}$ 1	3.16841 $_{10}$ 1	3.32752 $_{10}$ 1	3.47754 $_{10}$ 1	3.61950 $_{10}$ 1
	2.99899 $_{10}$ 1	3.16838 $_{10}$ 1	3.32752 $_{10}$ 1	3.47752 $_{10}$ 1	3.61949 $_{10}$ 1
	-7.51019 $_{10}$-5	-2.63214 $_{10}$-4	2.74181 $_{10}$-6	-2.17676 $_{10}$-4	-6.41346 $_{10}$-5
4.16000 $_{10}$ 2	2.84074 $_{10}$ 1	3.00746 $_{10}$ 1	3.16449 $_{10}$ 1	3.31294 $_{10}$ 1	3.45374 $_{10}$ 1
	2.84075 $_{10}$ 1	3.00746 $_{10}$ 1	3.16452 $_{10}$ 1	3.31294 $_{10}$ 1	3.45375 $_{10}$ 1
	9.90033 $_{10}$-5	-5.48363 $_{10}$-6	2.59757 $_{10}$-4	-1.12057 $_{10}$-5	7.61747 $_{10}$-5
4.24000 $_{10}$ 2	2.68795 $_{10}$ 1	2.85155 $_{10}$ 1	3.00608 $_{10}$ 1	3.15254 $_{10}$ 1	3.29177 $_{10}$ 1
	2.68796 $_{10}$ 1	2.85155 $_{10}$ 1	3.00610 $_{10}$ 1	3.15254 $_{10}$ 1	3.29178 $_{10}$ 1
	9.03010 $_{10}$-5	2.20537 $_{10}$-6	2.05278 $_{10}$-4	-4.73857 $_{10}$-5	9.72748 $_{10}$-5
4.32000 $_{10}$ 2	2.54086 $_{10}$ 1	2.70096 $_{10}$ 1	2.85259 $_{10}$ 1	2.99668 $_{10}$ 1	3.13398 $_{10}$ 1
	2.54085 $_{10}$ 1	2.70094 $_{10}$ 1	2.85259 $_{10}$ 1	2.99666 $_{10}$ 1	3.13397 $_{10}$ 1
	-9.26852 $_{10}$-5	-2.40326 $_{10}$-4	3.58224 $_{10}$-5	-2.00033 $_{10}$-4	-8.29697 $_{10}$-5
4.40000 $_{10}$ 2	2.39965 $_{10}$ 1	2.55590 $_{10}$ 1	2.70430 $_{10}$ 1	2.84567 $_{10}$ 1	2.98070 $_{10}$ 1
	2.39966 $_{10}$ 1	2.55590 $_{10}$ 1	2.70431 $_{10}$ 1	2.84567 $_{10}$ 1	2.98071 $_{10}$ 1
	5.84722 $_{10}$-5	-3.27229 $_{10}$-5	1.47879 $_{10}$-4	-4.25577 $_{10}$-5	5.32269 $_{10}$-5

PART 3

x3 2.00000 $_{10}$ 1

INPUT z, CALCULATED z, AND ERRORS

x2 \ x1	2.00000 $_{10}$ 2	2.20000 $_{10}$ 2	2.40000 $_{10}$ 2	2.60000 $_{10}$ 2	2.80000 $_{10}$ 2
4.00000 $_{10}$ 2	2.54249 $_{10}$ 1	2.68176 $_{10}$ 1	2.81221 $_{10}$ 1	2.93491 $_{10}$ 1	3.05074 $_{10}$ 1
	2.54250 $_{10}$ 1	2.68175 $_{10}$ 1	2.81223 $_{10}$ 1	2.93490 $_{10}$ 1	3.05075 $_{10}$ 1
	6.09756 $_{10}$-5	-7.03931 $_{10}$-5	1.99080 $_{10}$-4	-7.78437 $_{10}$-5	5.09620 $_{10}$-5
4.08000 $_{10}$ 2	2.40996 $_{10}$ 1	2.54739 $_{10}$ 1	2.67648 $_{10}$ 1	2.79822 $_{10}$ 1	2.91341 $_{10}$ 1
	2.40995 $_{10}$ 1	2.54737 $_{10}$ 1	2.67648 $_{10}$ 1	2.79820 $_{10}$ 1	2.91341 $_{10}$ 1
	-8.51154 $_{10}$-5	-2.35856 $_{10}$-4	3.77893 $_{10}$-5	-2.16305 $_{10}$-4	-4.99487 $_{10}$-5

$4.16000_{10}2$	$2.28171_{10}1$	$2.41690_{10}1$	$2.54426_{10}1$	$2.66468_{10}1$	$2.77890_{10}1$
	$2.28172_{10}1$	$2.41690_{10}1$	$2.54428_{10}1$	$2.66468_{10}1$	$2.77891_{10}1$
	$1.03295_{10}-4$	$1.78814_{10}-6$	$2.22027_{10}-4$	$-3.99351_{10}-6$	$5.64456_{10}-5$
$4.24000_{10}2$	$2.15798_{10}1$	$2.29058_{10}1$	$2.41586_{10}1$	$2.53463_{10}1$	$2.64754_{10}1$
	$2.15799_{10}1$	$2.29058_{10}1$	$2.41588_{10}1$	$2.53463_{10}1$	$2.64755_{10}1$
	$6.30021_{10}-5$	$-2.55108_{10}-5$	$1.92344_{10}-4$	$-4.98295_{10}-5$	$8.15392_{10}-5$
$4.32000_{10}2$	$2.03894_{10}1$	$2.16864_{10}1$	$2.29153_{10}1$	$2.40833_{10}1$	$2.51964_{10}1$
	$2.03893_{10}1$	$2.16862_{10}1$	$2.29153_{10}1$	$2.40831_{10}1$	$2.51963_{10}1$
	$-6.93798_{10}-5$	$-1.86026_{10}-4$	$-1.07288_{10}-5$	$-1.63376_{10}-4$	$-6.30021_{10}-5$
$4.40000_{10}2$	$1.92474_{10}1$	$2.05126_{10}1$	$2.17147_{10}1$	$2.28603_{10}1$	$2.39546_{10}1$
	$1.92474_{10}1$	$2.05126_{10}1$	$2.17149_{10}1$	$2.28602_{10}1$	$2.39546_{10}1$
	$4.27961_{10}-5$	$-4.77433_{10}-5$	$1.53065_{10}-4$	$-5.33462_{10}-5$	$3.45707_{10}-5$

Mean error in z: $1.39065_{10}-4$

EXTENDED TABLES OF FUNCTION VALUES

PART 1

x3 5.00000

CALCULATED z-VALUES

x2 \ x1	$2.20000_{10}2$	$2.30000_{10}2$	$2.40000_{10}2$	$2.50000_{10}2$
$4.08000_{10}2$	$3.51638_{10}1$	$3.60596_{10}1$	$3.69286_{10}1$	$3.77724_{10}1$
$4.12000_{10}2$	$3.42648_{10}1$	$3.51547_{10}1$	$3.60186_{10}1$	$3.68579_{10}1$
$4.16000_{10}2$	$3.33792_{10}1$	$3.42626_{10}1$	$3.51208_{10}1$	$3.59551_{10}1$
$4.20000_{10}2$	$3.25075_{10}1$	$3.33839_{10}1$	$3.42357_{10}1$	$3.50644_{10}1$
$4.24000_{10}2$	$3.16502_{10}1$	$3.25188_{10}1$	$3.33638_{10}1$	$3.41863_{10}1$

PART 2

x3 $1.00000_{10}1$

CALCULATED z-VALUES

x2 \ x1	$2.20000_{10}2$	$2.30000_{10}2$	$2.40000_{10}2$	$2.50000_{10}2$
$4.08000_{10}2$	$3.16838_{10}1$	$3.24916_{10}1$	$3.32752_{10}1$	$3.40359_{10}1$
$4.12000_{10}2$	$3.08731_{10}1$	$3.16757_{10}1$	$3.24546_{10}1$	$3.32114_{10}1$
$4.16000_{10}2$	$3.00746_{10}1$	$3.08713_{10}1$	$3.16452_{10}1$	$3.23974_{10}1$
$4.20000_{10}2$	$2.92886_{10}1$	$3.00790_{10}1$	$3.08471_{10}1$	$3.15944_{10}1$
$4.24000_{10}2$	$2.85155_{10}1$	$2.92990_{10}1$	$3.00610_{10}1$	$3.08027_{10}1$

PART 3

x3 $1.50000_{10}1$

CALCULATED z-VALUES

x2 \ x1	$2.20000_{10}2$	$2.30000_{10}2$	$2.40000_{10}2$	$2.50000_{10}2$
$4.08000_{10}2$	$2.84538_{10}1$	$2.91815_{10}1$	$2.98873_{10}1$	$3.05725_{10}1$
$4.12000_{10}2$	$2.77236_{10}1$	$2.84465_{10}1$	$2.91482_{10}1$	$2.98298_{10}1$
$4.16000_{10}2$	$2.70045_{10}1$	$2.77221_{10}1$	$2.84192_{10}1$	$2.90967_{10}1$
$4.20000_{10}2$	$2.62968_{10}1$	$2.70086_{10}1$	$2.77005_{10}1$	$2.83736_{10}1$
$4.24000_{10}2$	$2.56007_{10}1$	$2.63064_{10}1$	$2.69927_{10}1$	$2.76607_{10}1$

PART 4

x3 $2.00000_{10}1$

CALCULATED z-VALUES

x2 \ x1	$2.20000_{10}2$	$2.30000_{10}2$	$2.40000_{10}2$	$2.50000_{10}2$
$4.08000_{10}2$	$2.54737_{10}1$	$2.61291_{10}1$	$2.67648_{10}1$	$2.73821_{10}1$
$4.12000_{10}2$	$2.48163_{10}1$	$2.54673_{10}1$	$2.60992_{10}1$	$2.67132_{10}1$
$4.16000_{10}2$	$2.41690_{10}1$	$2.48152_{10}1$	$2.54428_{10}1$	$2.60530_{10}1$
$4.20000_{10}2$	$2.35321_{10}1$	$2.41729_{10}1$	$2.47959_{10}1$	$2.54019_{10}1$
$4.24000_{10}2$	$2.29058_{10}1$	$2.35409_{10}1$	$2.41588_{10}1$	$2.47603_{10}1$

REVERSED POLYNOMIAL COEFFICIENTS

2 4 2 4

$-4.79872584_{10}-13$ $1.20761495_{10}-10$

$-1.00719687_{10}-12$ $-2.34666469_{10}-10$ $1.05912334_{10}-7$

$-6.53439481_{10}-13$ $1.76876147_{10}-10$ $-5.93619375_{10}-8$ $-3.79854184_{10}-5$

$2.16390823_{10}-11$ $-1.22989546_{10}-8$ $1.45830891_{10}-5$ $4.83901185_{10}-3$

2 4 3 4

$3.50978851_{10}-13$ $-1.31238299_{10}-10$ $-2.25678499_{10}-8$

$-1.17932048_{10}-13$ $-1.09934078_{10}-11$ $3.19948560_{10}-8$ $-7.48910993_{10}-6$

$4.36891829_{10}-11$ $-1.57484792_{10}-8$ $2.96378538_{10}-6$ $4.13175642_{10}-3$

$-4.36179375_{10}-9$ $2.12868351_{10}-6$ $-1.57761480_{10}-3$ $-6.53514765_{10}-1$

3 4 4 4

$1.92354096_{10}-13$ $-2.71649440_{10}-11$ $5.83326158_{10}-9$ $1.04055343_{10}-6$

$6.27829685_{10}-12$ $9.46976404_{10}-10$ $-1.57075355_{10}-6$ $3.70989791_{10}-4$

$-2.45422987_{10}-9$ $8.09867513_{10}-7$ $-1.39562318_{10}-4$ $-1.98035958_{10}-1$

$2.02475256_{10}-7$ $-1.04697549_{10}-4$ $7.57521614_{10}-2$ $3.08471467_{10}1$

NORMALIZING CORRECTIONS

$-2.40000000_{10}2$ $-4.20000000_{10}2$ $-1.00000000_{10}1$

SECTION 2

Analysis of Data from Calculation 5376
Function 1: Ammonia Production
Function 2: Exchanger Height

GENERAL DATA

Number of variables	2
Number of functions	2
Maximum sum of polynomial degrees	10
Calculate extended table, 0: no, 1: yes	0
Polynomial function analysis type	2
Find maximum of first function with spec. values of other functions	
Maximum number of iterations	100
Minimum pivot in inversions	$1.000_{10}-8$
Maximum step factor	4.0000^{10}
Desired value of function no. 1	0.0000
Desired value of function no. 2	2.0000

DATA FOR VARIABLES AND DEGREES

Variable no.	Number of values	Maximum polynomial degree	Correction type	Normalize	Take nat. log.	Take reciprocal
1	6	4	1	yes	no	no
2	8	6	1	yes	no	no

