FÖRORD

Denna rapport redovisar ett examensarbete, vilket är en del av civilingenjörsutbildningen. Arbetet har utförts vid Institutionen för Reglerteknik, Lunds Tekniska Högskola med universitetslektor Björn Wittenmark och civ. ing. Torkel Glad som handledare.

Avsikten var att undersöka några extremalsökande algoritmer och implementera dessa på en mikrodator av typen Intel 8080. Några körningar på mikrodator har emellertid ej ägt rum dels på grund av att den övriga undersökningen tog längre tid än beräknat och dels att mikrodatorn byggdes parallellt av en annan exjobbare och ej var riktigt klar när detta arbete avslutades. Fortsättningen av detta arbete kommer att utföras av en annan examensarbetare.

Jag riktar ett tack till Björn Wittenmark och Torkel Glad för handledning och tackar speciellt examensarbetare Jan-Eric Aspernäs som har utvecklat programspråket HILP för mikrodatorer. Användningen av HILP förenklade mitt programmeringsarbete betydligt.

Inge Sixtensson
Maj 1976

INNEHÅLLSFÖRTECKNING

ABSTRACT

This report deals with two extremal-seeking algorithms namely Fletcher-Reeves and Absolute Bias. The former is based on the theory of conjugated directions while the latter is a random seeking method. These algorithms have been programmed in FORTRAN and tested with different parametervalues. This have been done in order to get a general idea of their behaviour.

When translating FORTRAN-code to 8080 machine language we used a high level language for 8080 which has been developed at the Department of Automatic Control at Lund Institute of Technology. It is called HILP. Further development of this language is going on for the present. To get an idea of the effectiveness of HILP, I have done some testing-programs and tried these in HILP and PL/M. Where PL/M is a high level language developed by Intel for their 8008 and 8080 microprocessors.

Finally the extremal-seeking algorithms have been programmed in HILP and are thereby almost ready to be loaded into an 8080 microcomputor.

SAMMANFATTNING

Denna rapport behandlar två extremalsökande algoritmer nämligen Fletcher-Reeves metod och metoden Absolute Bias. Den förra metoden bygger på teorin om konjugerade riktningar medan den senare är en slumpsökningsmetod. Dessa algoritmer har programmerats i FORTRAN och testats med olika parametervärden. Detta har gjorts för att få en allmän uppfattning om deras beteende.

För att överföra FORTRAN-programmen till 8080 maskinkod användes ett högnivåspråk för 8080 som utvecklats vid institutionen för Reglerteknik vid Lunds Tekniska Högskola. Språket heter HILP och vidareutveckling av detta pågår för närvarande. För att få en uppfattning om hur pass effektivt språket är har jag gjort några testprogram och kört dessa dels i HILP och dels i PL/M. PL/M är ett av Intel utvecklat högnivåspråk för 8008 och 8080.

Slutligen har programmen om extremalsökning programmerats i HILP och dessa är därmed i stort sett klara för att stoppas in i en 8080 mikrodator.

SLUTSATS

Undersökningen av de båda metoderna gav att Fletcher-Reeves metod var bäst vid testfunktionen Rosenbrocks curved valley. Detta var att vänta då slumpsökningsmetoden är effektivast när funktionen är betydligt enklare. Absolute Bias hade också den nackdelen att ett flertal parametrar skulle justeras in innan metoden gav bra resultat. När de båda huvudprogrammens storlek jämfördes var Fletcher-Reeves ungefär dubbelt så stor som Absolute Bias. Minnesutrymmet i mikrodatorn var begränsat till 2k och det var knappt att Fletcher-Reeves fick plats ty subrutinerna tog också stor plats. Om ytterligare subrutiner tillkommer måste nog programmet förenklas med följd att effektiviteten blir något sämre.
Jämförelsen mellan PL/M och HILP gav att HILP genererar mindre kod än vad PL/M gör beroende på att HILP inte alltid ligger på lika hög programmeringsnivå som PL/M. Den största fördelen med HILP är dock att man kan programmera på en valfri nivå och vid behov skriva tämligen optimala program.

KAPITEL 1. INLEDNING

Anledningen till att man studerar extremalsökande algoritmer inom reglertekniken är att man har en fysikalisk process som man vill styra optimalt. Processen är begränsad av vissa villkor t. ex. att effekten ej får överstiga ett bestämt värde eller att koncentrationen av ett ämne i en vätska skall vara nära c_0. Man konstruerar en förlustfunktion som tar hänsyn till dessa villkor. Det bästa uppförandet hos processen har man sedan när förlustfunktionen antar sitt minimivärde.

Kapitel 2 behandlar teorin för två extremalsökande algoritmer. Resultatet av testkörningar med dessa programmerade i FORTRAN redovisas också.

Kapitel 3 behandlar några olika alternativ som användaren har för att få programmet i sådan form att det kan matas in i en Intel 8080 mikrodator. En jämförelse mellan två högnivåspråk för 8080, PL/M och HILP,görs. Denna jämförelse baseras på antalet ord som genereras för ett visst program.

Kapitel 4 innehåller något om talrepresentation i en mikrodator samt ovanstående algoritmer programmerade i HILP.

I appendix finns alla program listade.

KAPITEL 2. OM EXTREMALSÖKANDE ALGORITMER

2.1 Inledning

Detta kapitel förklarar lite om extremalsökande algoritmer och beskriver två stycken lite närmare, nämligen en sökmetod Absolute Bias och en descentmetod Fletcher-Reeves. Resultatet av körningar med dessa metoder på testfunktioner finns också redovisade. I slutet av kapitlet ges synpunkter på en jämförelse mellan metoderna.

2.2 Allmänt om sökmetoder och descentmetoder

Under rubriken sökmetoder samlas metoder som bygger på att objektfunktionens värde jämförs i olika punkter. Gemensamt är också att de inte försöker bygga upp information om objektfunktionens utseende som förstörs av dåliga kontinuitetsegenskaper. Exempel på sökmetoder är Pattern Search, Rosenbrocks metod, Simplexmetoden samt olika former av slumpsökningsmetoder. Dessa algoritmer är generellt långsammare än descentmetoder men kan vara mera robusta t. ex. då objektfunktionen inte har kontinuerliga derivator. Simplexmetoden, Rosenbrocks metod och Pattern Search är i stort sett lika snabba men slumpsökningsmetoderna är betydligt långsammare.
I descentmetoder består varje iteration av i huvudsak tre delar. Först hittar man en descentriktning d^k sedan bestämmer man en steglängd λ^k och till slut görs steget $x^{k+1} = x^k + \lambda^k d^k$
Descentriktningen är en n-dimensionell vektor. d^k säges vara en descentriktning med avseende på funktionen $q(x)$ vid punkten x^k om det finns ett $\bar{\lambda} > 0$ så att för alla λ där $\bar{\lambda} \geq \lambda > 0$ har vi att

$$q(x^{k+1}) = q(x^k + \lambda d^k) < q(x^k)$$

om $q(x)$ är differentierbar så är d^k en descentriktning om

$$\lim_{\lambda \to 0} \frac{q(x^k + \lambda d^k) - q(x^k)}{\lambda} = \frac{d\ q(x^k + \lambda d^k)}{d\lambda}\Bigg|_{\lambda=0} =$$

$$= (d^k)^T g^k < 0$$

där g^k är gradienten av funktionen $q(x)$ i punkten x^k. Om $q(x)$ är differentierbar är produkten $(d^k)^T g^k$ definitionsmässigt riktningsderivatan av $q(x)$ i riktningen d^k tagen i punkten x^k. Steglängden λ är en skalär och kan väljas optimalt eller ej. Optimalt λ har vi, om vi med steget λ når minimipunkten på linjen som är definierad av d^k. Att finna optimalt λ medför mycket arbete och tar extra tid, därför nöjer man sig vanligtvis med ett λ-värde som ger hyfsad funktionsvärdesminskning. Descentmetoder kan uppdelas i Steepest descent metoder och metoder som använder konjugerade riktningar.

Skillnader mellan sökmetoder och descentmetoder är bland annat den mängd information som överförs från en iteration till nästa. Sökmetoder i allmänhet använder sig bara till liten del av gammal information man säger att de har litet minne, vilket gör att de speciellt lämpar sig för problem där störningar förekommer. Fortsättningsvis är det så att man genererar en ny riktning i varje iteration i de flesta descentmetoder medan man i många sökmetoder bara testar funktionsvärden längs ett antal förutbestämda riktningar enligt något visst system. Varje descent-iteration innehåller vanligtvis mera beräkningsarbete men ger i gengäld oftast avsevärd förbättring av funktionsvärdet. Man kan vänta sig att descentmetoder skall konvergera till en lösning i färre antal steg än hos sökmetoder.

Rent allmänt kan man säga att descentmetoder är mera tillförlitliga och effektiva än sökmetoder, speciellt då variablernas antal är stort. Sökmetoder är emellertid enkla och effektiva för problem med få variabler.

2.3.1 Teoretisk bakgrund till Fletcher-Reeves metod. [4]

Fletcher-Reeves metod är en descentmetod som bygger på teorin för konjugerade riktningar. Motivet för användning av konjugerade riktningar inses om vi studerar en kvadratisk funktion

$$q(x) = \frac{1}{2} x^T Q x - b^T x$$

Vi har följande definition:

$\{d_k\}_1^n$ är konjugerade m.a.p. den positivt definita matrisen Q om $d_i^T Q d_j = 0 \quad i \neq j$

dvs. $\{d_k\}_1^n$ är linjärt oberoende om de är konjugerade. Då kan

varje vektor x skrivas på följande sätt

$$x = \sum_{i=1}^{n} \alpha_i d_i \qquad \text{för något val av } \{\alpha_i\}_1^n$$

Lösningen till det kvadratiska problemet satisfierar $Qx = b$.
Detta ger

$$\sum_{i=1}^{n} \alpha_i Q d_i = b \qquad \text{och}$$

$$\alpha_i = \frac{d_i^T b}{d_i^T Q d_i}$$

varför lösningen ges av

$$\hat{x} = \sum_{i=1}^{n} \frac{d_i^T b}{d_i^T Q d_i} d_i$$

vilket kan ses som n steg från $x^0 = 0$ med $\{d_i\}$ som sökriktningar.

För en kvadratisk funktion når vi minimum i n steg från en godtycklig startpunkt x^0 om sökriktningarna är konjugerade och steglängden är optimal.