TABLES OF ORIGINAL FUNCTION VALUES

Function no. 1

INPUT z, CALCULATED z, AND ERRORS

x2 \ x1	$4.00000_{10}2$	$4.10000_{10}2$	$4.20000_{10}2$	$4.30000_{10}2$	$4.40000_{10}2$
$6.60000_{10}1$	$6.74900_{10}1$	$6.88000_{10}1$	$7.00000_{10}1$	$7.11200_{10}1$	$7.18800_{10}1$
	$6.74932_{10}1$	$6.87864_{10}1$	$7.00262_{10}1$	$7.10941_{10}1$	$7.18930_{10}1$
	$3.17073_{10}-3$	$-1.35641_{10}-2$	$2.61910_{10}-2$	$-2.59178_{10}-2$	$1.30453_{10}-2$
$6.80000_{10}1$	$6.86900_{10}1$	$7.00300_{10}1$	$7.12700_{10}1$	$7.20600_{10}1$	$7.25500_{10}1$
	$6.86837_{10}1$	$7.00454_{10}1$	$7.12458_{10}1$	$7.20823_{10}1$	$7.25382_{10}1$
	$-6.30069_{10}-3$	$1.53573_{10}-2$	$-2.42379_{10}-2$	$2.22740_{10}-2$	$-1.17726_{10}-2$
$7.00000_{10}1$	$6.98800_{10}1$	$7.12500_{10}1$	$7.21000_{10}1$	$7.26400_{10}1$	$7.28500_{10}1$
	$6.98926_{10}1$	$7.12355_{10}1$	$7.21096_{10}1$	$7.26363_{10}1$	$7.28538_{10}1$
	$1.25837_{10}-2$	$-1.45376_{10}-2$	$9.60469_{10}-3$	$-3.73840_{10}-3$	$3.76129_{10}-3$
$7.20000_{10}1$	$7.10300_{10}1$	$7.20200_{10}1$	$7.26400_{10}1$	$7.28500_{10}1$	$7.27200_{10}1$
	$7.10102_{10}1$	$7.20385_{10}1$	$7.26355_{10}1$	$7.28447_{10}1$	$7.27195_{10}1$
	$-1.98293_{10}-2$	$1.84650_{10}-2$	$-4.50969_{10}-3$	$-5.28622_{10}-3$	$-5.21660_{10}-4$

$7.40000_{10}1$	$7.18100_{10}1$	$7.25300_{10}1$	$7.28100_{10}1$	$7.26800_{10}1$	$7.22800_{10}1$
	$7.18285_{10}1$	$7.25181_{10}1$	$7.28014_{10}1$	$7.26984_{10}1$	$7.22740_{10}1$
	$1.85151_{10}-2$	$-1.19247_{10}-2$	$-8.59261_{10}-3$	$1.83754_{10}-2$	$-6.03080_{10}-3$
$7.60000_{10}1$	$7.23200_{10}1$	$7.27000_{10}1$	$7.26300_{10}1$	$7.22500_{10}1$	$7.15900_{10}1$
	$7.23088_{10}1$	$7.27074_{10}1$	$7.26347_{10}1$	$7.22394_{10}1$	$7.15934_{10}1$
	$-1.11620_{10}-2$	$7.38549_{10}-3$	$4.67157_{10}-3$	$-1.05560_{10}-2$	$3.37434_{10}-3$
$7.80000_{10}1$	$7.25400_{10}1$	$7.25500_{10}1$	$7.22000_{10}1$	$7.15600_{10}1$	$7.06400_{10}1$
	$7.25445_{10}1$	$7.25438_{10}1$	$7.22060_{10}1$	$7.15560_{10}1$	$7.06426_{10}1$
	$4.47083_{10}-3$	$-6.23727_{10}-3$	$5.97763_{10}-3$	$-4.02570_{10}-3$	$2.64096_{10}-3$
$8.00000_{10}1$	$7.24200_{10}1$	$7.21500_{10}1$	$7.15300_{10}1$	$7.06200_{10}1$	$6.95000_{10}1$
	$7.24193_{10}1$	$7.21513_{10}1$	$7.15284_{10}1$	$7.06213_{10}1$	$6.94992_{10}1$
	$-7.19070_{10}-4$	$1.25194_{10}-3$	$-1.59955_{10}-3$	$1.30439_{10}-3$	$-7.52449_{10}-4$

x2	x1
	$4.50000_{10}2$
$6.60000_{10}1$	$7.23500_{10}1$
	$7.23473_{10}1$
	$-2.69556_{10}-3$
$6.80000_{10}1$	$7.27800_{10}1$
	$7.27829_{10}1$
	$2.93064_{10}-3$
$7.00000_{10}1$	$7.27200_{10}1$
	$7.27175_{10}1$
	$-2.49505_{10}-3$
$7.20000_{10}1$	$7.23200_{10}1$
	$7.23230_{10}1$
	$2.99788_{10}-3$
$7.40000_{10}1$	$7.16400_{10}1$
	$7.16383_{10}1$
	$-1.69444_{10}-3$
$7.60000_{10}1$	$7.06900_{10}1$
	$7.06911_{10}1$
	$1.05572_{10}-3$
$7.80000_{10}1$	$6.95400_{10}1$
	$6.95389_{10}1$
	$-1.10960_{10}-3$
$8.00000_{10}1$	$6.82300_{10}1$
	$6.82302_{10}1$
	$2.20776_{10}-4$

Mean error in z: $1.13349_{10}-2$

Function no. 2

INPUT z, CALCULATED z, AND ERRORS

x2	x1 = 4.00000 $_{10}$ 2	x1 = 4.10000 $_{10}$ 2	x1 = 4.20000 $_{10}$ 2	x1 = 4.30000 $_{10}$ 2	x1 = 4.40000 $_{10}$ 2
6.60000 $_{10}$ 1	1.92000	2.13000	2.41000	2.81000	2.96000
	1.92465	2.10799	2.45359	2.76636	2.98173
	4.65074 $_{10}$-3	-2.20128 $_{10}$-2	4.35882 $_{10}$-2	-4.36408 $_{10}$-2	2.17271 $_{10}$-2
6.80000 $_{10}$ 1	1.92000	2.12000	2.38000	2.48000	2.61000
	1.91570	2.13282	2.35742	2.50295	2.59918
	-4.30161 $_{10}$-3	1.28189 $_{10}$-2	-2.25786 $_{10}$-2	2.29455 $_{10}$-2	-1.08202 $_{10}$-2
7.00000 $_{10}$ 1	1.93000	2.12000	2.18000	2.28000	2.41000
	1.93679	2.11210	2.18662	2.27227	2.41190
	6.79347 $_{10}$-3	-7.90145 $_{10}$-3	6.62422 $_{10}$-3	-7.72585 $_{10}$-3	1.90475 $_{10}$-3
7.20000 $_{10}$ 1	1.93000	1.98000	2.05000	2.15000	2.29000
	1.91992	1.98695	2.05139	2.15044	2.29304
	-1.00793 $_{10}$-2	6.95186 $_{10}$-3	1.39318 $_{10}$-3	4.42296 $_{10}$-4	3.04214 $_{10}$-3
7.40000 $_{10}$ 1	1.83000	1.88000	1.97000	2.08000	2.22000
	1.83984	1.87424	1.96623	2.08194	2.21577
	9.84101 $_{10}$-3	-5.76161 $_{10}$-3	-3.77437 $_{10}$-3	1.93845 $_{10}$-3	-4.23279 $_{10}$-3
7.60000 $_{10}$ 1	1.75000	1.82000	1.91000	2.03000	2.17000
	1.74420	1.82294	1.91330	2.02780	2.17306
	-5.80163 $_{10}$-3	2.94086 $_{10}$-3	3.29611 $_{10}$-3	-2.19505 $_{10}$-3	3.05533 $_{10}$-3
7.80000 $_{10}$ 1	1.70000	1.78000	1.88000	2.00000	2.15000
	1.70192	1.77909	1.87877	2.00086	2.14892
	1.92048 $_{10}$-3	-9.14343 $_{10}$-4	-1.23117 $_{10}$-3	8.63783 $_{10}$-4	-1.08476 $_{10}$-3
8.00000 $_{10}$ 1	1.67000	1.76000	1.86000	1.99000	2.14000
	1.66991	1.75919	1.86205	1.98801	2.14109
	-8.75294 $_{10}$-5	-8.05013 $_{10}$-4	2.04633 $_{10}$-3	-1.99443 $_{10}$-3	1.09005 $_{10}$-3

x2	x1 = 4.50000 $_{10}$ 2
6.60000 $_{10}$ 1	3.17000
	3.16564
	-4.35952 $_{10}$-3
6.80000 $_{10}$ 1	2.79000
	2.79226
	2.26185 $_{10}$-3
7.00000 $_{10}$ 1	2.58000
	2.57933
	-6.74807 $_{10}$-4

7.20000 $_{10}$ 1 2.46000
2.45988
-1.19016 $_{10}$-4

7.40000 $_{10}$ 1 2.39000
2.39036
3.56883 $_{10}$-4

7.60000 $_{10}$ 1 2.35000
2.34968
-3.17618 $_{10}$-4

7.80000 $_{10}$ 1 2.33000
2.33012
1.18904 $_{10}$-4

8.00000 $_{10}$ 1 2.32000
2.31980
-2.04585 $_{10}$-4

Mean error in z: 1.21334 $_{10}$-2

REVERSED POLYNOMIAL COEFFICIENTS

Function no. 1

2 7 5 5

-1.24106108 $_{10}$-10 3.78753232 $_{10}$-10 7.82273479 $_{10}$-8 -1.39502949 $_{10}$-7 -4.59112057 $_{10}$-6
1.75068175 $_{10}$-10 6.64902182 $_{10}$-10 -1.10372704 $_{10}$-7 -7.55024243 $_{10}$-9 1.14692890 $_{10}$-6
8.68402933 $_{10}$-9 -2.65178107 $_{10}$-8 -5.41809638 $_{10}$-6 5.81344904 $_{10}$-6 3.48688620 $_{10}$-4
-1.04235041 $_{10}$-8 -1.22111606 $_{10}$-8 5.34066130 $_{10}$-6 3.12212539 $_{10}$-5 6.12119569 $_{10}$-4
-1.31536665 $_{10}$-7 1.43899835 $_{10}$-7 9.23098548 $_{10}$-5 1.14469350 $_{10}$-4 -4.66654458 $_{10}$-2
8.36920109 $_{10}$-8 9.82256779 $_{10}$-7 -5.43822742 $_{10}$-6 -1.56682745 $_{10}$-2 4.24073965 $_{10}$-3
2.37557741 $_{10}$-7 9.74320895 $_{10}$-6 -1.90311331 $_{10}$-3 4.94292188 $_{10}$-3 7.28366501 $_{10}$ 1

Function no. 2

2 7 5 5

-5.19929536 $_{10}$-11 3.63698740 $_{10}$-10 3.46996401 $_{10}$-8 -1.95448830 $_{10}$-7 -2.35410205 $_{10}$-6
1.48570914 $_{10}$-10 4.15478309 $_{10}$-10 -1.25283714 $_{10}$-7 -3.86861225 $_{10}$-8 1.23541210 $_{10}$-5
3.69149945 $_{10}$-9 -3.62392806 $_{10}$-8 -2.33346601 $_{10}$-6 2.00152602 $_{10}$-5 1.35240166 $_{10}$-4
-1.07247284 $_{10}$-8 2.15024861 $_{10}$-8 8.70536402 $_{10}$-6 -4.42799940 $_{10}$-5 -8.95571015 $_{10}$-4
-5.17149028 $_{10}$-8 8.09620147 $_{10}$-7 2.50598155 $_{10}$-5 -2.85644712 $_{10}$-4 3.43945356 $_{10}$-3
1.28303867 $_{10}$-7 -1.23463327 $_{10}$-6 -8.34082655 $_{10}$-5 9.07821868 $_{10}$-4 -3.57375066 $_{10}$-2
4.80753206 $_{10}$-8 -4.95970836 $_{10}$-7 1.27239012 $_{10}$-4 1.09969776 $_{10}$-2 2.05517755

NORMALIZING CORRECTIONS

-4.25000000 $_{10}$ 2 -7.30000000 $_{10}$ 1

ANALYSIS OF POLYNOMIAL FUNCTIONS

x: 4.200000 $_{10}$ 2 7.200000 $_{10}$ 1
y: 7.263549 $_{10}$ 1 2.051393
x: 4.220000 $_{10}$ 2 7.200000 $_{10}$ 1
y: 7.270679 $_{10}$ 1 2.067741
x: 4.200000 $_{10}$ 2 7.400000 $_{10}$ 1
y: 7.280141 $_{10}$ 1 1.966226
x: 4.180000 $_{10}$ 2 7.200000 $_{10}$ 1
y: 7.254864 $_{10}$ 1 2.036520
x: 4.200000 $_{10}$ 2 7.000000 $_{10}$ 1
y: 7.210960 $_{10}$ 1 2.186624
x: 4.220000 $_{10}$ 2 7.400000 $_{10}$ 1
y: 7.281028 $_{10}$ 1 1.988057

x: 4.219114 $_{10}$ 2 7.359432 $_{10}$ 1
y: 7.281805 $_{10}$ 1 2.000561
x: 4.239114 $_{10}$ 2 7.359432 $_{10}$ 1
y: 7.282533 $_{10}$ 1 2.022648
x: 4.219114 $_{10}$ 2 7.559432 $_{10}$ 1
y: 7.265024 $_{10}$ 1 1.942463
x: 4.199114 $_{10}$ 2 7.359432 $_{10}$ 1
y: 7.279514 $_{10}$ 1 1.979301
x: 4.219114 $_{10}$ 2 7.159432 $_{10}$ 1
y: 7.263738 $_{10}$ 1 2.088642
x: 4.239114 $_{10}$ 2 7.559432 $_{10}$ 1
y: 7.259427 $_{10}$ 1 1.964461

x: 4.221835 $_{10}$ 2 7.370160 $_{10}$ 1
y: 7.281881 $_{10}$ 1 1.999872
x: 4.241835 $_{10}$ 2 7.370160 $_{10}$ 1
y: 7.282059 $_{10}$ 1 2.022203
x: 4.221835 $_{10}$ 2 7.570160 $_{10}$ 1
y: 7.262554 $_{10}$ 1 1.942813
x: 4.201835 $_{10}$ 2 7.370160 $_{10}$ 1
y: 7.280154 $_{10}$ 1 1.978319
x: 4.221835 $_{10}$ 2 7.170160 $_{10}$ 1
y: 7.266627 $_{10}$ 1 2.084879
x: 4.241835 $_{10}$ 2 7.570160 $_{10}$ 1
y: 7.256472 $_{10}$ 1 1.964855
FOUND
x: 4.221763 $_{10}$ 2 7.370484 $_{10}$ 1
y: 7.281873 $_{10}$ 1 1.999683

e: 1.000000 1.000000 $_{10}$-1

8.2.4. Typisk beregning fra PA-8

December 10, 1964

File No. 358

Calculation Example

GIER Calculation No. 9099

Polynomial Approximation to One or More Functions of a Single Variable

GIER Program PA-8

SECTION 1

A-Function, Optimum Degree Calculated by Program

CALCULATION PARAMETERS

Number of functions	1
Number of normal points	8
Minimum value of polynomial degree	2
Maximum value of polynomial degree	7
Print extended tables, 0: no, 1: yes	1
Polynomial analysis type	2
Find specified value of first function	
Specified value of first function	$1.00000_{10}2$
Start value of x	$3.40000_{10}2$
Increment in x	$1.00000_{10}1$
Permissible error in x	1.00000

APPROXIMATION CALCULATIONS

Calculated Main Polynomial

Power	Coefficient
0	$-6.3296264_{10}5$
1	$9.0461316_{10}3$
2	$-5.1635584_{10}1$
3	$1.4709892_{10}-1$
4	$-2.0907634_{10}-4$
5	$1.1858974_{10}-7$

Fitting of Approximation Function

x	y, spec.	y, calc.	Error
$3.200000_{10}2$	$5.000000_{10}1$	$4.990039_{10}1$	$9.961_{10}-2$
$3.300000_{10}2$	$6.500000_{10}1$	$6.495703_{10}1$	$4.297_{10}-2$
$3.400000_{10}2$	$8.000000_{10}1$	$7.965430_{10}1$	$3.457_{10}-1$
$3.500000_{10}2$	$9.400000_{10}1$	$9.429492_{10}1$	$-2.949_{10}-1$
$3.600000_{10}2$	$1.080000_{10}2$	$1.074121_{10}2$	$5.879_{10}-1$
$3.700000_{10}2$	$1.170000_{10}2$	$1.171387_{10}2$	$-1.387_{10}-1$
$3.800000_{10}2$	$1.230000_{10}2$	$1.227422_{10}2$	$2.578_{10}-1$
$3.900000_{10}2$	$1.260000_{10}2$	$1.258555_{10}2$	$1.445_{10}-1$
Mean error			$3.095_{10}-1$

EXTENDED FUNCTION TABLES

Values of function:

x↓, dx->	0.000000	1.000000	2.000000	3.000000	4.000000
$3.200_{10}2$	$4.990039_{10}1$	$5.148633_{10}1$	$5.305078_{10}1$	$5.458008_{10}1$	$5.609766_{10}1$
$3.250_{10}2$	$5.759766_{10}1$	$5.908203_{10}1$	$6.057227_{10}1$	$6.202539_{10}1$	$6.350195_{10}1$
$3.300_{10}2$	$6.495703_{10}1$	$6.642578_{10}1$	$6.788477_{10}1$	$6.935352_{10}1$	$7.080859_{10}1$
$3.350_{10}2$	$7.227539_{10}1$	$7.373633_{10}1$	$7.522070_{10}1$	$7.670312_{10}1$	$7.816797_{10}1$
$3.400_{10}2$	$7.965430_{10}1$	$8.114258_{10}1$	$8.261719_{10}1$	$8.409766_{10}1$	$8.557617_{10}1$
$3.450_{10}2$	$8.703906_{10}1$	$8.850781_{10}1$	$8.999023_{10}1$	$9.142578_{10}1$	$9.285742_{10}1$

First-order derivative:

x↓, dx->	0.000000	1.000000	2.000000	3.000000	4.000000
$3.200_{10}2$	1.609833	1.577179	1.549622	1.526611	1.507843
$3.250_{10}2$	1.492889	1.481201	1.472504	1.466461	1.462646
$3.300_{10}2$	1.460693	1.460388	1.461304	1.463104	1.465790
$3.350_{10}2$	1.468658	1.471802	1.474731	1.477478	1.479614
$3.400_{10}2$	1.480957	1.481384	1.480774	1.478912	1.475586
$3.450_{10}2$	1.470764	1.464294	1.456055	1.445923	1.433929

POLYNOMIAL ANALYSIS

x	y
$3.400000_{10}2$	$7.965430_{10}1$
$3.500000_{10}2$	$9.429492_{10}1$
$3.538967_{10}2$	$9.968945_{10}1$
$3.638967_{10}2$	$1.116641_{10}2$

FOUND

$3.538967_{10}2$ $9.968945_{10}1$

SECTION 2

A-Function, Degree 4

CALCULATION PARAMETERS

Minimum value of polynomial degree	4
Maximum value of polynomial degree	4
Print extended tables, 0: no, 1: yes	0
Polynomial analysis type	0
No analysis	
Specified value of first function	0.00000
Start value of x	0.00000
Increment in x	0.00000
Permissible error in x	0.00000

APPROXIMATION CALCULATIONS

Calculated Main Polynomial

Power	Coefficient
0	$2.7291227_{10}4$
1	$-3.0032344_{10}2$
2	1.2201610
3	$-2.1660354_{10}-3$
4	$1.4204546_{10}-6$

Fitting of Approximation Function

x	y, spec.	y, calc.	Error
$3.200000_{10}2$	$5.000000_{10}1$	$5.010822_{10}1$	$-1.082_{10}-1$
$3.300000_{10}2$	$6.500000_{10}1$	$6.467731_{10}1$	$3.227_{10}-1$
$3.400000_{10}2$	$8.000000_{10}1$	$8.005658_{10}1$	$-5.658_{10}-2$
$3.500000_{10}2$	$9.400000_{10}1$	$9.467163_{10}1$	$-6.716_{10}-1$
$3.600000_{10}2$	$1.080000_{10}2$	$1.072856_{10}2$	$7.144_{10}-1$
$3.700000_{10}2$	$1.170000_{10}2$	$1.170065_{10}2$	$-6.531_{10}-3$
$3.800000_{10}2$	$1.230000_{10}2$	$1.232803_{10}2$	$-2.803_{10}-1$
$3.900000_{10}2$	$1.260000_{10}2$	$1.258950_{10}2$	$1.050_{10}-1$

Mean error $4.088_{10}-1$

SECTION 3

A-Function, Degree 5

CALCULATION PARAMETERS

Minimum value of polynomial degree	5
Maximum value of polynomial degree	5

APPROXIMATION CALCULATIONS

Calculated Main Polynomial

Power	Coefficient
0	$-6.3296264_{10}5$
1	$9.0461316_{10}3$
2	$-5.1635584_{10}1$
3	$1.4709892_{10}-1$
4	$-2.0907634_{10}-4$
5	$1.1858974_{10}-7$

Fitting of Approximation Function

x	y, spec.	y, calc.	Error
$3.200000_{10}2$	$5.000000_{10}1$	$4.990039_{10}1$	$9.961_{10}-2$
$3.300000_{10}2$	$6.500000_{10}1$	$6.495703_{10}1$	$4.297_{10}-2$
$3.400000_{10}2$	$8.000000_{10}1$	$7.965430_{10}1$	$3.457_{10}-1$
$3.500000_{10}2$	$9.400000_{10}1$	$9.429492_{10}1$	$-2.949_{10}-1$
$3.600000_{10}2$	$1.080000_{10}2$	$1.074121_{10}2$	$5.879_{10}-1$
$3.700000_{10}2$	$1.170000_{10}2$	$1.171387_{10}2$	$-1.387_{10}-1$
$3.800000_{10}2$	$1.230000_{10}2$	$1.227422_{10}2$	$2.578_{10}-1$
$3.900000_{10}2$	$1.260000_{10}2$	$1.258555_{10}2$	$1.445_{10}-1$

Mean error $3.095_{10}-1$

SECTION 4

A-Function, Degree 6

CALCULATION PARAMETERS

Minimum value of polynomial degree	6
Maximum value of polynomial degree	6

APPROXIMATION CALCULATIONS

Calculated Main Polynomial

Power	Coefficient
0	$-2.0028291_{10}6$
1	$3.2311716_{10}4$
2	$-2.1611602_{10}2$
3	$7.6666791_{10}-1$
4	$-1.5205660_{10}-3$
5	$1.5977561_{10}-6$
6	$-6.9444430_{10}-10$

Fitting of Approximation Function

x	y, spec.	y, calc.	Error
$3.200000_{10}2$	$5.000000_{10}1$	$4.969922_{10}1$	$3.008_{10}-1$
$3.300000_{10}2$	$6.500000_{10}1$	$6.475781_{10}1$	$2.422_{10}-1$
$3.400000_{10}2$	$8.000000_{10}1$	$7.936328_{10}1$	$6.367_{10}-1$
$3.500000_{10}2$	$9.400000_{10}1$	$9.401562_{10}1$	$-1.563_{10}-2$
$3.600000_{10}2$	$1.080000_{10}2$	$1.072187_{10}2$	$7.813_{10}-1$
$3.700000_{10}2$	$1.170000_{10}2$	$1.168555_{10}2$	$1.445_{10}-1$
$3.800000_{10}2$	$1.230000_{10}2$	$1.225156_{10}2$	$4.844_{10}-1$
$3.900000_{10}2$	$1.260000_{10}2$	$1.254727_{10}2$	$5.273_{10}-1$

Mean error $4.926_{10}-1$

SECTION 5

A-Function, ln(A) as a Function of x

CALCULATION PARAMETERS

Minimum value of polynomial degree	5
Maximum value of polynomial degree	5
Correction type for y-values	2

Correction type	Normalize	Take nat. log.	Take reciprocal
0	no	no	no
2	no	yes	no

APPROXIMATION CALCULATIONS

Calculated Main Polynomial

Power	Coefficient
0	$-6.2003961_{10}3$
1	$8.7480164_{10}1$
2	$-4.9370763_{10}-1$
3	$1.3932445_{10}-3$
4	$-1.9649091_{10}-6$
5	$1.1074678_{10}-9$

Fitting of Approximation Function

x	y, spec.	y, calc.	Error
$3.200000_{10}2$	$5.000000_{10}1$	$4.994484_{10}1$	$5.516_{10}-2$
$3.300000_{10}2$	$6.500000_{10}1$	$6.496267_{10}1$	$3.733_{10}-2$
$3.400000_{10}2$	$8.000000_{10}1$	$7.974711_{10}1$	$2.529_{10}-1$
$3.500000_{10}2$	$9.400000_{10}1$	$9.424362_{10}1$	$-2.436_{10}-1$
$3.600000_{10}2$	$1.080000_{10}2$	$1.073590_{10}2$	$6.410_{10}-1$
$3.700000_{10}2$	$1.170000_{10}2$	$1.171861_{10}2$	$-1.861_{10}-1$
$3.800000_{10}2$	$1.230000_{10}2$	$1.226845_{10}2$	$3.155_{10}-1$
$3.900000_{10}2$	$1.260000_{10}2$	$1.257838_{10}2$	$2.162_{10}-1$
Mean error			$3.206_{10}-1$

SECTION 6

A-Function, ln(A) as a Function of 1/T

CALCULATION PARAMETERS

Correction type for x-values 4

Correction type	Normalize	Take nat. log.	Take reciprocal
2	no	yes	no
4	no	no	yes

APPROXIMATION CALCULATIONS

Calculated Main Polynomial

Power	Coefficient
0	$1.0891478_{10}5$
1	$-3.4172381_{10}8$
2	$4.2864160_{10}11$
3	$-2.6868959_{10}14$
4	$8.4171747_{10}16$
5	$-1.0542868_{10}19$

Fitting of Approximation Function

x	y, spec.	y, calc.	Error
$5.930000_{10}2$	$5.000000_{10}1$	$4.972432_{10}1$	$2.757_{10}-1$
$6.030000_{10}2$	$6.500000_{10}1$	$6.471040_{10}1$	$2.896_{10}-1$
$6.130000_{10}2$	$8.000000_{10}1$	$7.947864_{10}1$	$5.214_{10}-1$
$6.230000_{10}2$	$9.400000_{10}1$	$9.390056_{10}1$	$9.944_{10}-2$
$6.330000_{10}2$	$1.080000_{10}2$	$1.071331_{10}2$	$8.669_{10}-1$
$6.430000_{10}2$	$1.170000_{10}2$	$1.168897_{10}2$	$1.103_{10}-1$
$6.530000_{10}2$	$1.230000_{10}2$	$1.224096_{10}2$	$5.904_{10}-1$
$6.630000_{10}2$	$1.260000_{10}2$	$1.253431_{10}2$	$6.569_{10}-1$