Tekniken innebär alltså en sekvens av n minimeringar längs descentriktningarna. Steglängderna λ^k skall vara optimala dvs. de skall vara lösningar till ett endimensionellt minimeringsproblem längs respektive descentriktningar

$$q(x^k + \lambda^k d^k) = \min_{\lambda} q(x^k + \lambda d^k) \quad , \; k=0,..,n-1$$

Descentlinjen vid x^k är en linje genom x^k som pekar i descentriktningen d^k. Sträckan längs denna linje mäts i λ som är ett positivt tal. Vi kan se att derivatan av $q(x)$ i riktningen d^k tagen i punkten x^{k+1} är noll. Detta gäller för alla k=0, ...,n-1 dvs. $(d^k)^T g^{k+1} = 0 \qquad k=0, \ldots,n-1$
där g^{k+1} betecknar derivatan i punkten x^{k+1}. Med andra ord är gradienten i punkten x^{k+1} ortogonal mot riktningarna d^k k=0, ...,n-1. Betrakta funktionen i slutpunkten $q(x^n)$ som en funktion av n steglängder λ^k. När denna funktion antar sitt minimum är $g^n = 0$ då gäller även

$$\frac{\partial q(x^n)}{\partial \lambda^1} = (d^k)^T g^n = 0 \qquad k=0, \ldots,n-1$$

Detta betyder att riktningsderivatan av funktionen $q(x)$ i varje

konjugerad descentriktning, beräknad i punkten x^n är noll.
Fletcher-Reeves metod använder sig av en vanlig form av Gram-Schmidt ortogonaliseringsprocess för att generera de konjugerade descentriktningarna. Metoden bygger på att $(d^m)^T g^k = 0$, m=0,...,k-1
Härav följer att gradientriktningen g^k är linjärt beroende av alla tidigare konjugerade riktningar $d^0, d^1, \ldots d^{k-1}$ och därför kan den användas för att generera en ny konjugerad riktning.
Vi får följande förslag till formel för descentriktningen d^k

$$d^k = -g^k + \sum_{i=1}^{k} \alpha^i d^{i-1}$$

där α^i är skalärer valda så att d^k är konjugerad till alla tidigare riktningar $d^0, \ldots d^{k-1}$. Vi skall nu bestämma utseendet hos α^i. Start sker med $d^0 = -g^0$ där derivatan är beräknad i startpunkten x^0. Med hjälp av villkoret för konjugerade riktningar $(d^k)^T Q d^m = 0$,m=0, ...,k-1 skall vi nu bestämma α^i. Enligt formeln har vi $d^1 = -g^1 + \alpha^1 d^0$ och alltså

$$(d^1)^T Q d^0 = (-g^1 + \alpha^1 d^0)^T Q d^0 = 0$$

Detta kan vi skriva om eftersom

$$q(x) = \frac{1}{2} x^T Q x - b^T x \Rightarrow g = Qx - b$$

$$x^{k+1} - x^k = \lambda^k d^k$$

$$Qd^0 = \frac{1}{\lambda^0} Q(x^1 - x^0) = \frac{1}{\lambda^0}(g^1 - g^0)$$

ty $g^1 - g^0 = Qx^1 - b - (Qx^0 - b) = Q(x^1 - x^0)$
om $\lambda^0 \neq 0$ så har vi

$$(-g^1 - \alpha^1 g^0)^T (g^1 - g^0) = 0$$

$$\alpha^1 = \frac{(g^1)^T g^1}{(g^0)^T g^0}$$

ty $(g^1)^T d^0 = -(g^1)^T g^0 = 0$
på liknande sätt får vi

$$\alpha^2 = \frac{(g^2)^T g^2}{(g^1)^T g^L} \quad \text{och} \quad \alpha^1 = 0$$

Den generella formeln för α^k vid beräkning av den konjugerade riktningen d^k är

$$\alpha^k = \frac{(g^k)^T g^k}{(g^{k-1})^T g^{k-1}} \qquad \alpha^m = 0 \text{ för } m=0, \ldots, k-1$$

Eftersom d^0 är en descentriktning så följer av ovanstående att även d^k k=1, ... är descentriktningar. Dessa formler kan även användas för att bestämma descentriktningar när q(x) är en generell icke-linjär funktion. I detta fall blir emellertid sökriktningarna ej konjugerade. Praktiska experiment visar på en förbättring om metoden omstartas då och då. Detta beror på att informationen från ett steg till ett annat t. ex. genom avrundningar vid beräkningar, kan bli något felaktig. Ett förslag är omstart efter n+1 iterationer.

Här följer lämplig uppläggning för ett program som använder sig av Fletcher-Reeves metod.

1. Initialvärden, sätt k=0, beräkna $q(x^0)$ där x^0 är given.
2. Beräkna gradienten g^k i punkten x^k.
3. Beräkna descentriktningen d^k där $d^k = -g^k + \alpha^k d^{k-1}$

$$\alpha^k = \frac{(g^k)^T g^k}{(g^{k-1})^T g^{k-1}} \qquad \alpha^m = 0 \text{ för } m=0, \ldots, k-1$$

4. Beräkna steglängden λ^k så att

$$q(x^k + \lambda^k d^k) = \min_{\lambda} q(x^k + \lambda d^k)$$

5. Beräkna steget $\Delta x^k = \lambda^k d^k$
6. Gör steget $x^{k+1} = x^k + \Delta x^k$
7. Beräkna funktionsvärdet $q(x^{k+1})$
8. Om $q(x^k) - q(x^{k+1}) < \varepsilon_1$ och $\| \Delta x^k \| < \varepsilon_2$ avsluta iterationen annars gå till steg 9. ε_1 och ε_2 är förutbestämda gränsvärden.
9. Acceptera punkten x^{k+1}, om k< n gör k=k+1 annars $x^0 = x^{k+1}$ k=0 gå till steg 2.

Allmänt gäller om konjugerade gradientmetoder att de överallt har goda konvergensegenskaper och är relativt lätta att programmera Fletcher-Reeves metod behöver ej stort minnesutrymme och är lämplig t. ex. för en mikrodator.

2.3.2. Beskrivning av programmet

För att på bästa sätt testa ut programmet användes FORTRAN som programspråk. Detta gjorde att t. ex. tidtagning på algoritmen

var möjlig. Det var också väldigt lätt att komma åt för att köra FORTRAN-program eftersom institutionens PDP-15 användes. Programmet för Fletcher-Reeves metod finns listat i appendix A och skall nu närmare förklaras.

Testfunktionen har lagts in som ett FUNCTION-underprogram FN(X). Detta beräknar bara funktionsvärdet i punkten x, där x vanligtvis är en vektor. Beräkningen av gradienten sker numeriskt med hjälp av differenser i en subrutin, antingen NUDER1 eller NUDER2. I den förra fås derivatan enligt följande formel

$$y'(x) = \frac{1}{h}(y(x+h) - y(x)) \quad \text{där felet är } \mathcal{O}(h)$$

I den andra är formeln

$$y'(x) = \frac{1}{2h}(y(x+h) - y(x-h)) \quad \text{där felet är } \mathcal{O}(h^2).$$

Den andra formeln är alltså mera exakt men kräver i gengäld fler funktionsberäkningar och tar alltså längre tid. I ett program kan man lämpligen göra så att NUDER1 används i början av algoritmen och NUDER2 när vi ligger i närheten av minimum. Denna metod har emellertid ej använts här eftersom det har visat sig att på testfunktionen som kallas "Rosenbrocks curved valley" har NUDER1 gett väldigt dåligt resultat redan vid startpunkten.

TIME10 är en subrutin som finns i FORTRAN-biblioteket hos PDP-15. Tiden mäts från anropet tills satsen IOFF=1 påträffas. Tiden redovisas som minst i tiondels sekunder.

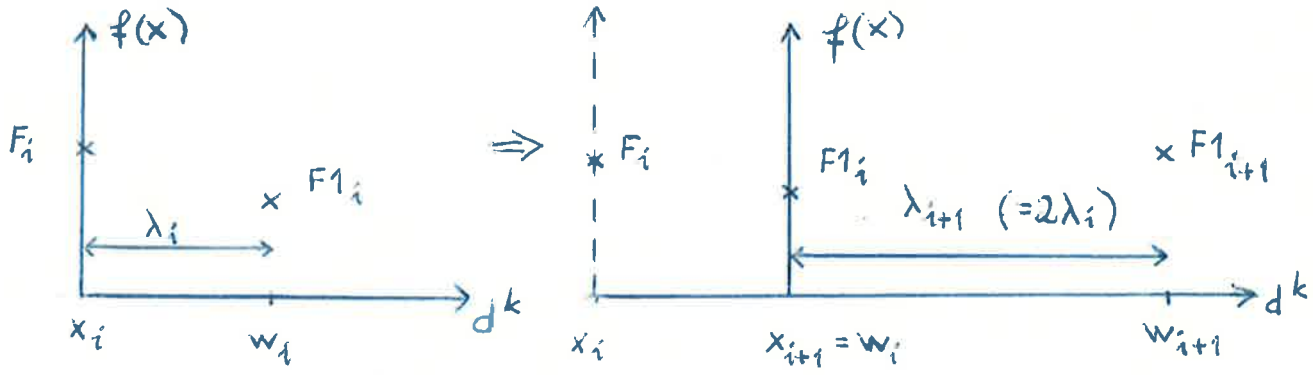
Skalären α^k fås genom att ALPHA=DXNEW/DXOLD där DXNEW = $(g^k)^T g^k$.

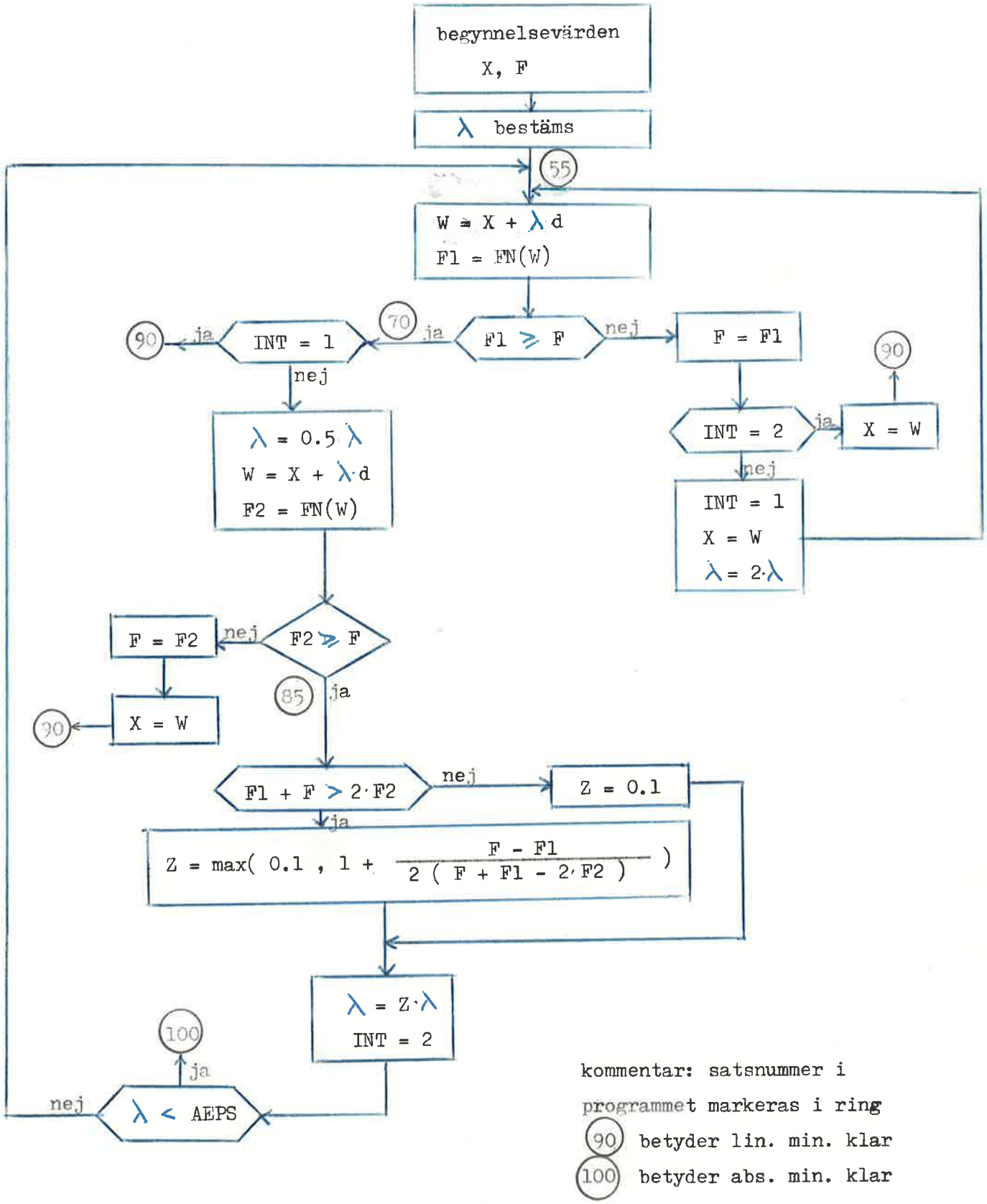
När vi beräknat en sökriktning d^k = RIKTN(I) undersöker vi värdet på riktningsderivatan $(g^k)^T d^k$ = GSO, om GSO >0 är riktningen ej en descentriktning och vi undersöker då funktionen längs den negativa riktningen.

Den linjära minimeringen sker enligt flödesschemat figur 2-1.

Med hjälp av några figurer skall minimeringen närmare belysas.

Fall 1. Det nya funktionsvärdet F1 är mindre än F dvs. förbättring.



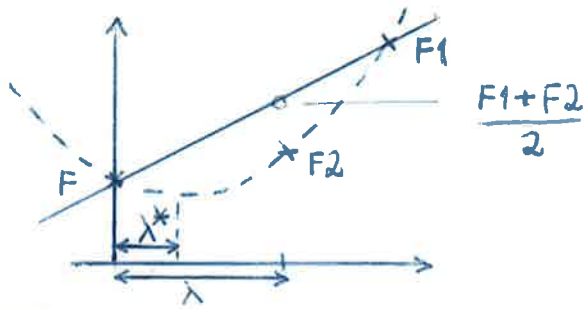


figur 2-1 Flödesschema över linjär minimering.
Fletcher-Reeves metod

Om $Fl_{i+1} < F_{i+1}$ upprepas proceduren. Om $Fl_{i+1} > F_{i+1}$ så accepteras x_{i+1} som det linjära minimat.

Fall 2. Det nya funktionsvärdet Fl är större än F. Detta innebär att eftersom riktningen d^k är en descentriktning så finns ett funktionsvärde mindre än F. Detta kommer då att accepteras som det linjära minimat. Vi börjar med att sätta $\lambda_{i+1} = 0.5\,\lambda_i$ och beräknar funktionsvärdet F2. Beroende på hur detta ligger i förhållande till F och Fl fås två fall.

a) $\frac{Fl + F}{2} > F2$

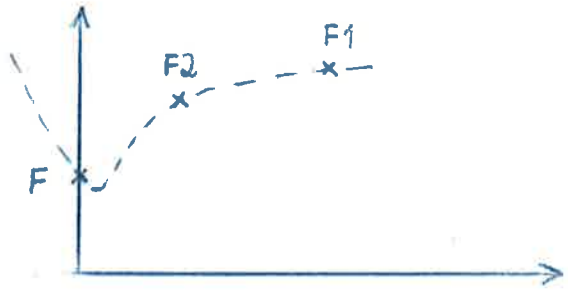


i detta fall approximerar vi en andragradskurva genom punkterna och söker det λ^* som minimerar kurvan dvs. vi interpolerar kvadratiskt.

$$\lambda^* = \lambda\left(1 + \frac{F - Fl}{2(F + Fl - 2\cdot F2)}\right)$$

Vi väljer emellertid ej det nya λ-värdet hur litet som helst utan minst $\frac{1}{10}$ av det gamla värdet.

b) $\frac{Fl + F}{2} < F2$



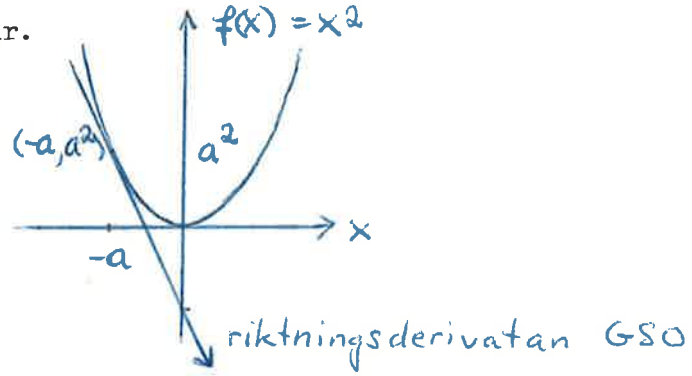
Det linjära minimat ligger troligen nära det ursprungliga värdet. Vi väljer $\lambda_{i+1} = 0.1\cdot\lambda_i$

Bestämning av det första λ-värdet vid den linjära minimeringen kan ha stor inverkan på snabbheten hos algoritmen. I programmet finns en variabel IND med vars hjälp man kan bestämma enligt vilken metod väljs.

IND=0 Detta är det enklaste sättet. Vi tar det λ-värde som blev bäst vid föregående minimering.

IND≠0 Här har vi en mer komplicerad metod. Vi antar att kur-

van är kvadratisk och att vi känner till hur mycket funktionsvärdet kommer att förbättras. Då kan vi beräkna den steglängd som behövs för att nå minimum. Vi behöver dessutom beräkna riktningsderivatan GSO = $(g^k)^T d^k$. Se figur.



$f'(x) = 2 \cdot x$ dvs. riktningsderivatan i punkten är

$$GSO = f'(-a) = -2 \cdot a$$

Från denna punkt är steget till minimipunkten (dvs. optimala steglängden) $\lambda_{opt} = a$

Funktionsförbättringen är $DFF = a^2$

Ur detta får vi följande formel.

$$\lambda_{opt} = \frac{-2\,DFF}{GSO}$$

Problemet är förutom att kurvan kanske ej kan approximeras som kvadratisk, att vi ej känner DFF. Vi kan emellertid beräkna funktionsförbättringen vid föregående minimering DF. Vi antar att DFF = konst.· DF Denna konstant kan vi ändra genom värdet på IND på så sätt att

$$DFF = \frac{DF}{IND}$$

Rent experimentellt finner man det IND-värde som är bäst.
För testfunktionen "Rosenbrocks curved valley" blev IND = 1 bäst, se närmare resultatet av olika körningar i nästa avsnitt.
Algoritmen avslutas om något av följande villkor uppfylls
$\lambda < \varepsilon_1$ eller $q(x^k) - q(x^{k+1}) < \varepsilon_2$. Det sista villkoret är ekvivalent med $DF < \varepsilon_2$

2.3.3. Resultat

Fletcher-Reeves metod har använts på testfunktionen

$$f(x_1,x_2) = 100 \cdot (x_2 - x_1^2)^2 + (1 - x_1)^2$$

funktionen finns uppritad i figur 2-20, på sidan 2-38.

Startpunkt (-1.2,1.0) med funktionsvärdet 24.2
Minimipunkt (1.0,1.0) med funktionsvärdet 0.0
Denna testfunktion kallas Rosenbrocks curved valley.
De parametrar som kan ändras är följande: H,IND. Övriga är konstanta dvs. EPS1 = 0.1 10^{-4}, EPS2 = 0.1 10^{-6}. Vi börjar med H = 0.1 10^{-4} och undersöker IND = 0,1,2,3,4 se figur 2-2,2-3, 2-4. IND =1 verkar klart bäst dvs. vi antar att funktionsförbättringen blir ungefär lika stor som den var vid föregående iteration. IND = 0 är också hyfsad och undersöks närmare eftersom startvärdet på XLAMB vid varje iteration då är samma som slutvärdet på XLAMB vid föregående iteration. För IND = 2, figur 2-3, får vi ett för tidigt avbrott på grund av att DF EPS2. När EPS2 minskats till 0.1 10^{-8} går vi förbi den kritiska punkten och fortsätter. Figur 2-5 och 2-6 visar att om H ökar försämras konvergensen, Detta är troligt, ty noggrannheten vid beräkningen av derivatan blir då sämre. Det bästa resultatet var alltså med parametrarna H = 0.1 10^{-4} och IND = 1.

2.3.4. Allmänt om parametrars inverkan

H är differensen vid beräkningen av derivatan och bör väljas så liten som möjligt för att få en bra approximation av derivatan. Man kan dock inte välja H hur litet som helst beroende på den noggrannhet med vilken ett tal kan representeras.
IND är en parameter som beror av funktionens utseende. IND = 0 ger nog alltid ett hyggligt resultat medan övriga IND-värden kan ge både bättre och sämre resultat. IND = 1 och IND = 2 är nog ganska vanliga och bör testas.
Subrutinen NUDER2 dvs. beräkning av derivatan med hjälp av centrala differenser, bör man använda när funktionen är besvärlig. Man kan testa både NUDER1 och NUDER2. Om skillnaden ej blir stor kan man använda NUDER1 ty denna är snabbare. Sätter man noggrannhet före snabbhet bör man använda NUDER2 även för enkla funktioner.

figur 2-2

```
 PARAMETERS
 EPS1= 0.100E-04    EPS2= 0.100E-06
 H= 0.10E-04    IND= 0

      F(X)
  1  0.568E+01
     0.420E+01
     0.415E+01
     0.412E+01
     0.408E+01
     0.389E+01
     0.375E+01
     0.374E+01
     0.216E+01
 10  0.214E+01
     0.182E+01
     0.167E+01
     0.141E+01
     0.135E+01
     0.769E+00
     0.738E+00
     0.489E+00
     0.330E+00
     0.225E+00
 20  0.222E+00
     0.219E+00
     0.215E+00
     0.149E+00
     0.140E+00
     0.117E+00
     0.792E-01
     0.762E-01
     0.426E-01
     0.391E-01
 30  0.350E-01
     0.349E-01
     0.349E-01
     0.185E-01
     0.162E-01
     0.161E-01
     0.161E-01
     0.160E-01
     0.647E-02
     0.642E-02
 40  0.614E-02
 41  0.441E-02
     0.440E-02
     0.439E-02
     0.149E-02
     0.332E-03
     0.329E-03
     0.328E-03
     0.326E-03
     0.325E-03
 50  0.328E-04
     0.302E-04
     0.298E-04
     0.294E-04
     0.293E-04

 NUMBER OF ITERATIONS 55

 POINT OF MINIMUM
         0.995
         0.989
 WITH THE FUNCTIONVALUE 0.292E-04
```

```
 PARAMETERS
 EPS1= 0.100E-04    EPS2= 0.100E-06
 H= 0.10E-04    IND= 1

      F(X)
  1  0.763E+01
     0.413E+01
     0.412E+01
     0.411E+01
     0.363E+01
     0.291E+01
     0.281E+01
     0.280E+01
     0.142E+01
 10  0.131E+01
     0.128E+01
     0.544E+00
     0.530E+00
     0.374E+00
     0.358E+00
     0.354E+00
     0.336E+00
     0.273E+00
     0.267E+00
 20  0.592E-01
     0.138E-01
     0.677E-02
     0.672E-02
     0.196E-02
     0.238E-03
     0.973E-04
     0.971E-04
     0.968E-04
     0.966E-04
 30  0.964E-04
     0.962E-04
     0.961E-04
     0.959E-04
     0.956E-04
     0.942E-04
     0.900E-04
     0.898E-04
     0.132E-04
     0.215E-05

 NUMBER OF ITERATIONS 40

 POINT OF MINIMUM
         0.999
         0.997
 WITH THE FUNCTIONVALUE 0.215E-05
```

figur 2-3

```
PARAMETERS
EPS1= 0.100E-08   EPS2= 0.100E-06
H= 0.10E-04   IND= 2

   F(X)
 0.453E+01
 0.414E+01
 0.412E+01
 0.411E+01
 0.290E+01
 0.254E+01
 0.184E+01
 0.176E+01
 0.122E+01
 0.121E+01
 0.500E+00
 0.457E+00
 0.449E+00
 0.373E+00
 0.227E+00
 0.220E+00
 0.196E+00
 0.195E+00
 0.192E+00
 0.191E+00
 0.191E+00
 0.188E+00
 0.187E+00
 0.186E+00
 0.179E+00
 0.178E+00

NUMBER OF ITERATIONS 27

POINT OF MINIMUM
        0.582
        0.333
WITH THE FUNCTIONVALUE 0.178E+00
```

```
PARAMETERS
EPS1= 0.100E-08   EPS2= 0.100E-08
H= 0.10E-04   IND= 2

     F(X)
 1 0.453E+01
   0.414E+01
   0.412E+01
   0.411E+01
   0.290E+01
   0.254E+01
   0.184E+01
   0.176E+01
   0.122E+01
10 0.121E+01
   0.500E+00
   0.457E+00
   0.449E+00
   0.373E+00
   0.227E+00
   0.220E+00
   0.196E+00
   0.195E+00
   0.192E+00
20 0.191E+00
   0.191E+00
   0.188E+00
   0.187E+00
   0.186E+00
   0.179E+00
   0.178E+00
   0.178E+00
   0.176E+00
   0.151E+00
30 0.553E-01
   0.129E-01
   0.127E-01
   0.122E-01
   0.122E-01
   0.489E-02
   0.419E-02
   0.411E-02
   0.406E-02
   0.406E-02
40 0.405E-02
41 0.403E-02
   0.399E-02
   0.386E-02
   0.385E-02
   0.377E-02
   0.371E-02
   0.371E-02
   0.371E-02
   0.370E-02
50 0.354E-02
   0.222E-03
   0.213E-03
   0.440E-04
   0.262E-04
   0.133E-04
   0.129E-04
   0.100E-04
   0.999E-05
   0.975E-05
60 0.753E-05
   0.726E-05
   0.207E-06
   0.187E-06

NUMBER OF ITERATIONS 64

POINT OF MINIMUM
        1.000
        1.001
WITH THE FUNCTIONVALUE 0.187E-06
```