Mean error $5.326_{10}-1$

SECTION 7

Boiling Point Curve for Ethanol

CALCULATION PARAMETERS

Number of normal points	16
Number of special points	3
Minimum value of polynomial degree	6
Maximum value of polynomial degree	6
Use automatic weighing, 0: no, 1: yes	1
Correction type for x-values	0
Correction type for y-values	0

Correction type	Normalize	Take nat. log.	Take reciprocal
0	no	no	no

DATA FOR SPECIAL POINTS

Specified Points

x	y	type	
0.00000	1.00000_{10} 2	0	y is the function itself
8.94040_{10}-1	7.81500_{10} 1	0	y is the function itself
8.94040_{10}-1	0.00000	1	y is the derivative of order 1

Calculated Modifying Polynomial

Power	Coefficient
0	1.0000000_{10} 2
1	-4.8879246_{10} 1
2	2.7336162_{10} 1

APPROXIMATION CALCULATIONS

Calculated Main Polynomial

Power	Coefficient
0	-3.0170893_{10} 2
1	2.4326338_{10} 3
2	-1.1016406_{10} 4
3	2.8692010_{10} 4
4	-4.2185528_{10} 4
5	3.2360254_{10} 4
6	-1.0014741_{10} 4

Fitting of Approximation Function

x	y, spec.	y, calc.	Error	Weight, calc.
$1.000000_{10}-2$	$9.741000_{10}1$	$9.733773_{10}1$	$7.227_{10}-2$	$6.108_{10}-5$
$2.000000_{10}-2$	$9.516000_{10}1$	$9.510300_{10}1$	$5.700_{10}-2$	$2.334_{10}-4$
$4.000000_{10}-2$	$9.160000_{10}1$	$9.166128_{10}1$	$-6.128_{10}-2$	$8.512_{10}-4$
$6.000000_{10}-2$	$8.917000_{10}1$	$8.924664_{10}1$	$-7.664_{10}-2$	$1.742_{10}-3$
$8.000000_{10}-2$	$8.756000_{10}1$	$8.754201_{10}1$	$1.799_{10}-2$	$2.810_{10}-3$
$1.000000_{10}-1$	$8.638000_{10}1$	$8.631743_{10}1$	$6.257_{10}-2$	$3.975_{10}-3$
$1.400000_{10}-1$	$8.473000_{10}1$	$8.470802_{10}1$	$2.198_{10}-2$	$6.336_{10}-3$
$1.800000_{10}-1$	$8.362000_{10}1$	$8.364929_{10}1$	$-2.929_{10}-2$	$8.422_{10}-3$
$2.500000_{10}-1$	$8.226000_{10}1$	$8.229279_{10}1$	$-3.279_{10}-2$	$1.075_{10}-2$
$3.500000_{10}-1$	$8.100000_{10}1$	$8.094516_{10}1$	$5.484_{10}-2$	$1.073_{10}-2$
$4.500000_{10}-1$	$8.003000_{10}1$	$8.006365_{10}1$	$-3.365_{10}-2$	$7.873_{10}-3$
$5.500000_{10}-1$	$7.933000_{10}1$	$7.934127_{10}1$	$-1.127_{10}-2$	$4.238_{10}-3$
$6.500000_{10}-1$	$7.880000_{10}1$	$7.875851_{10}1$	$4.149_{10}-2$	$1.499_{10}-3$
$7.500000_{10}-1$	$7.836000_{10}1$	$7.839888_{10}1$	$-3.888_{10}-2$	$2.421_{10}-4$
$8.500000_{10}-1$	$7.818000_{10}1$	$7.818226_{10}1$	$-2.261_{10}-3$	$2.718_{10}-6$
$9.500000_{10}-1$	$7.820000_{10}1$	$7.818924_{10}1$	$1.076_{10}-2$	$8.850_{10}-6$
Mean error			$4.646_{10}-2$	

8.3. Programmer i ALGOL

8.3.1. ALGOL-program til LINEAR-EQUATIONS-4

```
Program LINEAR EQUATIONS-4
begin
   boolean save, print original, print inverse, wrong;
   integer LRest, calcno, sectno, pageno, N, NRS, i, j, dtop, dinverse, drhs;
   real inveps;
   comment library LINE;
   comment library PSHIFT;
   sectno := 1;
AA: begin comment cover and headline;
      integer i;
      array B[0:39];
      comment library CENTEXT;
      comment library COVER1;
      comment library HEADLINE;
      if sectno > 1 then go to BB;
      COVER1(6, 24,
      {<Solution of Linear Equations and Matrix Inversion with Full Pivoting},
      {<GIER Program LINEAR EQUATIONS-4});
      dtop := tromleplads;
      for i := 0 step 1 until 39 do B[i] := 0;
      for i := tiltromle (B) while tromleplads > 2719 do;
      for i := 0 step 1 until 3 do B[i] := 1;
      B[5] := 1_10-12;
      til tromle (B);
BB:   HEADLINE(20, ZZ)
   end of cover and headline;
   begin comment input block;
      integer N2;
      real R;
      boolean more;
      array B[0:39];
      comment library READ5;
      comment library READ6;
      READ6(67, 67);
      tromleplads := 2719;
      fra tromle(B);
      N := B[0];
      NRS := B[1];
      save := B[2] = 1;
      print original := B[3] = 1;
comment
```

```
;
      print inverse := B[4] = 1;
      inveps := B[5];
      wrong := false;
      N2 := N↑2;
      dinverse := dtop;
      if save then dinverse := dinverse - N2;
      drhs := dinverse - N2;
      if drhs - NRS×N < 2719 then
      begin
         trykteksт({<Too many equations});
         LINE(1);
         for i := 1 step 1 until NRS + 1 do
         for j := læstegn while tegn ≠ 53 do;
         go to QQ
      end if tromleak;
      if print original then
      begin
         PSHIFT(15);
         trykml(32);
         trykteksт({<ORIGINAL MATRIX});
         LINE(2)
      end if print original;
      begin comment matrix input;
         array ROW[1:N];
         more := true;
         for i := 1 step 1 until N do
         begin
            tromleplads := dtop + (i-N)×N;
            fra tromle(ROW);
            for j := 1 step 1 until N do if more then READ5(ROW[j], j, L3);
L1:         tromleplads := dtop + (i-N)×N;
            til tromle(ROW);
            if save then
            begin
               tromleplads := dinverse + (i-N)×N;
               til tromle(ROW)
            end if save;
comment
```

```
;
            if print original then
            begin
               for j := 1 step 1 until N do
               begin
                  R := ROW[j];
                  if R = 0 then
                  tryk({-nd}, R, trykml(13))
                  else
                  tryk({-n.ddddddddd10-dd}, trykml(1), R);
                  if j:5×5 = j ∧ j ≠ N then LINE(1)
               end for j;
               LINE(1)
            end if print original
         end for i;
         tromleplads := drhs;
         if print original then LINE(2);
         if more then for i := læstegn while tegn ≠ 53 do;
         for i := 1 step 1 until NRS do
         begin
            tromleplads := drhs - (i-1)×N;
            fra tromle (ROW);
            for j := 1 step 1 until N do
            READ5(ROW[j], j, L2);
            for j := læstegn while tegn ≠ 53 do;
L2:         tromleplads := drhs - (i-1)×N;
            til tromle (ROW)
         end for i;
         go to L4;
L3:      more := false;
         go to L1;
L4:   end matrix input
   end input block;
   if N < 25 then
   begin comment core inversion;
      array MATRIX[1:N, 1:N];
      comment library INVERT2;
      tromleplads := dinverse;
      fratromle(MATRIX);
      INVERT2(N, MATRIX, inveps, E1);
comment
```

```
;
      tromleplads := dinverse;
      tiltromle (MATRIX);
      go to E2;
E1:wrong := true;
E2:end if N < 25
   else
   begin comment drum inversion;
      comment library INVERT3;
      INVERT3(N, drhs, N, inveps, E3);
      go to E4;
E3:   wrong := true;
E4: end drum inversion;
   if wrong then
   begin
      trykteksт({<ERROR IN MATRIX INVERSION});
      LINE(2)
   end wrong
   else
   begin comment matrix printing and multiplication;
      integer r;
      real R, x;
      array ROW, RS[1:N];
      if print inverse then
      begin
         PSHIFT(15);
         trykml(33);
         trykteksт({<INVERSE MATRIX});
         LINE(2);
         for i := 1 step 1 until N do
         begin
            tromleplads := dinverse + (i-N)×N;
            fra tromle (ROW);
            for j := 1 step 1 until N do
            begin
               R := ROW[j];
               if R = 0 then
               tryk({-nd}, R, trykml(13))
               else
               tryk({-n.ddddddddd10-dd}, trykml(1), R);
comment
```

```
;
            if j:5×5 = j ∧ j ≠ N then LINE(1)
         end for j;
         LINE(1)
      end for i;
      LINE(2)
   end if print inverse;
   if NRS > 0 then
   begin
      PSHIFT(15);
      trykml(26);
      tryktekst({<SOLUTION OF LINEAR EQUATIONS});
      LINE(2);
      tryktekst({<  INPUT VECTOR   SOLUTION VECTOR});
      LINE(2);
      for r := 1 step 1 until NRS do
      begin
         if NRS > 1 then
         begin
            tryktekst({<Right-hand side no.});
            tryk({-nd}, r);
            LINE(2)
         end if NRS > 1;
         tromleplads := drhs - (r-1)×N;
         fra tromle (RS);
         for i := 1 step 1 until N do
         begin
            x := 0;
            tromleplads := dinverse + (i-N)×N;
            fra tromle (ROW);
            for j := 1 step 1 until N do
            x := x + ROW[j]×RS[j];
            for R := RS[i], x do
            begin
              if R = 0 then
               tryk({-nd}, R, trykml(13))
               else
               tryk({-n.ddddddddd10-dd}, trykml(1), R)
            end for R;
            LINE(1)
         end for i;
comment
```

```
;
            LINE(2)
         end for r
      end if NRS > 0
   end matrix printing and multiplication;
   begin comment printing of inveps;
      integer m;
      array DATA[0:39];
      tromleplads := 2719;
      fra tromle (DATA);
      m := DATA[39];
      if m > 31 then
      begin
         tryktekst(‹<Minimum inversion pivot›);
         trykml(5);
         tryk(‹-n.dd10-dd›, DATA[5]);
         LINE(3)
      end if printing
   end inveps printing;
QQ: sectno := sectno + 1;
   go to AA;
ZZ: end of program;
```