figur 2-4

```
 PARAMETERS
 EPS1= 0.100E-04   EPS2= 0.100E-06
 H= 0.10E-04   IND= 4

      F(X)
 1  0.413E+01
    0.412E+01
    0.342E+01
    0.340E+01
    0.319E+01
    0.318E+01
    0.316E+01
    0.305E+01
    0.277E+01
10  0.255E+01
    0.250E+01
    0.175E+01
    0.156E+01
    0.925E+00
    0.715E+00
    0.649E+00
    0.635E+00
    0.269E+00
    0.228E+00
20  0.212E+00
    0.208E+00
    0.160E+00
    0.635E-01
    0.625E-01
    0.620E-01
    0.609E-01
    0.588E-01
    0.583E-01
    0.570E-01
30  0.402E-01
    0.275E-01
    0.273E-01
    0.974E-02
    0.887E-02
    0.724E-02
    0.723E-02
    0.721E-02
    0.172E-02
    0.153E-02
40  0.152E-02
41  0.183E-03
    0.124E-03
    0.456E-04
    0.408E-04
    0.296E-04
    0.260E-04
    0.259E-04
    0.117E-04
    0.513E-05
50  0.429E-05
    0.403E-05

NUMBER OF ITERATIONS 52

POINT OF MINIMUM
        1.002
        1.004
WITH THE FUNCTIONVALUE 0.402E-05
```

```
 PARAMETERS
 EPS1= 0.100E-04   EPS2= 0.100E-06
 H= 0.10E-04   IND= 3

      F(X)
 1  0.587E+01
    0.524E+01
    0.413E+01
    0.407E+01
    0.282E+01
    0.240E+01
    0.197E+01
    0.134E+01
    0.985E+00
10  0.955E+00
    0.881E+00
    0.618E+00
    0.545E+00
    0.519E+00
    0.454E+00
    0.428E+00
    0.422E+00
    0.160E+00
    0.124E+00
20  0.122E+00
    0.114E+00
    0.106E+00
    0.106E+00
    0.105E+00
    0.101E+00
    0.101E+00
    0.973E-01
    0.939E-01
    0.674E-01
30  0.500E-01
    0.410E-01
    0.380E-01
    0.377E-01
    0.371E-01
    0.369E-01
    0.363E-01
    0.360E-01
    0.359E-01
    0.318E-01
40  0.269E-01
41  0.266E-01
    0.249E-01
    0.225E-01
    0.221E-01
    0.219E-01
    0.219E-01
    0.720E-02
    0.575E-02
    0.301E-02
50  0.290E-02
    0.224E-02
    0.196E-02
    0.194E-02
    0.191E-02
    0.191E-02
    0.138E-02
    0.141E-03
    0.618E-04
    0.155E-04
60  0.138E-04

NUMBER OF ITERATIONS 61

POINT OF MINIMUM
        1.004
        1.007
WITH THE FUNCTIONVALUE 0.138E-04
```

figur 2-5

$EPS1 = 0.1\ 10^{-4}$

$EPS2 = 0.1\ 10^{-6}$

$IND = 0$

$H = 0.1\ 10^{-2}$

	F(X)
1	0.568E+01
	0.420E+01
	0.415E+01
	0.412E+01
	0.408E+01
	0.390E+01
	0.376E+01
	0.375E+01
	0.370E+01
10	0.360E+01
	0.359E+01
	0.358E+01
	0.357E+01
	0.343E+01
	0.318E+01
	0.249E+01
	0.247E+01
	0.194E+01
	0.164E+01
20	0.133E+01
	0.124E+01
	0.111E+01
	0.730E+00
	0.495E+00
	0.451E+00
	0.436E+00
	0.251E+00
	0.249E+00
	0.224E+00
30	0.198E+00
	0.197E+00
	0.176E+00
	0.172E+00
	0.129E+00
	0.126E+00
	0.125E+00
	0.124E+00
	0.122E+00
	0.932E-01
40	0.863E-01
41	0.859E-01
	0.656E-01
	0.508E-01
	0.370E-01
	0.251E-01
	0.757E-03
	0.723E-03
	0.714E-03
	0.707E-03
50	0.703E-03
	0.698E-03
	0.696E-03
	0.685E-03
	0.634E-03
	0.549E-03
	0.537E-03
	0.531E-03
	0.246E-03
	0.538E-04
60	0.470E-04
	0.223E-04
	0.129E-05
	0.118E-05

NUMBER OF ITERATIONS 64

POINT OF MINIMUM
1.001
1.002
WITH THE FUNCTIONVALUE 0.118E-05

$H = 0.1\ 10^{-4}$

	F(X)
1	0.568E+01
	0.420E+01
	0.415E+01
	0.412E+01
	0.408E+01
	0.389E+01
	0.375E+01
	0.374E+01
	0.216E+01
10	0.214E+01
	0.182E+01
	0.167E+01
	0.141E+01
	0.135E+01
	0.769E+00
	0.738E+00
	0.489E+00
	0.330E+00
	0.225E+00
20	0.222E+00
	0.219E+00
	0.215E+00
	0.149E+00
	0.140E+00
	0.117E+00
	0.792E-01
	0.762E-01
	0.426E-01
	0.391E-01
30	0.350E-01
	0.349E-01
	0.349E-01
	0.185E-01
	0.162E-01
	0.161E-01
	0.161E-01
	0.160E-01
	0.647E-02
	0.642E-02
40	0.614E-02
41	0.441E-02
	0.440E-02
	0.439E-02
	0.149E-02
	0.332E-03
	0.329E-03
	0.328E-03
	0.326E-03
	0.325E-03
50	0.328E-04
	0.302E-04
	0.298E-04
	0.294E-04
	0.293E-04

NUMBER OF ITERATIONS 55

POINT OF MINIMUM
0.995
0.989
WITH THE FUNCTIONVALUE 0.292E-04

figur 2-6

$EPS1 = 0.1\ 10^{-4}$

$EPS2 = 0.1\ 10^{-6}$

IND = 1

$H = 0.1\ 10^{-2}$

	F(X)
1	0.763E+01
	0.413E+01
	0.412E+01
	0.411E+01
	0.366E+01
	0.299E+01
	0.296E+01
	0.265E+01
	0.251E+01
10	0.245E+01
	0.234E+01
	0.219E+01
	0.216E+01
	0.182E+01
	0.166E+01
	0.133E+01
	0.128E+01
	0.913E+00
	0.870E+00
20	0.561E+00
	0.434E+00
	0.422E+00
	0.183E+00
	0.180E+00
	0.178E+00
	0.154E+00
	0.127E+00
	0.125E+00
	0.118E+00
30	0.905E-01
	0.864E-01
	0.267E-01
	0.257E-01
	0.730E-02
	0.530E-02
	0.528E-02
	0.526E-02
	0.472E-02
	0.385E-02
40	0.971E-03
41	0.969E-03
	0.950E-03
	0.942E-03
	0.938E-03
	0.919E-03
	0.900E-03
	0.402E-03
	0.778E-04
	0.560E-04
50	0.555E-04
	0.166E-05
	0.115E-05

NUMBER OF ITERATIONS 53

POINT OF MINIMUM

0.999

0.998

WITH THE FUNCTIONVALUE 0.115E-05

$H = 0.1\ 10^{-4}$

	F(X)
1	0.763E+01
	0.413E+01
	0.412E+01
	0.411E+01
	0.363E+01
	0.291E+01
	0.281E+01
	0.280E+01
	0.142E+01
10	0.131E+01
	0.128E+01
	0.544E+00
	0.530E+00
	0.374E+00
	0.358E+00
	0.354E+00
	0.336E+00
	0.273E+00
	0.267E+00
20	0.592E-01
	0.138E-01
	0.677E-02
	0.672E-02
	0.196E-02
	0.238E-03
	0.973E-04
	0.971E-04
	0.968E-04
	0.966E-04
30	0.964E-04
	0.962E-04
	0.961E-04
	0.959E-04
	0.956E-04
	0.942E-04
	0.900E-04
	0.898E-04
	0.132E-04
	0.215E-05

NUMBER OF ITERATIONS 40

POINT OF MINIMUM

0.999

0.997

WITH THE FUNCTIONVALUE 0.215E-05

2.4.1. Teoretisk bakgrund till metoden Absolute Bias [1]

Absolute Bias är en slumpmetod där sökriktningen fås av en slumpvektor. Steglängden sätts till 1 dvs.

$$u^{k+1} = u^m + du^{k+1}$$

där du^{k+1} är slumpvektorn som är normalfördelad med medelvärde noll och standardavvikelse σ. I denna metod bestäms nästa ändring Δu^{k+1} endast av de två sista experimenten. Sökningen i en funnen lyckosam riktning fortsätts tills en sämre punkt påträffas. Den punkt som för närvarande är bäst kallas q^m och tillhörande koordinater kallas u^m dvs. $q^m = q(u^m)$. Den punkt som undersöks kallas q^k och dess koordinater u^k. Det nya värdet q^k accepteras om $q^k < q^m - \varepsilon$ där ε är den minst accepterade förbättringen. När störningar kan förväntas vid beräkning av funktionsvärdet bör ε vara skild från noll.

Inför straffunktionen ς^k, definierad som

$$\varsigma^k = \begin{cases} 0 & \text{om } q^k < q^m - \varepsilon \quad \text{(dvs. förbättring)} \\ 1 & \text{om } q^k \geq q^m - \varepsilon \end{cases}$$

ς^k används till att uppdatera u^m och q^m

$$u^m = \varsigma^k u^m + (1 - \varsigma^k) u^k$$

$$q^m = \varsigma^k q^m + (1 - \varsigma^k) q^k$$

Inför r^k och s^k där r^k definieras som

$$r^k = \begin{cases} 0 & \text{om } \varsigma^k = 0 \text{ eller } \Delta u^k = du^k \\ r^{k-1} + 1 & \text{om } \varsigma^k = 1 \end{cases}$$

r^k räknar antalet misslyckade försök i följd innan ny slumpriktning väljs.

s^k definieras som

$$s^k = \begin{cases} 0 & \text{om } \varsigma^k = 1 \\ s^{k-1} + 1 & \text{om } \varsigma^k = 0 \end{cases}$$

s^k är antalet experiment i följd som varit lyckosamma.

I algoritmen fås ett nytt värde genom formeln

$$u^{k+1} = u^m + \Delta u^{k+1}$$

där ändringen Δu^{k+1} definieras enligt följande

1. $\Delta u^{k+1} = \Delta u^k$ om $\varsigma^k = 0$ dvs. sista steget upprepas om det var lyckosamt.
2. $\Delta u^{k+1} = -\Delta u^k$ om $\varsigma^k = 1$ och $r^k = 1$ och $s^{k-1} = 0$ dvs. om experimentet k ej var lyckosamt och det genererades en ny slumpriktning i föregående experiment k-1, så tas ett steg i mot-

satt riktning.

3. $\Delta u^{k+1} = du^{k+1}$ om $\varsigma^k = 1$ och $r^k \geq 2$ eller $s^{k-1} \geq 1$ dvs. om varken du^k eller $-du^k$ ger förbättring eller om vi har ett misslyckat försök efter ett eller flera i följd lyckosamma försök så väljs en ny slumpvektor.