8.3.2. ALGOL-program til PA-6

```
Program PA-6
begin
   integer LRest, calcno, sectno, pageno, dtop, var, obs, i, d2, j, d3;
   real ymean, meanerror, mean2;
   comment library LINE;
   comment library PSHIFT;
   sectno := 1;
AA: begin comment cover and headline;
      integer i;
      array B[0:39];
      comment library CENTEXT;
      comment library COVER1;
      comment library HEADLINE;
      if sectno > 1 then go to BB;
      COVER1(12, 31,
      {<Linear Approximation to a Function of Several Variables},
      {<GIER Program PA-6})
      for i := 0 step 1 until 39 do B[i] := 0;
      dtop := tromleplads;
      tromleplads := 2719;
      tiltromle (B);
BB:   HEADLINE(20, ZZ)
   end of cover and headline;
   begin comment input block;
      boolean more;
      array DATA[0:39];
      comment library READ5;
      comment library READ6;
      READ6( 67, 67);
      tromleplads := 2719;
      fratromle(DATA);
      var := entier(DATA[0] + 0.1);
      obs := entier(DATA[1] + 0.1);
      begin comment corlist block;
         boolean array corlist[0:var];
         for i := 0 step 1 until var do
         corlist[i] := entier(DATA[2+i] + 0.1) = 1;
         tromleplads := dtop;
         d2 := dtop + tiltromle(corlist)
      end corlist block;
comment
```

```
;
      begin comment Data material input;
         array IDENT, y[1:obs], INPLINE[-1:var], xLINE[1:var];
         boolean array corlist[0:var];
         tromleplads := dtop;
         fratromle(corlist);
         more := true;
         for i := 1 step 1 until obs do
         begin
            tromleplads := d2 - (i-1)×(var+2);
            fratromle(INPLINE);
            if more then
            for j := -1 step 1 until var do
            READ5(INPLINE[j], j, L2);
L1:         tromleplads := d2 - (i-1)×(var+2);
            tiltromle(INPLINE);
            IDENT[i] := INPLINE[-1];
            y[i] := if corlist[0] then
            ln(INPLINE[0]) else INPLINE[0];
            for j := 1 step 1 until var do
            xLINE[j] := if corlist[j] then ln(INPLINE[j]) else INPLINE[j];
            tromleplads := d2-obs×(var+2) - (i-1)×var;
            tiltromle(xLINE)
         end for i;
         if more then for i := læstegn while tegn ≠ 53 do;
         comment search for stop-e;
         go to L3;
L2:      more := false;
         go to L1;
L3:      tiltromle(y);
         tiltromle(IDENT);
         d3 := tromleplads
      end DATA input
   end input block;
   begin comment calculation block;
      real tol;
      array y, ycalc[1:obs], xmean, coef, coeferror[1:var], x[1:obs, 1:var];
      comment library FIT2;
      comment library INVERT2;
      tromleplads := d2 - obs×(var+2);
comment
```

```
;
      for i := 1 step 1 until obs do
      begin
         fratromle(xmean);
         for j := 1 step 1 until var do
         x[i, j] := xmean[j]
      end for i;
      fratromle(y);
      tol := 1_10-8;
GG:   FIT2(var, obs, x, y, xmean, ymean, ycalc, coef, coeferror, meanerror, tol, E
      tromleplads := d3;
      tiltromle(ycalc);
      tiltromle(xmean);
      tiltromle(coef);
      tiltromle(coeferror);
      go to FF;
EE:   tol := tol×1_10-4;
      go to GG;
FF: end calculation block;
   begin comment printing block;
      boolean correct;
      real yact, yobs, y0;
      boolean array corlist[0:var];
      array ycalc[1:obs], xmean, coef, coeferror[1:var], INPLINE[-1:var];
      procedure HEAD;
      begin
         integer i;
         PSHIFT(10);
         tryktekst({<<
 Obs. Ident.  y(obs)        y(calc)       });
         LRest := LRest - 1;
         for i := 1 step 1 until 3 do
         if i ≤ var then
         begin
            tryktekst({<x});
            tryk({n}, i);
            trykml(12)
         end for i;
         LINE(1);
         tryktekst({<  No.    No.});
comment
```

```
;
        LINE(2)
    end HEAD;
    tromleplads := dtop;
    fratromle(corlist);
    for i := 0 step 1 until var do
    if corlist[i] then go to PR1;
    go to NPR;
PR1: PSHIFT(8);
    trykml(30);
    tryktekst(‹<LIST OF CORRECTIONS›);
    LINE(2);
    tryktekst(
‹<The following variables are internally converted into natural logarithms:›);
    LINE(2);
    for i := 0 step 1 until var do
    if corlist[i] then
    begin
        tryktekst(if i = 0 then ‹<y› else ‹<x›);
        if i > 0 then
        tryk(if i > 9 then ‹nd› else ‹n›, i);
        trykml(3)
    end for i;
    LINE(3);
NPR: tromleplads := d3;
    fratromle(ycalc);
    fratromle(xmean);
    fratromle(coef);
    fratromle(coeferror);
    mean2 := 0;
    PSHIFT(20);
    trykml(33);
    tryktekst(‹<DATA MATERIAL›);
    LINE(2);
    HEAD;
    for i := 1 step 1 until obs do
    begin
        tromleplads := d2 - (i-1)*(var+2);
        fratromle(INPLINE);
        yact := ycalc[i];
comment
```

```
;
        if corlist[0] then yact := exp(yact);
        yobs := INPLINE[0];
        mean2 := mean2 + (yobs-yact)↑2;
        tryk({nddd}, i);
        tryk({-ndddd}, INPLINE[-1]);
        trykml(1);
        tryk({-ndd.dd00₁₀-dd}, yobs, trykml(1), yact, trykml(1));
        for j := 1 step 1 until var do
        begin
           tryk({-ndd.dd00₁₀-dd}, INPLINE[j]);
           trykml(1);
           if j:3×3 = j ∧ j < var then
           begin
              LINE(1);
              trykml(39)
           end if
        end for j;
        LINE(1);
        if LRest < 5 ∧ obs - 1 > 2 then
        begin
           LINE(100);
           HEAD
        end if
     end for i;
     LINE(1);
     tryktekst({<Mean error:});
     trykml(14);
     tryk({-ndd.dd00₁₀-dd}, sqrt(mean2/(obs-var-1)));
     LINE(3);
     PSHIFT(15);
     trykml(28);
     tryktekst({<CALCULATED COEFFICIENTS});
     LINE(2);
     tryktekst({<Expression for y: });
     LINE(2);
     tryktekst({<y := });
     if corlist[0] then
     tryktekst({<exp(});
     tryktekst({<y0 });
comment
```

```
;
      y0 := ymean;
      for j := 1 step 1 until var do
      begin
         y0 := y0 - xmean[j]×coef[j];
         correct := corlist[j];
         trykktekst(‹<< + ›);
         if correct then trykktekst(‹<ln(›);
         trykktekst(‹<x›);
         tryk(if j > 9 then ‹nd› else ‹n›, j);
         if correct then trykktekst(‹<)›);
         trykktekst(‹<×c›);
         tryk(if j > 9 then ‹nd› else ‹n›, j);
         if j:5×5 = j ∧ j < var then LINE(1)
      end for j;
      trykktekst(if corlist[0] then ‹<);› else ‹<;›);
      LINE(3);
      PSHIFT(var+5);
      trykktekst(‹<
Variable    Coefficient      Coefficient
   No.                          Error
›);
      LRest := LRest - 3;
      for j := 1 step 1 until var do
      begin
         tryk(‹ndddd›, j);
         trykml(7);
         tryk(‹-n.ddddd10-dd›, coef[j], trykml(4), coeferror[j]);
         LINE(1)
      end for j;
      LINE(1);
      trykktekst(‹<y0:›);
      trykml(9);
      tryk(‹-n.ddddd10-dd›, y0);
      LINE(3)
   end printing;
   sectno := sectno + 1;
   go to AA;
ZZ: end of program;
```

8.3.3. ALGOL-program til PA-7

Program PA-7.

```
begin
   integer LRest, calcno, sectno, pageno, dtop, V, F, xplace, i, N;
   real procedure ASS(chan, item);
   value chan, item;
   integer chan, item;
   begin
      array D[1:1];
      tromleplads := 40*chan + item;
      fra tromle(D);
      ASS := D[1]
   end ASS;
   integer procedure place(n);
   value n;
   integer n;
   begin
      integer k;
      procedure LIST(expr);
      value expr;
      integer expr;
      begin
         k := k + 1;
         if k = n then
         begin
            place := tromleplads := expr;
            go to EX
         end if
      end LIST;
      k := 0;
      LIST(2*xplace-dtop-N*i);
      LIST(70*40+39+4*V);
      LIST(place(2) + F);
EX:end place;
   comment library POLY7;
   comment library LINE;
   comment library PSHIFT;
   sectno := 1;
AA:  begin comment cover and headline;
      integer i;
      array B[0:39];
comment
```

```
;
      comment library CENTEXT;
      comment library COVER1;
      comment library HEADLINE;
      if sectno > 1 then go to BB;
      COVER1(5, 31,
      {<Polynomial Approximation to One or More Functions of Several Variables},
      {<GIER Program PA-7});
      dtop := tromleplads - 40;
      for i := 0 step 1 until 39 do B[i] := 0;
      for i := til tromle(B) while tromleplads > 2679 do;
BB:   HEADLINE(20, ZZ)
   end of cover and headline;
   begin comment input block;
      boolean table, analyse, more;
      integer drum, n, t, j, n1, group, k;
      array DATA[0:39];
      procedure FIND E;
      for t := læstegn while tegn ≠ 53 do;
      comment library READ5;
      comment library READ6;
      READ6(67, 68);
      V := ASS(67, 0);
      F := ASS(67, 1);
      N := 1;
      table := ASS(67, 6) = 1;
      analyse := ASS(67, 7) ≠ 0;
      drum := dtop;
      for i := 1 step 1 until V do
      begin
         n := ASS(68, 3×(i-1));
         if i = 1 then n1 := n;
         N := n×N;
         begin comment x-block;
            array x[1:n];
            tromleplads := drum;
            fra tromle(x);
            for j := 1 step 1 until n do
            READ5(x[j], j, L1);
            FIND E;
L1:         tromleplads := drum;
comment
```

```
;
            drum := drum + til tromle(x)
         end x-block
      end for i;
      xplace := drum;
      group := N:n1;
      for i := 1 step 1 until F do
      begin
         drum := place(1);
         more := true;
         for j := 1 step 1 until group do
         begin comment z-block;
            array z[1:n1];
            drum := drum + n1;
            tromleplads := drum;
            fra tromle(z);
            for k := 1 step 1 until n1 do
            READ5(z[k], k, L2);
            go to L3;
L2:         more := false;
L3:         tromleplads := drum;
            til tromle(z);
            if -, more then go to L4
         end z-block and for j;
         FIND E;
L4:   end for i;
      if table then READ6(69, 69);
      if analyse then READ6(70, 70)
   end input block;
   begin comment printing of general data;
      integer mcount, mfact;
      array DATA[0:39];
      comment library TABLE2;
      comment library T3;
      boolean procedure print;
      begin
         i := DATA[39];
         print := (i-(i:mfact)×mfact)×2>mfact
      end print;
      TABLE2(67, 8, NP1,
      {<GENERAL DATA}, 34);
      T3({<Number of variables}, 36, {-nddd});
comment
```

```
;
      T3(«<Number of functions», 36, «-nddd»);
      T3(«<Print polynomial coefficients, 0: no, 1: yes», 13, «-nd»);
      T3(«<Print original z-table, 0: no, 1: yes», 20, «-nd»);
      T3(«<Correction type for z-function», 27, «-nd»);
      if print then
      begin
         i := DATA[4];
         trykktekst(if i = 0 then
         «<No correction» else
         «<Take natural logarithm»);
         LINE(1)
      end if explanation;
      T3(«<Maximum sum of polynomial degrees», 23, «-ndd»);
      T3(«<Calculate extended table, 0: no, 1: yes», 18, «-nd»);
      T3(«<Polynomial function analysis type», 24, «-nd»);
      if print then
      begin
         i := DATA[7];
         trykktekst(if i = 0 then
         «<No correction» else if i = 1 then
         «<Find maximum of first function»
         else if i = 2 then
         «<Find maximum of first function with spec. values of other functions»
         else
         if i = 3 then
         «<Find specified values of all functions»
         else
         «<Vary first variable to fixed value of first function»);
         if i = 4 then
         begin
            LINE(1);
            trykktekst(«<and maximize for other variables»)
         end if i = 4;
         LINE(1)
      end if explanation;
      T3(«<Maximum number of iterations», 27, «-nddd»);
      T3(«<Minimum pivot in inversions», 31, «-n.ddd10-dd»);
      T3(«<Maximum step factor», 37, «-ndd.dd00»);
comment
```

```
;
      if DATA[39] ≥ 2048 then
      for i := 1 step 1 until F do
      begin
         tryktekst(«<Desired value of function no.»);
         tryk(«-n», i);
         trykmi(27);
         tryk(«-n.dddd10-dd», DATA[10+i]);
         LINE(1)
      end for i;
      LINE(2);
NP1: end printing of general data;
   begin comment correction of x-values;
      boolean norm, log, recip, error;
      integer j, n, cortype, d1, d2, mark;
      real X, XMIN, XMAX;
      integer array xlist, deglist, corlist[1:V];
      array DATA[0:39], NORM[1:V];
      procedure MESSAGE(explanation);
      string explanation;
      begin
         skrvvr;
         skrvtekst(explanation);
         skrv(«-ndd», skrvtekst(«< element »), j, skrvtekst(«< of variable no.»), i);
         error := true
      end MESSAGE;
      tromleplads := 40×68 + 39;
      fra tromle(DATA);
      d1 := dtop;
      d2 := xplace;
      error := false;
      for i := 1 step 1 until V do
      begin
         n := xlist[i] := DATA[3×(i-1)];
         deglist[i] := DATA[3×(i-1) + 1];
         cortype := corlist[i] := DATA[3×(i-1) + 2];
         norm := cortype:2×2 ≠ cortype;
         log := cortype:4×2 ≠ cortype:2;
         recip := cortype ≥ 4;
         begin comment x-block;
            array x[1:n];
            tromleplads := d1;
comment
```

```
;
            d1 := d1 + fra tromle(x);
            for j := 1 step 1 until n do
            begin
               X := x[j];
               if recip then
               begin
                  if X = 0 then
                  MESSAGE({<Zero x-value in});
                  else
                  X := 1/X
               end if recip;
               if log then
               begin
                  if X < 0 then
                  MESSAGE({<Negative x-value in});
                  else
                  X := ln(X)
               end if log;
               if j = 1 then XMIN := XMAX := X;
               if X < XMIN then XMIN := X;
               if X > XMAX then XMAX := X;
               x[j] := X
            end for j;
            NORM[i] := if norm then
            - (XMIN+XMAX)/2 else 0;
            if norm then
            for j := 1 step 1 until n do
            x[j] := x[j] + NORM[i];
            tromleplads := d2;
            d2 := d2 + til tromle(x)
         end x-block
      end for i;
      if error then go to ZZ;
      mark := ASS(68, 39);
      if mark ≠ 0 then
      begin
         PSHIFT(16+V);
         trykml(25);
         tryktekst({<DATA FOR VARIABLES AND DEGREES});
         LINE(1);
comment
```

```
;
      trykteskt(⟨<
   Vari-  Number   Maximum  Correc-  Nor-    Take   Take
   able     of      poly-    tion    ma-     nat.   reci-
    no.   values   nomial    type    lize    log.   procal
                   degree
⟩);
        LRest := LRest - 5;
        for i := 1 step 1 until V do
        begin
           LINE(1);
           tryk(⟨-nddddddd⟩, i, xlist[i], deglist[i], trykml(2), corlist[i]);
            cortype := corlist[i];
           for j := 1, 2, 4 do
           trykteskt(if cortype : (2×j)×2 ≠ cortype : j then
           ⟨<      yes⟩ else ⟨<      no⟩)
        end for i;
        LINE(3)
     end if mark;
     place(2);
     til tromle(xlist);
     til tromle(deglist);
     til tromle(corlist);
     til tromle(NORM)
  end correction block;
  begin comment calculation block;
     integer drumbot, coefbot;
     integer array obslist, deglist[1:V], cplace[1:F];
     comment library POLY8;
     drumbot := place(3);
     fra tromle(cplace);
     fra tromle(obslist);
     fra tromle(deglist);
     i := F;
     coefbot := place(1);
     for i := 1 step 1 until F do
     begin
        cplace[i] := coefbot;
        POLY8(V, ASS(67, 5), obslist, deglist, xplace, place(1) + 1,
              coefbot, coefbot, drumbot, ASS(67, 4), FULL)
     end for i;
comment
```

```
;
      tromleplads := drumbot;
      til tromle(cplace);
      go to EX;
FULL: skrvtekst({<
Full drum});
      go to ZZ;
EX: end calculation block;
   begin comment printing of tables;
      boolean first, corz;
      integer n1, n2, N2, groups, j, k, m, blocks, c1, c2, c, row, ii, zelem;
      real ERROR, Z, DEV, ZIN;
      integer array cplace[1:F], obslist, corlist, ilist[1:V];
      array NORM, x, xcor[1:V], TDATA[0:3*V-1];
      procedure TABLES;
      begin
         array x1[1:n1];
         procedure HEAD1;
         begin
            PSHIFT(20);
            trykml(23);
            tryktekst(if first then
            {<TABLES OF ORIGINAL FUNCTION VALUES}
            else
            {<EXTENDED TABLES OF FUNCTION VALUES});
            LINE(2)
         end HEAD1;
         procedure prleft(n);
         value n;
         integer n;
         tryk(if n < 10 then {n} else
         if n < 100 then {nd} else {ndd}, n);
         procedure CORRECT(a1, an);
         value a1, an;
         integer a1, an;
         begin
            integer i, cortype;
            real X;
            for i := a1 step 1 until an do
            begin
               X := x[i];
               cortype := corlist[i];
comment
```

```
;
            if cortype ≥ 4 then X := 1/X;
            if cortype:4×2 ≠ cortype:2 then X := ln(X);
            xcor[i] := X + NORM[i]
         end for i
      end CORRECT;
      real procedure DRUMELEM(place);
      value place;
      integer place;
      begin
         array E[1:1];
         tromleplads := place;
         fra tromle(E);
         DRUMELEM := E[1]
      end DRUMELEM;
      HEAD1;
      for i := 1 step 1 until F do
      begin
         PSHIFT(18);
         if F > 1 then
         begin
            tryktekst({<   Function no.});
            tryk({-nd}, i);
            LINE(2)
         end if F > 1;
         groups := (if first then N else N2):n1;
         if V > 1 then groups := groups:n2;
         place(3);
         fra tromle(cplace);
         fra tromle(obslist);
         tromleplads := tromleplads - V;
         fra tromle(corlist);
         fra tromle(NORM);
         tromleplads := 69×40 - 1 + 3×V;
         fra tromle(TDATA);
         ERROR := 0;
         if first then
         begin
            tromleplads := dtop;
comment
```

```
;
      fra tromle(x1)
   end
   else
   for m := 1 step 1 until n1 do
   x1[m] := TDATA[1] + (m-1)×TDATA[2];
   corz := ASS(67, 4) = 1;
   for j := 1 step 1 until groups do
   begin
      if V > 2 then
      begin
         ilist[3] := j - 1;
         for m := 4 step 1 until V do
         begin
            k := if first then obslist[m-1]
            else
            TDATA[3×(m-2)];
            ilist[m] := ilist[m-1]:k;
            ilist[m-1] := ilist[m-1] - ilist[m]×k + 1
         end for m;
         ilist[V] := ilist[V] + 1;
         if first then
         begin
            tromleplads := dtop;
            for m := 1 step 1 until V do
            begin
               array xin[1:obslist[m]];
               fra tromle(xin);
               if m > 2 then
               x[m] := xin[ilist[m]]
            end for m
         end if first
         else
         for m := 3 step 1 until V do
         x[m] := TDATA[3×m-2] + (ilist[m] - 1)×TDATA[3×m-1];
         PSHIFT(6+V);
         trykml(37);
         tryktekst(£<PART £);
comment
```

```
;
        prleft(j);
        LINE(2);
        for m := 3 step 1 until V do
        begin
            tryktekst({<x});
            tryk({n}, m);
            trykml(10);
            tryk({-n.ddddd10-dd}, x[m]);
            LINE(1)
        end for m
    end if V > 2;
    LINE(2);
    blocks := 1 + (n1-1):5;
    PSHIFT(10);
    trykml(if first then 23 else 30);
    tryktekst(if first then
    {<INPUT z, CALCULATED z, AND ERRORS}
    else
    {<CALCULATED z-VALUES});
    LINE(2);
    CORRECT(3, V);
    for k := 1 step 1 until blocks do
    begin
        c1 := 1 + 5×(k-1);
        c2 := c1 + 4;
        if c2 > n1 then c2 := n1;
        PSHIFT(8);
        trykml(14);
        tryktekst({<x1});
        LINE(1);
        if V > 1 then
        begin
            tryktekst({<   x2});
            LINE(1)
        end if V > 1;
        trykml(12);
        for c := c1 step 1 until c2 do
        tryk({-n.ddddd10-dd}, trykml(1), x1[c]);
        LINE(2);
        if V = 1 then n2 := 1;
comment
```

```
for row := 1 step 1 until n2 do
begin
   array DATA[c1:c2, 1:3];
   if first then
   begin
      PSHIFT(4);
      zelem := place(1) + c1 + (j-1)×n1×n2 + n1×(row-1)
   end if first;
   if V > 1 then
   begin
      x[2] := if first then
      DRUMELEM(dtop-n1-n2+row)
      else
      TDATA[4] + (row-1)×TDATA[5];
      tryk(⟨-n.ddddd10-dd⟩, x[2]);
      CORRECT(2,2)
   end
   else
   trykml(12);
   for c := c1 step 1 until c2 do
   begin
      x[1] := x1[c];
      CORRECT(1, 1);
      Z := POLY7(V, ii, 1, xcor[ii], cplace[i]);
      if corz then Z := exp(Z);
      if first then
      begin
         ZIN := DATA[c, 1] := DRUMELEM(zelem);
         zelem := zelem + 1;
         DATA[c, 2] := Z;
         DEV := DATA[c, 3] := Z - ZIN;
         ERROR := ERROR + DEV↑2
      end
      else DATA[c, 1] := Z;
      tryk(⟨-n.ddddd10-dd⟩, trykml(1), DATA[c, 1])
   end for c;
   LINE(1);
   if first then
   for m := 2, 3 do
   begin
      trykml(12);
comment
```

```
;
                        for c := c1 step 1 until c2 do
                        tryk({-n.ddddd10-dd}, trykml(1), DATA[c, m]);
                        LINE(1)
                     end for m;
                     if first then LINE(1)
                  end for row;
                  LINE(1)
               end for k
            end for j;
            if first then
            begin
               tryktekst({<Mean error in z:          });
               if N > V then
               ERROR := ERROR/(N-V);
               tryk({-n.ddddd10-dd}, sqrt(ERROR));
               LINE(3)
            end if first
         end for i
      end TABLES;
      n1 := ASS(68, 0);
      n2 := ASS(68, 3);
      first := true;
      if ASS(67, 3) = 1 then TABLES;
      if ASS(67, 6) = 1 then
      begin
         first := false;
         N2 := 1;
         n1 := ASS(69, 0);
         for i := 1 step 1 until V do N2 := N2*ASS(69, 3*(i-1));
         n2 := ASS(69, 3);
         TABLES
      end if table
   end printing of tables;
   if ASS(67, 2) = 1 then
   begin comment print polynomial coefficients;
      integer c1, COLMAX, COL1, ROW, INDEX, row, col;
      integer array cplace[1:F];
      array DATA[0:39], NORM[1:V];
      procedure NEW DATA;
comment
```

```
;
        begin
           fra tromle(DATA);
           c1 := 39
        end NEW DATA;
        procedure CUT(high, low);
        integer high, low;
        begin
           high := low:100;
           low := low - high×100
        end CUT;
        place(3);
        fra tromle(cplace);
        PSHIFT(10);
        trykml(24);
        tryktekst({<REVERSED POLYNOMIAL COEFFICIENTS});
        LINE(2);
        for i := 1 step 1 until F do
        begin
           PSHIFT(8);
           if F > 1 then
           begin
              tryktekst({<   Function no.});
              tryk({-nd}, i);
              LINE(2)
           end if F > 1;
           tromleplads := cplace[i];
           c1 := -1;
L1 :       if c1 < 0 then NEW DATA;
           COLMAX := DATA[c1];
           c1 := c1 - 1;
           CUT(COL1, COLMAX);
           CUT(ROW, COL1);
           CUT(INDEX, ROW);
           tryk({-nddd}, INDEX, ROW, COL1, COLMAX);
           LINE(1);
           for row := 1 step 1 until ROW do
           begin
              for col := 1 step 1 until COL1 do
              begin
                 if c1 < 0 then NEW DATA;
comment
```

```
;
                trykml(1);
                tryk({-n.ddddddddd10-dd}, DATA[c1]);
                c1 := c1 - 1;
                if col = COL1 ∨ col:5×5 = col
                then LINE(1)
             end for col;
             if COL1 < COLMAX then COL1 := COL1 + 1
          end for row;
          LINE(1);
          if INDEX < V then go to L1;
          LINE(2)
       end for i;
       PSHIFT(4);
       trykml(28);
       tryktekst({<NORMALIZING CORRECTIONS});
       LINE(2);
       tromleplads := 70×40 + 39 + V;
       fra tromle(NORM);
       for i := 1 step 1 until V do
       begin
          trykml(1);
          tryk({-n.ddddddddd10-dd}, NORM[i]);
          if i:5×5 = i then LINE(1)
       end for i;
       LINE(2)
    end printing of polynomial coefficients;
    if ASS(67, 7) ≠ 0 then
    begin comment polynomial analysis;
       boolean fin, corz;
       integer count, cond, type, j, cmax, fac, ii, cortype;
       real X, tol, maxstepfac, Y;
       integer array corlist[1:V], cplace[1:F];
       array xstart, del0x, xact, epsx, NORM, EPS, xcor[1:V], yact, FVAL[1:F],
             xold[1:(V+1)×(V+2):2, 1:V], yold[1:(V+1)×(V+2):2, 1:F];
       comment library INVERT2;
       comment library NOLEQ3;
       comment library OPT7;
       tromleplads := 67×40 + 10 + F;
       fra tromle(FVAL);
comment
```

```
;
      corz := ASS(67, 4) = 1;
      type := ASS(67, 7);
      cond := if type = 1 then 0
      else
      if type = 2 ∨ type = 4 then F - 1
      else F;
      place(3);
      fra tromle(cplace);
      tromleplads := tromleplads - 2×V;
      fra tromle(corlist);
      fra tromle(NORM);
      begin
         array ADATA[1:V, 1:3];
         tromleplads := 70×40 - 1 + 3×V;
         fra tromle(ADATA);
         for i := 1 step 1 until V do
         begin
            if corlist[i] ≠ 0 then
            begin
               for j := 1 step 1 until 3 do
               begin
                  X := EPS[i] := ADATA[i, j];
                  if corlist[i] ≥ 4 then X := 1/X;
                  if corlist[i]:4×2 ≠ corlist[i]:2
                  then X := ln(X);
                  ADATA[i, j] := X
               end for j
            end if corrections;
            xstart[i] := ADATA[i, 1]+ NORM[i];
            del0x[i] := ADATA[i, 2];
            epsx[i] := ADATA[i, 3]
         end for i
      end ADATA-block;
      PSHIFT(10);
      trykml(24);
      tryktekst({<ANALYSIS OF POLYNOMIAL FUNCTIONS});
      LINE(2);
comment
```

```
;
        maxstepfac := ASS(67, 10);
        cmax := ASS(67, 8);
        tol := ASS(67, 9);
        fac := (V+1)×(V+2):2;
        count := 0;
H1:     fin := false;
        OPT7(count, cmax, V, cond, false, type = 4, xstart, del0x, xcor,
             xold, epsx, yact, yold, tol, maxstepfac, F1, E1);
        go to G2;
F1:     LINE(1);
        trykteksт({<FOUND});
        go to G1;
E1:     LINE(1);
        trykteksт({<ERROR});
G1:     fin := true;
G2:     LINE(1);
        if (count-1):fac×fac = count - 1 then LINE(1);
        trykteksт({<x:});
        for i := 1 step 1 until V do
        begin
           X := xcor[i] - NORM[i];
           cortype := corlist[i];
           if cortype:4×2 ≠ cortype:2 then
           X := exp(X);
           if cortype ≥ 4 then X := 1/X;
           trykml(1);
           tryk({-n.ddddddd10-dd}, X);
           if i:5×5 = i ∧ i < V then
           begin
              LINE(1);
              trykml(2)
           end if
        end for i;
        LINE(1);
        trykteksт({<y:});
        for i := 1 step 1 until F do
        begin
           Y := POLY7(V, ii, 1, xcor[ii], cplace[i]);
           if corz then Y := exp(Y);
comment
```

```
;
        yact[i] := Y - FVAL[i];
        trykml(1);
        tryk({-n.dddddd10-dd}, Y);
        if i:5×5 = i ∧ i < F then
        begin
           LINE(1);
           trykml(2)
        end if
     end for i;
     if -, fin then go to H1;
     LINE(2);
     tryktekst({<e:});
     for i := 1 step 1 until V do tryk({-n.dddddd10-dd}, trykml(1), EPS[i]);
     LINE(2)
  end analysis;
  sectno := sectno + 1;
  go to AA;
ZZ: end of program;
```