Då ändringen Δu^{k+1} väljs från en normalfördelning med medelvärde noll så kan steglängden $|\Delta u^{k+1}|$ bli mycket liten, vilket kan ge långsam optimering. För att undvika detta inför vi att

$$\Delta u^{k+1} = du^{k+1} \quad \text{om} \quad \varsigma^k = 0 \quad \text{och} \quad s^k \geq S$$

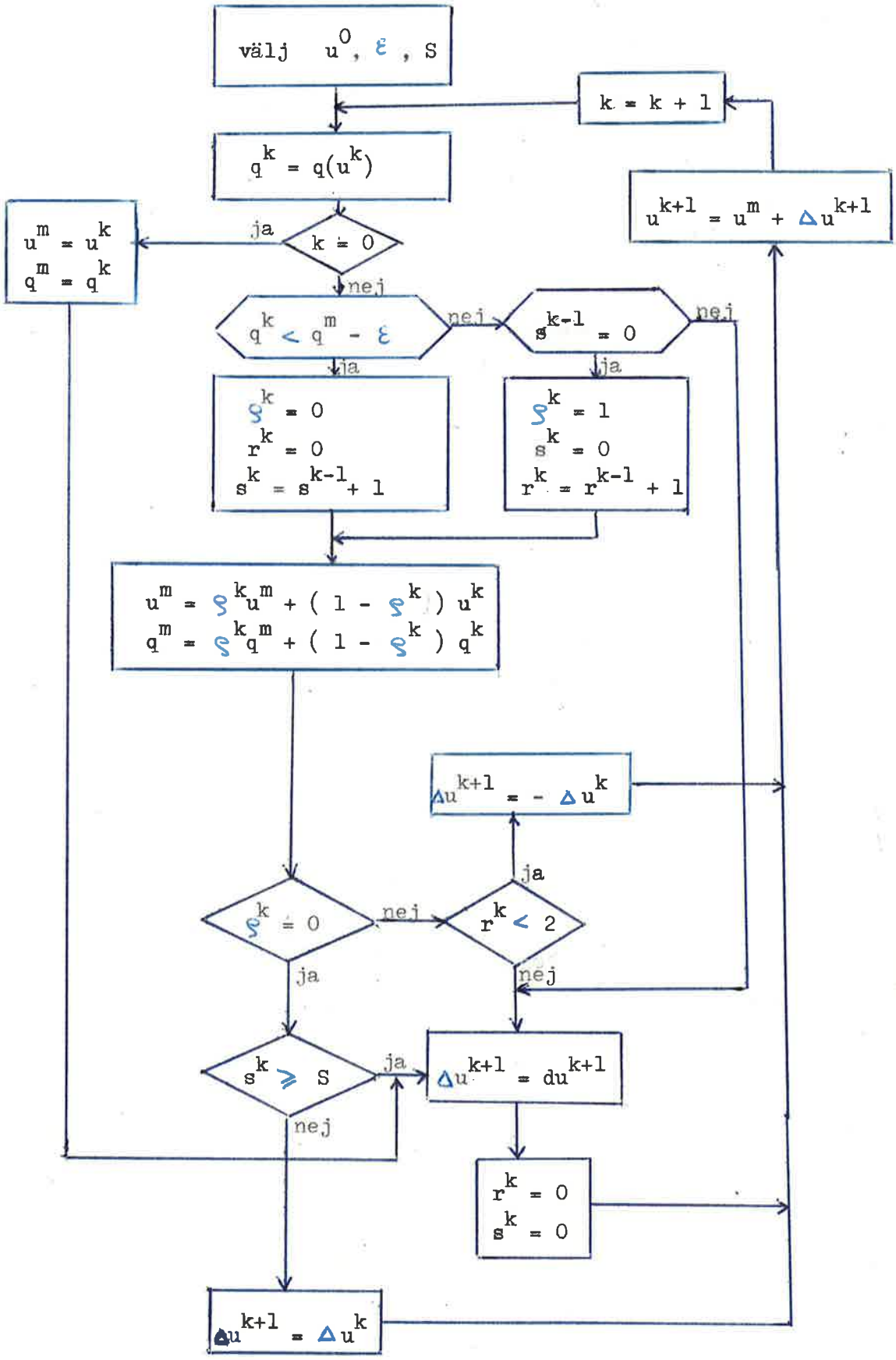
där S är det maximalt tillåtna antalet lyckosamma försök i följd. Algoritmen beskrivs i flödesschemat i figur 2-7. I flödesschemat finns inget som gör att minimeringen stoppas eftersom det i regel är meningen att den skall användas på en fysikalisk process och då sker minimeringen så länge processen är igång. Vid körning av testfunktioner vill man dock ha något villkor som stoppar minimeringen, detta beskrivs i nästa avsnitt i samband med beskrivning av programmet.

Störningar vid beräkning av funktionsvärdet uppträder i samband med minimering av en fysikalisk process t. ex. på grund av otillräckliga mätmetoder. Detta gör att en mätpunkt som i själva verket är dålig kan förklaras som bra vilket kan göra att sökmetoden spårar ur och ej har en chans att nå minimipunkten. Om störningar föreligger, är slumpmässiga sökmetoder bra, ty dessa har relativt litet minne, vilket innebär att en felaktig slutsats om en mätpunkt endast påverkar sökningen i ett eller ett par steg.

Vi har tidigare infört ε som den minsta accepterade funktionsvärdesförbättringen. Ju större ε är desto mindre är sannolikheten att ett felaktigt värde accepteras.

För att ytterligare reducera störningars effekt bör man låta ingångssignalen passera ett exponentialfilter. Detta har följande utseende $y_n = (1 - \beta) y_{n-1} + \beta \cdot x_n$

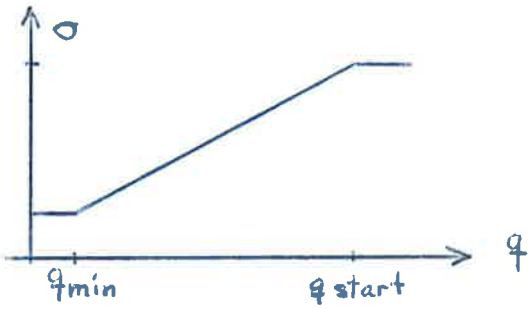
där y är det filtrerade värdet och x är det ofiltrerade ingångsvärdet. β kallas filterkonstant. Stor vikt vid gamla värden fås om β är litet. Drift i en fysikalisk process ställer också till problem. Detta kan man klara av genom att starta om försöket med ett nytt värde på q^m. Vi har följande kriterium för omstart. Omstart sker om förra försöket var misslyckat och om $z^k < Z$ där



figur 2-7 Algoritm för Absolute Bias

z^k är ett rektangelfördelat slumptal mellan 0 och 1 genererat i försök k. Z är en konstant mellan 0 och 1 dvs. sannolikheten för omstart. Vid våldsam drift skall Z vara stort. För det mesta kan man välja $0.2 < Z < 0.5$

För att sökmetoden skall vara så effektiv som möjligt bör man på något sätt justera steglängden så att långa steg tas långt från minimipunkten och korta steg i närheten av densamma. I metoden Absolute Bias kan vi ändra på den genomsnittliga steglängden genom att ändra på standardavvikelsen σ i normalfördelningen. Om vi vet eller kan gissa oss till funktionsvärdet i startpunkten respektive minimipunkten kan vi låta σ vara en funktion av funktionsvärdet q enligt följande figur



En annan metod kommer redovisas i nästa avsnitt i samband med beskrivningen av programmet.

2.4.2. Beskrivning av programmet

Programmet för metoden Absolute Bias skrivet i FORTRAN finns listat i appendix A och skall nu närmare förklaras.

Testfunktionen har lagts in som ett FUNCTION-underprogram FN(X) Detta beräknar bara funktionsvärdet i punkten x där x vanligtvis är en vektor.

Generering av slumptal finns i de flesta subrutinpaket för högnivå språk. Följande visar FORTRAN IV subrutinerna RANDU och GAUSS tagna från IBM/1130 Scientific Subroutine Package. Dessa gäller för en maskin med 16-bitars ordlängd.

```
      SUBROUTINE RANDU (IX,IY,YFL)
      IY=IX*899
      IF (IY) 5,6,6
    5 IY=IY+32767+1
    6 YFL=IY
      YFL=YFL/32767.0
      RETURN
      END
```

```
      SUBROUTINE GAUSS (IX,S,AM,V)
      A=0.0
      DO 50 I=1,12
      CALL RANDU(IX,IY,Y)
      IX=IY
50    A=A+Y
      V=(A-6.0)*S+AM
      RETURN
      END
```

Dessa rutiner är i programmet sammantagna till en subrutin kallad GAUSS. Resultatet från subrutinen är pseudoslumptal dvs. för ett visst givet startvärde IX fås alltid samma följd av slumptal. Ett rektangelfördelat tal fås alltså genom att multiplicera två stora tal med varandra. Resultatet blir då så stort att det ej kan representeras som heltal i ett ord i maskinen. Det är endast de minst signifikanta positionerna i talet som får plats i ordet. PDP-15 har 18-bitars ordlängd där biten längst till vänster är teckenbit vilket innebär att största talet som kan representeras är $2^{17} - 1 = 131071$

Exempel:

IX = 98705 = $011\ 000\ 000\ 110\ 010\ 001_2$

1795 = $000\ 000\ 011\ 100\ 000\ 011_2$

IX * 1795 = $177\ 175\ 475_{10}$ =

= $001\ 010\ 100\ 011\ 110\ 111\ 101\ 110\ 110\ 011_2$

operationen IY = IX * 1795 kommer att ge

IY = $110\ 111\ 101\ 110\ 110\ 011_2$

observera att teckenbiten är 1 dvs. vi har här ett negativt tal nämligen -33869 Detta görs positivt genom att addera 131072 därefter dividerar vi med 131071 för att få ett slumptal mellan 0 och 1. Talet 1795 som IX multipliceras med skall väljas så att man får en icke upprepande sekvens av maximal längd.

Metoden var alltså $x_{n+1} \equiv \lambda x_n \pmod{2^\beta}$ där β är antalet bitar i ett ord. Maximal period blir då $2^{\beta-2}$ och fås för $\lambda \equiv 3 \pmod 8$ eller $\lambda \equiv 5 \pmod 8$ och alla udda startvärden x_0 [5].

Vi har $\lambda = 1795 = 3 \pmod 8$ och startvärde IX är stort och udda.

För att få ett normalfördelat tal med medelvärde AM och standardavvikelse S adderar man alltså 12 rektangelfördelade tal som ligger mellan 0 och 1. Resultatet läggs i A och då fås det normalfördelade talet V som V = (A - 6.0) * S + AM.

TIME10 är en subrutin som finns i FORTRAN-biblioteket hos PDP-15

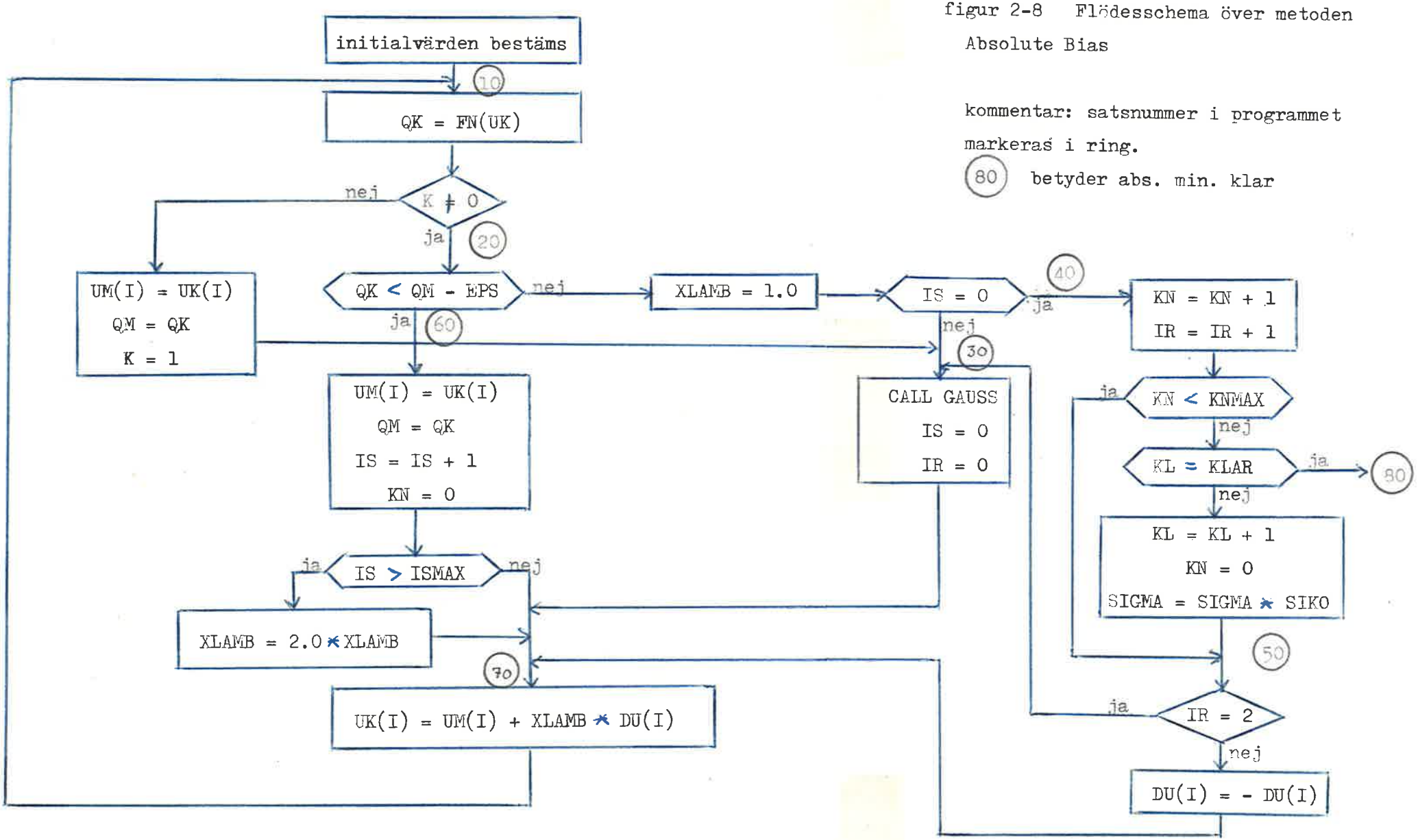
Tiden mäts från anropet tills satsen IOFF = 1 påträffas. Tiden redovisas som minst i tiondels sekunder.
Programmet som behandlar själva minimeringsprocessen finns redovisat i form av ett flödesschema figur 2-8. Några ändringar finns jämfört med figur 2-7. Betckningarna är något annorlunda. Vi har följande:

figur 2-7		figur 2-8
r	$\longleftrightarrow$	IR
s	$\longleftrightarrow$	IS
S	$\longleftrightarrow$	ISMAX
u	$\longleftrightarrow$	DU(I)

Vi avbryter inte sökningen i lyckosam riktning om IS > ISMAX. Detta indikerar ju att steget är för litet. Genom att införa XLAMB som steget i riktningen DU och göra satsen XLAMB = 2.0 * XLAMB varje gång IS > ISMAX kommer steget succesivt att fördubblas tills ett misslyckat försök uppträder. XLAMB kommer då att återställas till 1. Processen kommer att avslutas då ett visst antal mätpunkter kring ett minimivärde alla har förkastats.
Ett sätt att minska σ i närheten av minimipunkten är att utnyttja kriteriet för avslutning. I stället för att avsluta hela processen då ett visst antal punkter kring ett minimivärde förkastats så minskas σ. Detta gör att vi kan komma ännu närmare den verkliga minimipunkten. Snart inträffar återigen att kriteriet för avslutning är uppfyllt. Då kan σ minskas ytterligare osv. I programmet är högsta antalet förkastade punkter kring en minimipunkt = KNMAX.
KN är en räknare som räknar antalet misslyckade försök i följd.
KL är en räknare som räknas upp varje gång som SIGMA ändras. KLAR anger hur många olika SIGMA-värden som skall användas innan processen avslutas. SIKO är den konstant som bestämmer minskningen hos SIGMA.
För testfunktionen "Rosenbrocks curved valley" har omfattande körningar gjorts för att försöka utröna bästa värden på KNMAX, ISMAX,KLAR,SIGMA,SIKO se nästa avsnitt.
Drift kan man klara av genom att relativt ofta omstarta processen. Detta har ej gjorts i programmet. K = 0 innebär omstart eller start så om vi sätter K = 0 då $z^k < Z$ och föregående försök var misslyckat (beteckningar enligt förra avsnittet) kan vi alltså klara av drift.

figur 2-8 Flödesschema över metoden Absolute Bias

kommentar: satsnummer i programmet markeras i ring.
(80) betyder abs. min. klar



Störningar klaras av med parametern EPS som är minsta accepterade funktionsvärdesförbättringen.

2.4.3. Resultat

Metoden Absolute Bias har använts på testfunktionen.

$f(x_1,x_2) = 100 (x_2 - x_1^2)^2 + (1 - x_1)^2$

Funktionen finns uppritad i figur 2-23 på sidan 2-44.

Startpunkt (-1.2,1.0) med funktionsvärdet 24.2

Minimipunkt (1.0,1.0) med funktionsvärdet 0.0

Denna testfunktion kallas Rosenbrocks curved valley.

De parametrar som kan varieras är följande: EPS,IX,KLAR,SIGMA, SIKO,ISMAX,KNMAX. Eftersom inga störningar förekommer sättes EPS = 0.0 Med IX = 3847 och KLAR = 3 har övriga parametervärden undersökts. Först har SIGMA varierats därefter SIKO,ISMAX och KNMAX se figurerna 2-9, 2-10, 2-11, 2-12. Efterhand har parametrarna fått sina bästa värden, vilka befanns vara SIGMA = 0.2, SIKO = 0.2, ISMAX = 15, KNMAX = 40. Eftersom SIGMA = SIKO * SIGMA, bestämmer dessa två parametrar tillsammans vilka SIGMA-värden som genomlöps i algoritmen. Olika kombinationer har prövats för ISMAX = 15 och KNMAX = 40, men ingen förbättring uppnåddes, se figurerna 2-13, 2-14. En viktig fråga är hur känslig metoden är för olika IX-värden dvs. andra sekvenser av slumptal. I figur 2-15 redovisas några försök, som visar en viss försämring. Detta kan tolkas så att ovanstående parametrar i viss mening endast är optimala för IX = 3847. I figur 2-16 har därför parametrarna varierats något för ett av dessa nya IX-värden, men metoden konvergerar ej lika snabbt som för IX = 3847. Förklaringen till detta är att testfunktionen är svår för slumpsökningsmetoder eftersom det i närheten av minimipunkten är så få riktningar som leder till funktionsförbättring. Antalet riktningar som genereras innan processen avbryts kan naturligtvis ökas, men detta skulle medföra en långsammare konvergens.

Några exempel kördes även med KLAR = 4 se figurerna 2-17 och 2-18, dvs. vi tar ett varv till med ännu mindre SIGMA-värde. På detta sätt kommer vi alltså närmare minimipunkten på bekostnad av fler beräknade funktionsvärden.

2.4.4. Allmänt om parametrars inverkan

EPS anger minsta accepterade funktionsförbättringen. Även när vi testar på rent matematiska funktioner som ej är behäftade med störningar kan det vara lämpligt att ha EPS $\neq$ 0. Vi kommer visserligen att genomsöka flera olika riktningar, men i genomsnitt kommer funktionsförbättringen per steg att öka.

Ett stort värde på KLAR betyder att vi kommer nära minimipunkten ty då är SIGMA litet. SIGMA kommer att reduceras så många gånger som KLAR anger.

Valet av SIGMA beror på avståndet mellan startpunkt och minimipunkt och även på funktionens utseende. SIGMA skall väljas så att vi får en bra start.

SIKO skall sedan se till att de reducerade SIGMA-värdena ger en bra fortsättning av algoritmen. För en relativt enkel funktion bör SIKO vara litet eftersom algoritmen snabbt närmar sig minimipunkten. Välj SIKO = 0.1 eller mindre. För en knepig funktion bör vi ha SIKO något större t. ex. 0.3 - 0.5. Olika kombinationer av SIKO och SIGMA bör prövas.

Ett litet ISMAX snabbar upp systemet. Vi kommer inte att ta en mängd små steg med små funktionsförbättringar utan så snart antalet lyckade försök i följd har överskridit ISMAX kommer vi att fördubbla steglängden. Om man är mån om en lyckosam riktning kan man ha ISMAX någorlunda stort.

KNMAX är det maximala antalet riktningar som kommer att undersökas vid en given punkt. Om KNMAX är stort är det troligt att vi hittar en bra sökriktning. KNMAX bestämmer också när vi skall reducera SIGMA och när algoritmen skall avslutas. Om antalet undersökta riktningar i en punkt blir större än KNMAX kommer SIGMA att reduceras eller så slutar algoritmen. Ett stort värde på KNMAX kommer då att göra metoden långsam. Ett litet värde på KNMAX betyder att vi har små chanser att hitta en bra riktning och minimeringen blir dålig. Det optimala värdet på KNMAX ligger någonstans mittemellan. Det beror naturligtvis på funktionens utseende. Några olika värden på KNMAX bör testas. Om noggrannheten sätts före snabbhet bör KNMAX vara stort.

figur 2-9

KLAR = 3

EPS = 0.0

IX = 3847

ISMAX = 10

KNMAX = 30

SIKO = 0.2

Variation av SIGMA

NUMBER OF FUNCTION-EVALUATIONS	0.4 F(X)	0.2 F(X)	0.1 F(X)	0.05 F(X)
50	0.508E+00	0.123E+01	0.139E+01	0.150E+01
100	0.358E+00	0.915E-01	0.175E+00	0.205E+00
150	0.172E+00	0.735E-01	0.134E+00	0.160E+00
200	0.136E+00	0.605E-01	0.117E+00	0.144E+00
250	0.109E+00	0.157E-03	0.844E-01	0.119E+00
300		0.948E-04	0.744E-01	0.942E-01
350			0.651E-01	0.690E-01
400			0.466E-03	0.189E-02
450			0.368E-03	0.180E-02
500			0.325E-03	0.115E-02
550			0.237E-03	0.103E-02
600			0.191E-03	0.630E-03
650			0.157E-03	0.480E-03
700			0.110E-03	0.378E-03
750				0.258E-03
800				0.216E-03
850				0.146E-03
900				0.111E-03
950				0.755E-04
1000				0.550E-04
1050				0.352E-04

SIGMA = 0.2 verkar vara klart överlägsen. Ett lägre värde på SIGMA gör att vi kan nå en bättre minimipunkt men antalet steg blir fler. Ett högre värde på SIGMA kan medföra en bättre start men vi får dålig noggrannhet i resultatet.

figur 2-10

KLAR = 3

EPS = 0.0

IX = 3847

ISMAX = 10

KNMAX = 30

SIGMA = 0.2

Variation av SIKO

NUMBER OF FUNCTION-EVALUATIONS	0.3 F(X)	0.2 F(X)	0.1 F(X)	0.08 F(X)
50	0.123E+01	0.123E+01	0.123E+01	0.123E+01
100	0.915E-01	0.915E-01	0.915E-01	0.915E-01
150	0.660E-01	0.735E-01	0.735E-01	0.519E-01
200	0.572E-01	0.605E-01	0.715E-01	0.498E-01
250	0.185E-03	0.157E-03	0.645E-04	0.259E-03
300		0.948E-04	0.513E-04	0.200E-03
350				0.161E-03
400				0.111E-03
450				0.928E-04
500				0.580E-04
550				0.430E-04
600				0.291E-04
650				0.210E-04
700				0.131E-04

SIKO = 0.1 verkar vara bäst, men SIKO = 0.2 låg lite före vid 200 funktionsberäkningar. Förbättringarna vid 250 f. b. är enorm, man kan anta att en riktning blev en fullträff. Vad man egentligen önskar sig är ett värde på SIKO som bär sig hyfsat åt för det mesta. Om vi tittar på 200 f. b. för alla 4 exemplen är skillnaderna ganska små. SIKO = 0.2 väljs som det optimala värdet.