8.3.4. ALGOL-program til PA-8

```
Program PA-8.
begin
   boolean weigh, auto, tables, normx, recipx, logx, recipy, logy, error;
   integer LRest, calcno, sectno, pageno, dtop, F, obs, spec, deg1, degmax, deg,
           antype, i, spec2;
   real NORMX;
   real procedure ASS(chan, item);
   value chan, item;
   integer chan, item;
   begin
      array D[1:1];
      tromleplads := 40×chan + item;
      fra tromle(D);
      ASS := D[1]
   end ASS;
   integer procedure place(n);
   value n;
   integer n;
   begin
      integer k, pl;
      procedure LIST(expr);
      value expr;
      integer expr;
      begin
         k := k + 1;
         pl := pl - expr;
         if k = n then
         begin
            place := tromleplads := pl;
            go to EX
         end if
      end LIST;
      k := 0;
      pl := dtop;
      LIST(0); comment 1: input normal points;
      LIST(obs×(F+1)); comment 2: weights;
      LIST(if weigh ∨ auto then obs else 0);
      comment 3: xcor;
comment
```

```
;
      LIST(obs);   comment 4:  ycor[1];
      LIST(F*obs); comment 5:  ymcor;
      LIST(spec);  comment 6:  xmcor;
      LIST(spec);  comment 7:  mtype;
      LIST(spec);  comment 8:  mcoef;
      LIST(spec);  comment 9:  coef[F = 1];
EX: end place;
   real procedure CORRECT(yes, possible, variable, function, message);
   value yes, variable;
   boolean yes, possible;
   real variable, function;
   string message;
   begin
      CORRECT := variable;
      if yes then
      begin
         if possible then CORRECT := function
         else
         begin
            skrvvr;
            skrvtekst(message);
            error := true
         end impossible
      end yes
   end CORRECT;
   comment library INVERT2;
   comment library LINE;
   comment library PSHIFT;
   sectno := 1;
AA: begin comment cover and headline;
      array B[0:39];
      comment library CENTEXT;
      comment library COVER1;
      comment library HEADLINE;
comment
```

```
;
      if sectno = 1 then
      begin
         COVER1(4, 31,
{<Polynomial Approximation to One or More Functions of a Single Variable},
         {<GIER Program PA-8});
         dtop := tromleplads;
         for i := 0 step 1 until 39 do B[i] := 0;
         for i := til tromle(B) while tromleplads > 2679 do
      end if;
      HEADLINE(20, ZZ)
   end of cover and headline;
   begin comment input block;
      boolean more;
      integer cortype, t, j;
      real XMIN, XMAX, X, Y;
      procedure FIND E;
      for t := læstegn while tegn ≠ 53 do;
      comment library READ5;
      comment library READ6;
      READ6(67, 67);
      F := ASS(67, 0);
      obs := ASS(67, 1);
      spec := spec2 := ASS(67, 2);
      deg1 := ASS(67, 3);
      degmax := ASS(67, 4);
      weigh := ASS(67, 5) = 1;
      auto := ASS(67, 6) = 1;
      tables := ASS(67, 7) = 1;
      antype := ASS(67, 8);
      cortype := ASS(67, 9);
      normx := cortype:2×2 ≠ cortype;
      logx := cortype:4×2 ≠ cortype:2;
      recipx := cortype ≥ 4;
      cortype := ASS(67, 10);
      logy := cortype:4×2 ≠ cortype:2;
      recipy := cortype ≥ 4;
      if spec > 0 then READ6(68, 68);
      begin comment input of normal points;
         array ycor, xcor[1:obs], INLINE[1:F+1];
comment
```

```
;
        error := false;
        more := true;
        for i := 1 step 1 until obs do
        begin
           tromleplads := dtop - (i-1)*(F+1);
           fra tromle(INLINE);
           if more then
           for j := 1 step 1 until F + 1 do
           READ5(INLINE[j], j, L1);
L2:        tromleplads := dtop - (i-1)*(F+1);
           til tromle(INLINE);
           X := INLINE[1];
           X := CORRECT(recipx, X ≠ 0, X, 1/X, {<Zero x-value});
           X := CORRECT(logx, X > 0, X, ln(X), {<Negative x-value});
           if i = 1 then XMIN := XMAX := X;
           if X < XMIN then XMIN := X;
           if X > XMAX then XMAX := X;
           xcor[i] := X
        end for i;
        NORMX := 0;
        if normx then
        begin
           NORMX := - (XMIN + XMAX)/2;
           for i := 1 step 1 until obs do
           xcor[i] := xcor[i] + NORMX
        end if normx;
        place(3);
        til tromle(xcor);
        for j := 1 step 1 until F do
        begin
           for i := 1 step 1 until obs do
           begin
              tromleplads := dtop - (i-1)*(F+1);
              fra tromle(INLINE);
              Y := INLINE[1+j];
              Y := CORRECT(recipy, Y ≠ 0, Y, 1/Y, {<Zero y-value});
              Y := CORRECT(logy, Y > 0, Y, ln(Y), {<Negative y-value});
comment
```

```
;
                ycor[i] := Y
            end for i;
            tromleplads := place(4) - obs×(j-1);
            til tromle(ycor)
        end for j;
        if more then FIND E;
        go to EX;
L1:     more := false;
        go to L2;
EX:   end input of normal points;
      if weigh then
      begin comment input of weights;
        array weight[1:obs];
        place(2);
        fra tromle(weight);
        for j := 1 step 1 until obs do
        READ5(weight[j], j, L1);
        FIND E;
L1:     place(2);
        til tromle(weight)
      end input of weights;
      if tables then READ6(69, 69)
   end input block;
   begin comment printing of calculation parameters;
      boolean p1, p2;
      integer mcount, mfact, cortype, j;
      array DATA[0:39];
      boolean procedure print;
      begin
        i := DATA[39];
        print := (i-(i:mfact)×mfact)×2 ≥ mfact
      end print;
      comment library TABLE2;
      comment library T3;
      TABLE2(67, 8, NP1,
      {<CALCULATION PARAMETERS}, 29);
      T3({<Number of functions}, 37, {-ndd});
      T3({<Number of normal points}, 33, {-ndd});
      T3({<Number of special points}, 32, {-ndd});
comment
```

```
;
      T3(‹<Minimum value of polynomial degree›, 22, ‹-ndd›);
      T3(‹<Maximum value of polynomial degree›, 22, ‹-ndd›);
      T3(‹<Use specified weights, 0: no, 1: yes›, 22, ‹-n›);
      T3(‹<Use automatic weighing, 0: no, 1: yes›, 21, ‹-n›);
      T3(‹<Print extended tables, 0: no, 1: yes›, 22, ‹-n›);
      T3(‹<Polynomial analysis type›, 34, ‹-n›);
      if print then
      begin
         i := DATA[8];
         trykteksT(if i = 0 then
         ‹<No analysis› else
         if i = 1 then
         ‹<Find maximum of first function›
         else
         ‹<Find specified value of first function›);
         LINE(1)
      end if print explanation;
      T3(‹<Correction type for x-values›, 30, ‹-n›);
      p1 := print;
      T3(‹<Correction type for y-values›, 30, ‹-n›);
      p2 := print;
      if p1 ∨ p2 then
      begin
         LINE(1);
         PSHIFT(6);
         tryktekst(‹<
     Correc-  Nor-    Take    Take
      tion    ma      nat.    reci-
      type    lize    log.    procal
›);
         LRest := LRest - 4;
         for cortype := 0 step 1 until 7 do
         begin
            for i := 9, 10 do
            if cortype = DATA[i] then
            begin
               LINE(1);
               tryk(‹-nddddddd›, cortype);
comment
```

```
;
                for j := 1, 2, 4 do
                tryktekst(if cortype:(2×j)×2 ≠ cortype:j then {<      yes}
                else {<      no});
                go to EX
              end if this type;
EX:       end for cortype;
          LINE(2)
       end if p1 ∨ p2;
       T3({<Specified value of first function}, 25, {-n.ddddd10-dd});
       T3({<Start value of x}, 42, {-n.ddddd10-dd});
       T3({<Increment in x}, 44, {-n.ddddd10-dd});
       T3({<Permissible error in x}, 36, {-n.ddddd10-dd});
       LINE(2);
NP1: end printing of parameters;
   begin comment calculation of modifying polynomial;
      integer type, j, fac, m;
      real x, y, C;
      integer array mtype, mtype2[1:spec];
      array DATA[0:2], xmcor, xmcor2, ymcor[1:spec], MATRIX[1:spec, 1:spec],
            mcoef[0:spec-1];
      integer procedure numfactor;
      begin
         if type = 0 then numfactor := 1
         else
         if type ≥ j then numfactor := 0
         else
         begin
            fac := j - 1;
            for m := 2 step 1 until type do
            fac := fac×(j-m);
            numfactor := fac
         end
      end numfactor;
      if spec > 0 then
      begin
         for i := 1 step 1 until spec do
         begin
            tromleplads := 68×40 - 1 + 3×i;
            fra tromle(DATA);
            x := DATA[0];
comment
```

```
;
            y := DATA[1];
            type := DATA[2];
            mtype[i] := type + 1;
            x := CORRECT(recipx, x ≠ 0, x, 1/x, {<Zero special x-value>});
            x := CORRECT(logx, x > 0, x, ln(x), {<Negative special x-value>});
            xmcor[i] := x := x + NORMX;
            y := CORRECT(recipy, y ≠ 0, y, 1/y, {<Zero special y-value>});
            y := CORRECT(logy, y > 0, y, ln(y), {<Negative special y-value>});
            ymcor[i] := y;
            for j := 1 step 1 until spec do
            MATRIX[i, j] := numfactor×(if j-1-type ≤ 0 then 1 else x↑(j-1-type))
        end for i;
        INVERT2(spec, MATRIX, 1_10-20, E1);
        for i := 1 step 1 until spec do
        begin
            C := 0;
            for j := 1 step 1 until spec do
            C := C + MATRIX[i, j]×ymcor[j];
            mcoef[i-1] := C
        end for i;
        spec2 := 0;
        for i := 1 step 1 until spec do
        begin
            x := xmcor[i];
            type := mtype[i];
            for j := 1 step 1 until spec do
            if i ≠ j ∧ x = xmcor[j] ∧ mtype[j] > type then go to L1;
            spec2 := spec2 + 1;
            xmcor2[spec2] := x;
            mtype2[spec2] := type;
L1:     end for i;
        place(5);
        til tromle(ymcor);
        til tromle(xmcor2);
        til tromle(mtype2);
        til tromle(mcoef);
        if ASS(68, 39) ≠ 0 then
        begin comment printing of modifying polynomial;
comment
```

```
;
            PSHIFT(6+spec);
            trykml(28);
            tryktekst({<DATA FOR SPECIAL POINTS});
            LINE(2);
            tryktekst({<Specified Points});
            LINE(2);
            tryktekst({<      x                y          type});
            LINE(2);
            for i := 1  step 1 until spec do
            begin
               tromleplads := 68×40 - 1 + 3×i;
               fra tromle(DATA);
               tryk({-n.ddddd10-d}, DATA[0], trykml(5), DATA[1]);
               tryk({-ndddd}, DATA[2]);
               trykml(4);
               type := DATA[2];
               if type = 0 then
               tryktekst({<y is the function itself})
               else
               tryk({-nd}, tryktekst(
               {<y is the derivative of order}), type);
               LINE(1)
            end for i;
            LINE(1);
            PSHIFT(4 + spec);
            tryktekst({<
Calculated Modifying Polynomial

Power     Coefficient
});
            LRest := LRest - 4;
            for i := 0 step 1 until spec - 1 do
            begin
               LINE(1);
               tryk({-nd}, i);
               tryk({-n.ddddddd10-dd}, trykml(5), mcoef[i])
            end for i;
comment
```

```
;
            LINE(3)
         end if printing
      end if spec > 0;
      go to EX;
E1:   skrvtekst(skrvvr, {<Inversion error>});
      error := true;
EX: end calculation of modifying polynomial;
   begin comment main polynomial block;
      integer degbest, j, k;
      real EMIN, ERROR, A, B, X, Y, YIN, DEV, W;
      integer array mtype[1:spec];
      array xcor, ycor, weight[1:obs], xmcor[1:spec], mcoef[0:spec-1],
            INLINE[1:F+1], EL[1:1];
      real procedure PROD;
      begin
         A := 1;
         X := xcor[i];
         for j := 1 step 1 until spec2 do
         A := A*(X-xmcor[j])/mtype[j];
         PROD := A
      end PROD;
      real procedure MODPOL;
      begin
         B := 0;
         for j := spec - 1 step -1 until 0 do
         B := B*X + mcoef[j];
         MODPOL := B
      end MODPOL;
      procedure FIT(final, func);
      value final, func;
      boolean final;
      integer func;
      begin