figur 2-11

KLAR = 3

EPS = 0.0

IX = 3847

KNMAX = 30

SIGMA = 0.2

SIKO = 0.2

Variation av ISMAX

NUMBER OF FUNCTION-EVALUATIONS	5 F(X)	10 F(X)	15 F(X)
50	0.123E+01	0.123E+01	0.123E+01
100	0.915E-01	0.915E-01	0.915E-01
150	0.735E-01	0.735E-01	0.735E-01
200	0.605E-01	0.605E-01	0.605E-01
250	0.644E-03	0.157E-03	0.157E-03
300	0.307E-03	0.948E-04	0.948E-04
350	0.111E-03		

När antalet lyckosamma försök i följd blir större än ISMAX så kommer steget att fördubblas vid varje nytt försök tills vi får ett misslyckat försök. Av ovanstående experiment finner vi att antalet lyckosamma försök i någon riktning varit mindre än 10 ty de två sista experimenten är identiska. Då funktionen som undersöks är svår, vill (man) ta små steg i en lyckosam riktning, men ett för stort värde på ISMAX ger långsam konvergens. Vi väljer ISMAX = 15.

figur 2-12

KLAR = 3

EPS = 0.0

IX = 3847

ISMAX = 15

SIGMA = 0.2

SIKO = 0.2

Variation av KNMAX

NUMBER OF FUNCTION-EVALUATIONS	20 F(X)	30 F(X)	40 F(X)	50 F(X)
50	0.123E+01	0.123E+01	0.123E+01	0.123E+01
100	0.915E-01	0.915E-01	0.915E-01	0.915E-01
150	0.705E-01	0.735E-01	0.735E-01	0.915E-01
200		0.605E-01	0.605E-01	0.776E-01
250		0.157E-03	0.157E-03	0.548E-01
300		0.948E-04	0.948E-04	0.400E-01
350			0.193E-04	0.104E-02
400				0.804E-03
450				0.520E-03
500				0.232E-03
550				0.175E-03
600				0.438E-04

Storleken på KNMAX avgör när SIGMA skall reduceras och därmed också när metoden terminerar. Stort värde på KNMAX betyder att många riktningar undersöks. Detta ger bättre noggrannhet i resultatet men metoden tar längre tid. Av ovanstående experiment finner vi att ett bra värde är KNMAX = 40.

figur 2-13

KLAR = 3

EPS = 0.0

IX = 3847

ISMAX = 15

KNMAX = 40

SIGMA = 0,1

Variation av SIKO

NUMBER OF FUNCTION-EVALUATIONS	0.1 F(X)	0.2 F(X)	0.3 F(X)
50	0.139E+01	0.139E+01	0.139E+01
100	0.134E+00	0.134E+00	0.134E+00
150	0.120E+00	0.113E+00	0.114E+00
200	0.107E+00	0.102E+00	0.894E-01
250	0.625E-01	0.796E-01	0.600E-01
300	0.585E-01	0.688E-01	0.287E-01
350	0.798E-02	0.487E-01	0.542E-03
400	0.370E-02	0.144E-02	0.446E-03
450	0.225E-02	0.473E-03	0.291E-03
500	0.157E-02	0.185E-03	0.805E-04
550	0.759E-03	0.152E-03	
600	0.572E-03		
650	0.507E-03		
700	0.175E-03		
750	0.167E-03		
800	0.130E-03		
850	0.949E-04		
900	0.857E-04		
950	0.559E-04		
1000	0.407E-04		
1050	0.289E-04		
1100	0.174E-04		
1150	0.129E-04		
1200	0.836E-05		
1250	0.431E-05		
1300	0.294E-05		

Då det nya SIGMA-värdet fås genom formeln

SIGMA = SIGMA * SIKO ser vi att SIGMA och SIKO kanske inte bör optimeras var för sig. I denna figur och figur 2-14 undersöks några kombinationer av SIGMA och SIKO.

figur 2-14

KLAR = 3

EPS = 0.0

IX = 3847

ISMAX = 15

KNMAX = 40

Variation av SIGMA och SIKO

	SIGMA = 0.3			SIGMA = 0.4
	SIKO			SIKO = 0.1
	0.1	0.2	0.3	
NUMBER OF FUNCTION-EVALUATIONS	F(X)	F(X)	F(X)	F(X)
50	0.384E+00	0.384E+00	0.384E+00	0.508E+00
100	0.260E+00	0.260E+00	0.260E+00	0.358E+00
150	0.167E+00	0.189E+00	0.158E+00	0.197E+00
200	0.153E+00	0.170E+00	0.158E+00	0.192E+00
250	0.124E+00	0.137E+00	0.148E+00	0.117E+00
300	0.961E-01	0.897E-01	0.113E+00	0.963E-01
350	0.906E-01	0.852E-01	0.102E+00	0.943E-01
400	0.617E-01	0.454E-01	0.807E-01	0.745E-01
450	0.561E-03	0.432E-01	0.515E-01	0.980E-04
500	0.198E-04	0.434E-02	0.355E-01	0.127E-04
550	0.118E-04	0.404E-02	0.891E-03	0.217E-05
600	0.105E-04	0.243E-02		

Inget av experimenten redovisade i denna figur eller i figur 2-13 ger någon förbättring jämfört med tidigare funna optimala värden på SIGMA och SIKO.

figur 2-15

KLAR = 3

EPS = 0.0

ISMAX = 15

KNMAX = 40

SIGMA = 0.2

SIKO = 0.2

	Variation av IX			
	13421	6113	4711	9817
NUMBER OF FUNCTION-EVALUATIONS	F(X)	F(X)	F(X)	F(X)
50	0.339E+00	0.102E+01	0.106E+01	0.287E+00
100	0.292E+00	0.691E+00	0.499E-01	0.205E+00
150	0.110E+00	0.635E+00	0.223E-01	0.792E-01
200	0.107E+00	0.583E+00	0.212E-01	0.760E-01
250	0.889E-01	0.535E+00	0.187E-01	0.684E-01
300	0.799E-01	0.363E+00		0.501E-01
350	0.105E-01	0.349E+00		
400	0.450E-02	0.339E+00		
450	0.213E-02	0.305E+00		
500	0.184E-02	0.284E+00		
550	0.120E-02	0.276E+00		
600	0.953E-03	0.234E+00		
650	0.646E-03	0.198E+00		
700	0.320E-03	0.168E+00		
750	0.220E-03	0.141E+00		
800	0.864E-04	0.107E+00		
850		0.127E-01		
900		0.571E-02		
950		0.380E-02		
1000		0.264E-02		
1050		0.185E-02		
1100		0.130E-02		

Sekvensen av genererade slumptal verkar ha stor betydelse för resultatet. Vilket betyder att de tidigare funna optimala parametrarna i viss mening endast är optimala för IX = 3847 Uppförandet hos metoden beror på att funktionen som undersöks är väldigt besvärlig för en slumpsökningsmetod. I närheten av minimat finns endast ett fåtal riktningar som ger funktionsförbättring.

figur 2-16 IX = 9817 ISMAX = 15

EPS = 0.0 KNMAX = 60

KLAR	3	3	3	3	3	4
SIGMA	0.1	0.1	0.1	0.2	0.4	0.2
SIKO	0.1	0.2	0.3	0.2	0.2	0.2

NUMBER OF FUNCTION-EVALUATIONS	F(X)	F(X)	F(X)	F(X)	F(X)	F(X)
50	0.543E+00	0.543E+00	0.543E+00	0.287E+00	0.747E+00	0.287E+00
100	0.138E+00	0.138E+00	0.138E+00	0.205E+00	0.324E+00	0.205E+00
150	0.131E+00	0.131E+00	0.131E+00	0.664E-01	0.909E-01	0.664E-01
200	0.436E-01	0.436E-01	0.436E-01	0.632E-01	0.843E-01	0.632E-01
250	0.293E-01	0.299E-01	0.253E-01	0.577E-01	0.381E-01	0.577E-01
300	0.233E-01	0.278E-01	0.252E-01	0.354E-01	0.338E-01	0.354E-01
350	0.220E-01	0.273E-01	0.233E-01	0.354E-01	0.329E-01	0.354E-01
400	0.179E-01	0.238E-01	0.205E-01	0.294E-01	0.316E-01	0.294E-01
450	0.144E-01	0.213E-01	0.175E-01	0.276E-01	0.302E-01	0.276E-01
500	0.140E-01	0.209E-01	0.166E-01	0.255E-01	0.286E-01	0.255E-01
550	0.132E-01	0.179E-01	0.134E-01	0.230E-01	0.286E-01	0.230E-01
600	0.114E-01	0.158E-01	0.118E-01	0.190E-01	0.233E-01	0.190E-01
650	0.114E-01	0.155E-01	0.117E-01	0.156E-01	0.189E-01	0.156E-01
700	0.109E-01	0.140E-01	0.102E-01	0.149E-01	0.183E-01	0.149E-01
750	0.996E-02	0.128E-01	0.877E-02	0.128E-01	0.152E-01	0.128E-01
800	0.959E-02	0.121E-01	0.745E-02	0.116E-01		0.116E-01
850	0.893E-02	0.111E-01	0.731E-02	0.114E-01		0.114E-01
900	0.845E-02	0.106E-01	0.626E-02	0.102E-01		0.102E-01
950	0.827E-02	0.888E-02	0.525E-02	0.889E-02		0.889E-02
1000	0.760E-02	0.841E-02	0.478E-02	0.870E-02		0.870E-02
1050	0.737E-02	0.808E-02	0.443E-02	0.770E-02		0.770E-02
1100	0.702E-02	0.751E-02	0.371E-02	0.663E-02		0.663E-02
1150	0.647E-02	0.721E-02		0.568E-02		0.568E-02
1200	0.634E-02	0.620E-02		0.520E-02		0.520E-02
1250	0.604E-02	0.510E-02		0.488E-02		0.488E-02
1300	0.577E-02	0.486E-02		0.423E-02		0.423E-02
1350	0.565E-02	0.456E-02				0.400E-02
1400	0.546E-02	0.416E-02				0.375E-02
1450	0.516E-02	0.393E-02				0.342E-02
1500	0.487E-02	0.377E-02				0.318E-02
1550	0.469E-02	0.320E-02				0.297E-02
1600	0.445E-02	0.288E-02				0.271E-02
1650	0.436E-02	0.269E-02				0.251E-02
1700	0.416E-02	0.258E-02				0.239E-02
1750	0.382E-02	0.256E-02				0.213E-02
1800	0.367E-02	0.242E-02				0.193E-02
1850	0.354E-02	0.216E-02				0.178E-02
1900	0.325E-02	0.192E-02				0.164E-02
1950	0.310E-02	0.173E-02				0.152E-02
:	:					:
3100	0.492E-03					0.232E-03
3150	0.461E-03					0.220E-03
:	:					
4100	0.108E-03					
4150	0.107E-03					

Försök till ny optimering med IX = 9817

Resultatet är tämligen dåligt.

figur 2-17

EPS = 0.0

IX = 3847

ISMAX = 15

KNMAX = 40

KLAR = 4

SIGMA = 0.1

Variation av SIKO

NUMBER OF FUNCTION-EVALUATIONS	0.1 F(X)	0.2 F(X)	0.4 F(X)	0.6 F(X)
50	0.140E+01	0.140E+01	0.140E+01	0.140E+01
100	0.134E+00	0.134E+00	0.134E+00	0.134E+00
150	0.120E+00	0.113E+00	0.109E+00	0.102E+00
200	0.106E+00	0.102E+00	0.927E-01	0.102E+00
250	0.625E-01	0.795E-01	0.582E-01	0.802E-01
300	0.585E-01	0.687E-01	0.407E-01	0.546E-01
350	0.789E-02	0.486E-01	0.249E-02	0.533E-01
400	0.367E-02	0.148E-02	0.188E-02	0.236E-03
450	0.224E-02	0.461E-03	0.108E-02	0.236E-03
500	0.206E-02	0.207E-03	0.311E-03	
550	0.139E-02	0.155E-03	0.211E-03	
600	0.113E-02	0.902E-04	0.125E-03	
650	0.859E-03	0.378E-04	0.244E-04	
700	0.741E-03	0.221E-04		
750	0.525E-03	0.202E-04		
800	0.452E-03	0.148E-04		
850	0.325E-03	0.103E-04		
900	0.273E-03	0.545E-05		
950	0.187E-03	0.410E-05		
1000	0.170E-03	0.196E-05		
1050	0.104E-03			
1100	0.722E-04			
1150	0.652E-04			
1200	0.486E-04			
1250	0.428E-04			
1300	0.333E-04			
1350	0.227E-04			
:	:			
:	:			
:	:			
2250	0.511E-07			
2300	0.430E-07			