         array coef[0:deg];
         comment library POLY1;
         tromleplads := place(4) - obs*(func-1);
         fra tromle(ycor);
         for i := 1 step 1 until obs do
         begin
            ycor[i] := 1/PROD*(ycor[i] - MODPOL);
comment
```

```
;
            if auto then weight[i] := A↑2×(if weigh then weight[i] else 1)
         end for i;
         POLY1(obs, deg, xcor, ycor, weight, coef, weigh ∨ auto);
         if final then
         begin
            tromleplads := place(9) - (deg+1)×(func-1);
            til tromle(coef);
            PSHIFT(4+spec);
            tryktekst({<
Calculated Main Polynomial

Power     Coefficient
});
            LRest := LRest - 4;
            for i := 0 step 1 until deg do
            begin
               LINE(1);
               tryk({-nd}, i);
               tryk({-n.ddddddd10-dd}, trykml(5), coef[i])
            end for i;
            if normx then
            begin
               LINE(2);
               tryktekst({<NORMX:  });
               tryk({-n.ddddddd10-dd}, NORMX)
            end if normx;
            LINE(3);
            PSHIFT(10);
            tryktekst({<
                        Fitting of Approximation Function

   x          y, spec.        y, calc.     Error});
         LRest := LRest - 3;
         if weigh then
         tryktekst({<     Weight});
         if auto then
         tryktekst({<    Weight, calc.});
         LINE(2)
      end if final;
comment
```

```
;
        ERROR := 0;
        for i := 1 step 1 until obs do
        begin
           Y := 0;
           PROD;
           for j := deg step -1 until 0 do
           Y := Y×X + coef[j];
           Y := MODPOL + A×Y;
           Y := CORRECT(logy, Y < 354, Y, exp(Y), {<Large calculated y-value});
           Y := CORRECT(recipy, Y ≠ 0, Y, 1/Y, {<Zero calculated y-value});
           tromleplads := dtop - (i-1)×(F+1);
           fra tromle(INLINE);
           YIN := INLINE[1+func];
           DEV := YIN - Y;
           ERROR := ERROR + DEV↑2;
           if final then
           begin
              tryk({-n.dddddd10-dd}, INLINE[1], trykml(2), YIN, trykml(2), Y);
              tryk({-n.ddd10-dd}, trykml(2), DEV);
              if weigh ∨ auto then
              begin
                 tromleplads := place(3) + i;
                 fra tromle(EL);
                 W := EL[1];
                 if weigh then
                 tryk({-n.ddd10-dd}, trykml(2), W);
                 if auto then
                 tryk({-n.ddd10-dd}, trykml(2), weight[i]/(if weigh then W else 1))
              end if weights;
              LINE(1)
           end if final
        end for i
     end FIT;
     place(2);
     if weigh ∨ auto then fra tromle(weight);
     fra tromle(xcor);
     if spec > 0 then
     begin
        place(6);
comment
```

```
;
        fra tromle(xmcor);
        fra tromle(mtype);
        fra tromle(mcoef)
      end if spec > 0;
      EMIN := 1_10100;
      degbest := deg1;
      if deg1 ≠ degmax then
      for deg := deg1 step 1 until degmax do
      begin
        FIT(false, 1);
        if ERROR < EMIN then
        begin
          EMIN := ERROR;
          degbest := deg
        end
        else go to L1
      end for deg;
L1:   deg := degbest;
      PSHIFT(12);
      trykml(27);
      tryktekst({<APPROXIMATION CALCULATIONS});
      LINE(2);
      for k := 1 step 1 until F do
      begin
        if F > 1 then
        begin
          tryktekst({<   Function no.});
          tryk({-nd}, k);
          LINE(2)
        end if F > 1;
        FIT(true, k);
        if obs > 1 then ERROR := ERROR/(obs-1);
        LINE(1);
        tryktekst({<Mean error});
        trykml(35);
        tryk({-n.ddd_10-dd}, sqrt(ERROR));
        LINE(3)
      end for k
    end main polynomial block;
  comment
```

```
;
   if tables ∨ antype > 0 then
   begin comment printing of extended tables and polynomial analysis;
      boolean first, second;
      integer j, k, groups;
      real A, B, X, S, x;
      integer array mtype[1:spec];
      array xmcor[1:spec], mcoef[0:spec-1], coef[0:deg];
      real procedure MODPOL(order);
      value order;
      integer order;
      begin
         B := 0;
         for j := spec-1 step -1 until order do
         B := B×X + mcoef[j]×(if order = 0 then
         1 else if order = 1 then j else j×(j-1));
         MODPOL := B
      end MODPOL;
      real procedure MAINPOL(order);
      value order;
      integer order;
      begin
         B := 0;
         for j := deg step -1 until order do
         B := B×X + coef[j]×(if order = 0 then
         1 else if order = 1 then j else j×(j-1));
         MAINPOL := B
      end MAINPOL;
      real procedure PROD(order);
      value order;
      integer order;
      begin
         integer power, factor, r, s;
         S := 0;
         if spec = 0 then PROD := if order = 0 then 1 else 0
         else
         begin
            for k := 1 step 1 until spec↑order do
            begin
               A := 1;
comment
```

```
;
                for j := 1 step 1 until spec2 do
                begin
                   power := mtype[j];
                   factor := 1;
                   r := 1 + (k-1)÷spec2;
                   s := k - (r-1)×spec2;
                   if order  = 2 ∧ r = s then
                   begin
                      factor := power×(power-1);
                      power := power - 2
                   end
                   else
                   if order = 2 ∧ r ≠ s ∧ (r = j ∨ s = j) ∨ order = 1 ∧ s = j then
                   begin
                      factor := power;
                      power := power - 1
                   end;
                   A := A×(X-xmcor[j])↑power×factor
                end for j;
                S := S + A
             end for k;
             PROD := S
          end
       end PROD;
       real procedure y(order);
       value order;
       integer order;
       begin
          real Y, dYdX, d2YdX2, MO, PR, MA, MO1, PR1, MA1, dydY, dXdx, d2Xdx2;
          MO := MODPOL(0);
          PR := PROD(0);
          MA := MAINPOL(0);
          Y := MO + PR×MA;
          if order = 0 then
          begin
             Y := CORRECT(logy, Y < 354, Y, exp(Y), {<Large y-value>});
             Y := CORRECT(recipy, Y ≠ 0, Y, 1/Y, {<Zero y-value>});
comment
```

```
;
            y := Y
        end if order = 0
        else
        begin
            MO1 := MODPOL(1);
            PR1 := PROD(1);
            MA1 := MAINPOL(1);
            dYdX := MO1 + PR×MA1 + PR1×MA;
            dydY := CORRECT(logy, Y < 354, Y, exp(Y), {<Large dy-value>});
            if recipy then
            dydY := CORRECT(recipy, Y ≠ 0, Y, -1/Y↑2, {<Zero dy-value>});
            if logy ∧ recipy then
            dydY := CORRECT(true, Y < 354, Y, -exp(-Y), {<Zero dy-value>});
            if ¬ logy ∧ ¬ recipy then dydY := 1;
            if ¬ logx ∧ ¬ recipx then dXdx := 1
            else
            begin
                if x = 0 then dXdx := CORRECT(true, false, 1, 0, {<Zero x-value>})
                else
                begin
                    dXdx := if logx then 1/x else -1/x↑2;
                    if logx ∧ recipx then
                    dXdx := -1/x
                end
            end;
            if order = 1 then
            y := dydY×dYdX×dXdx
            else
            begin
                if ¬ logx ∧ ¬ recipx then d2Xdx2 := 0
                else
                begin
                    if x = 0 then d2Xdx2 := CORRECT
                    (true, false, 0, 0, {<Zero x-value>})
                    else
                    begin
                        d2Xdx2 := if logx then -1/x↑2 else -2/x↑3;
                        if logx ∧ recipx then d2Xdx2 := 1/x↑2
                    end
                end;
comment
```

```
;
            y := dydY×(dYdX×d2Xdx2 + dXdx↑2×(MIDPOL(2) + PR×MAINPOL(2)
                 + 2×PR1×MA1 + PROD(2)×MA))
         end if order = 2
      end order ≠ 0
   end y;
   procedure TABLE(order);
   value order;
   integer order;
   begin
      integer g, NP, k, np;
      real SV, INCR;
      array DATA[1:groups, 1:3];
      tromleplads := 69×40 + 2 + 3×groups;
      fra tromle(DATA);
      PSHIFT(10);
      for g := 1 step 1 until groups do
      begin
         NP := DATA[g, 1];
         SV := DATA[g, 2];
         INCR := DATA[g, 3];
         PSHIFT(5);
         tryktekst({< x↓, dx-> });
         for k := 1 step 1 until 5 do
         tryk({-n.dddddd10-dd}, trykml(1), (k-1)×INCR);
         LINE(1);
         for np := 1 step 1 until NP do
         begin
            X := x := SV + (np-1)×INCR;
            if (np-1)÷5×5 = np - 1 then
            begin
               LINE(1);
               tryk({-n.ddd10-dd}, x)
            end if first column;
            X := CORRECT(recipx, X ≠ 0, X, 1/X, {<Zero x-value});
            X := CORRECT(logx, X > 0, X, ln(X), {<Negative x-value});
            X := X + NORMX;
            tryk({-n.dddddd10-dd}, trykml(1), y(order))
         end for np;
         LINE(2)
      end for g;
comment
```

```
;
      LINE(1)
   end TABLE;
   if tables then
   begin
      groups := ASS(69, 0);
      PSHIFT(20);
      trykml(28);
      tryktekst({<EXTENDED FUNCTION TABLES});
      LINE(2);
      for i := 1 step 1 until F do
      begin
         PSHIFT(18);
         if F > 1 then
         begin
            tryktekst({<   Function no.});
            tryk({-nd}, i);
            LINE(2)
         end if F > 1;
         first := ASS(69, 1) = 1;
         second := ASS(69, 2) = 1;
         if first ∨ second then
         begin
            tryktekst({<Values of function:});
            LINE(2)
         end if first ∨ second;
         place(6);
         if spec > 0 then
         begin
            fra tromle(xmcor);
            fra tromle(mtype);
            fra tromle(mcoef)
         end if spec > 0;
         tromleplads := place(9) - (deg+1)×(i-1);
         fra tromle(coef);
         TABLE(0);
         if first then
         begin
            PSHIFT(16);
            tryktekst({<First-order derivative:});
comment
```

```
;
                LINE(2);
                TABLE(1)
            end if first;
            if second then
            begin
                PSHIFT(16);
                tryktekst({<Second-order derivative:});
                LINE(2);
                TABLE(2)
            end if second
        end for i
    end if tables;
    if antype > 0 then
    begin
        integer count;
        boolean fin;
        real FVAL, yy;
        array xstart, del0x, xact, epsx, yact, y0[1:1], yold[1:1, 1:1];
        comment library NOLEQ3;
        place(6);
        if spec > 0 then
        begin
            fra tromle(xmcor);
            fra tromle(mtype);
            fra tromle(mcoef)
        end if spec > 0;
        fra tromle(coef);
        FVAL := ASS(67, 11);
        xstart[1] := ASS(67, 12);
        del0x[1] := ASS(67, 13);
        epsx[1] := ASS(67, 14);
        PSHIFT(10);
        trykml(30);
        tryktekst({<POLYNOMIAL ANALYSIS});
        LINE(2);
        tryktekst({<     x}); trykml(13);
        count := 0; tryktekst(if antype = 1 then {<dy/dx} else {<y});
        LINE(2);
comment
```

```
;
H1:     fin := false;
        NOLEQ3(1, count, 50, false, xstart, del0x, xact, epsx, yact, yold,
              y0, 1_10-20, 4, F1, E1);
        go to G2;
F1:     LINE(1);
        tryktekst({<FOUND});
        go to G1;
E1:     LINE(1);
        tryktekst({<ERROR});
G1:     fin := true;
G2:     LINE(1);
        X := xact[1];
        X := CORRECT(recipx, X ≠ 0, X, 1/X, {<Zero x-value});
        X := CORRECT(logx, X > 0, X, ln(X), {<Negative x-value});
        X := X + NORMX;
        yy := y(if antype = 1 then 1 else 0);
        yact[1] := yy - FVAL;
        tryk({-n.dddddd_10-dd}, xact[1], trykml(1), yy);
        if -, fin then go to H1;
        LINE(3)
    end analysis
  end table ∨ analysis;
  if error then
  begin
    tryktekst({<CALCULATION ERROR});
    LINE(3)
  end if error;
  sectno := sectno + 1;
  go to AA;
ZZ: end of program;
```