När KLAR ökas en enhet betyder det att i stället för att avsluta, så reduceras SIGMA ytterligare en gång. På detta sättet kommer vi att komma närmare minimipunkten.

figur 2-18

EPS = 0.0

IX = 3847

ISMAX = 15

KNMAX = 40

KLAR = 4

SIGMA = 0.3

Variation av SIKO

NUMBER OF FUNCTION-EVALUATIONS	0.1 F(X)	0.2 F(X)	0.4 F(X)	0.6 F(X)
50	0.384E+00	0.384E+00	0.384E+00	0.384E+00
100	0.260E+00	0.260E+00	0.260E+00	0.260E+00
150	0.167E+00	0.189E+00	0.211E+00	0.194E+00
200	0.153E+00	0.169E+00	0.201E+00	0.179E+00
250	0.124E+00	0.137E+00	0.175E+00	0.158E+00
300	0.961E-01	0.897E-01	0.127E+00	0.960E-01
350	0.906E-01	0.852E-01	0.102E+00	0.670E-01
400	0.617E-01	0.454E-01	0.993E-01	0.626E-01
450	0.561E-03	0.432E-01	0.756E-01	
500	0.198E-04	0.432E-02	0.443E-01	
550	0.118E-04	0.402E-02	0.426E-01	
600	0.105E-04	0.241E-02	0.267E-02	
650	0.864E-05	0.151E-02	0.260E-02	
700	0.688E-05	0.115E-02	0.194E-02	
750	0.617E-05	0.762E-03	0.782E-03	
800	0.488E-05	0.705E-03		
850	0.356E-05	0.466E-03		
900	0.238E-05	0.374E-03		
950	0.142E-05	0.240E-03		
1000	0.102E-05	0.175E-03		
1050	0.648E-06	0.132E-03		
1100	0.229E-06	0.102E-03		
1150	0.202E-06	0.736E-04		
1200		0.506E-04		
1250		0.365E-04		
1300		0.288E-04		
1350		0.114E-04		

Vi har endast ändrat SIGMA till 0.3 jämfört med figur 2-18.

2.5 Jämförelse Absolute Bias - Fletcher-Reeves

Absolute Bias var klart lättast att programmera men bestod av ett flertal variabler vars inverkan på resultatet var ganska stor. Det behövdes omfattande testkörningar innan resultaten blev så fina att de kunde jämföras med de resultat som Fletcher-Reeves gav efter några få tester. I noggrannhet kan de bägge metoderna fås att komma godtyckligt nära minimipunkten, men det tar avsevärt längre tid för Absolute Bias. Om störningar inverkar kan emellertid Absolute Bias vara att föredra eftersom den inte använder sig av gammal information i så hög grad som Fletcher-Reeves.
Det bästa resultatet för Fletcher-Reeves metod var minimivärdet $0.215 \cdot 10^{-5}$ efter 0.3 sekunder. I figur 2-19 redovisas denna körning med varje iterations minimipunkt utskriven. Dessa punkter finns sedan plottade i figur 2-20 för att åskådliggöra hur metoden arbetar sig framåt. I figuren ser vi att metoden kan stå och stampa på ungefär samma fläck ett antal gånger innan en riktning som ger avsevärd funktionsvärdesförbättring hittas. Detta beror delvis på att metoden startas om efter n+1 iterationer.
För metoden Absolute Bias har vi tre körningar som ger ganska bra resultat, dessa tre finns redovisade i figur 2-21 och 2-22. Den snabbaste varianten tar 1.2 sekunder och når minimivärdet $0.209 \cdot 10^{-4}$. De andra körningarna når ett bättre minimivärde men på avsevärt längre tid. I figur 2-23 har vart 50:e minimivärde plottats för körning nummer 2. Vi ser där att metoden redan efter 100 funktionsberäkningar befinner sig ganska nära minimipunkten, men sedan när i stort sett bara en riktning leder till funktionsvärdesförbättring så blir metoden betydligt långsammare. Som jämförelse mellan de båda metoderna finner vi att Fletcher-Reeves metod är avsevärt snabbare än metoden Absolute Bias.
En annan funktion testades också nämligen

$$f(x_1,x_2,x_3) = 0.5 \cdot x_1^2 + x_2^2 + 2 \cdot x_3^2 + 5 \cdot x_1^2 x_2^2$$

med startpunkt (1.0, 1.0, 1.0), funktionsvärde 8.5 och minimipunkt i origo med funktionsvärde 0.0. Med Fletcher-Reeves metod gjordes fyra experiment se figur 2-24.I de två första beräknades derivatan med hjälp av framåtdifferenser, dvs. subrutin NUDER1 användes. För H = 0.05 blev resultatet ganska dåligt medan $H = 0.1 \cdot 10^{-4}$ gav utmärkt resultat. Då derivatan beräknades med centrala differenser

blev resultatet utmärkt även för H = 0.05. Vi ser här att valet av H och valet av beräkningsmetod har stor betydelse om funktionen ej är väldigt enkel. Det bästa resultatet var minimivärdet $0.417 \cdot 10^{-7}$ efter 0.1 sekunder. Med metoden Absolute Bias gjorde vi först några körningar där SIGMA och SIKO varierades, se figur 2-25, därefter försökte vi minska beräkningstiden genom att minska KNMAX och ISMAX, se figur 2-26. Tiden minskades från 1.4 s till 0.9 s och minimivärdet var $0.408\ 10^{-5}$.

Även för denna testfunktion var Fletcher-Reeves metod avsevärt bättre. Allmänt gäller att slumpsökningsmetoder fungerar sämre då antalet variabler ökar.

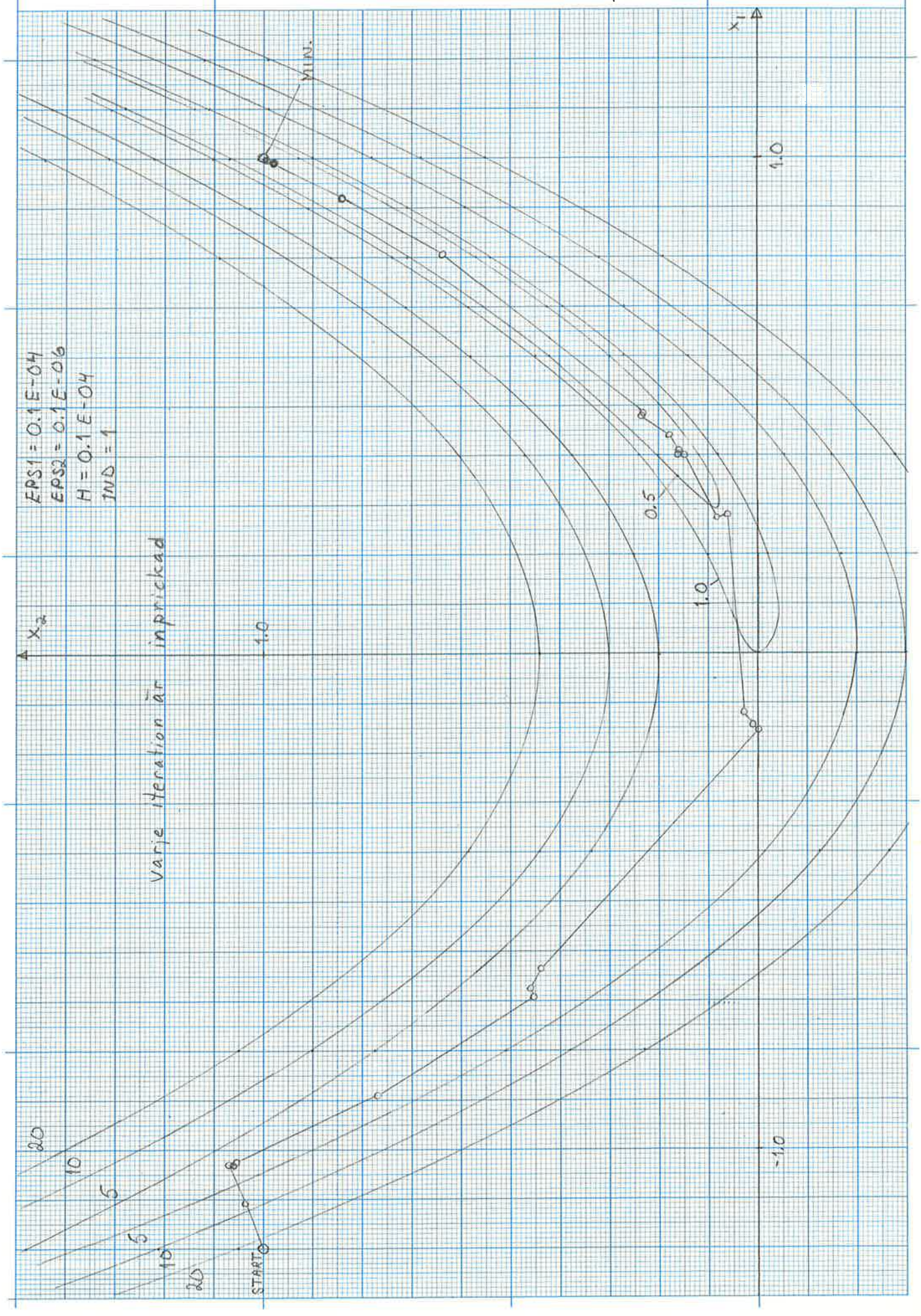
```
PARAMETERS
EPS1= 0.100E-04    EPS2= 0.100E-06
H= 0.10E-04    IND= 1

   F(X)          X(I)

 0.763E+01    -1.104   1.039
 0.413E+01    -1.030   1.071
 0.412E+01    -1.030   1.061
 0.411E+01    -1.025   1.061
 0.363E+01    -0.891   0.769
 0.291E+01    -0.692   0.456
 0.281E+01    -0.676   0.463
 0.280E+01    -0.635   0.439
 0.142E+01    -0.155  -0.005
 0.131E+01    -0.143   0.012
 0.128E+01    -0.118   0.032
 0.544E+00     0.280   0.062
 0.530E+00     0.277   0.086
 0.374E+00     0.399   0.148
 0.358E+00     0.402   0.163
 0.354E+00     0.413   0.161
 0.336E+00     0.441   0.179
 0.273E+00     0.479   0.234
 0.267E+00     0.485   0.231
 0.592E-01     0.807   0.637
 0.138E-01     0.921   0.840
 0.677E-02     0.918   0.842
 0.672E-02     0.918   0.843
 0.196E-02     0.992   0.979
 0.238E-03     0.990   0.981
 0.973E-04     0.990   0.980
 0.971E-04     0.990   0.980
 0.968E-04     0.990   0.980
 0.966E-04     0.990   0.980
 0.964E-04     0.990   0.980
 0.962E-04     0.990   0.980
 0.961E-04     0.990   0.980
 0.959E-04     0.990   0.981
 0.956E-04     0.990   0.981
 0.942E-04     0.990   0.981
 0.900E-04     0.991   0.981
 0.898E-04     0.991   0.981
 0.132E-04     0.998   0.997
 0.215E-05     0.999   0.997


NUMBER OF ITERATIONS 40
TIME IN SECONDS 0.3
POINT OF MINIMUM
        0.999
        0.997
WITH THE FUNCTIONVALUE 0.215E-05
```

figur 2-20

Nivåkurvor för Rosenbrocks curved valley

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

Fletcher - Reeves



figur 2-21

```
ISMAX=15   KNMAX=40   KLAR= 3   SIKO=0.20
SIGMA=0.200   EPS=0.00   IX= 3847
```

NUMBER OF FUNCTION-EVALUATIONS	F(X)	X(I)	
50	0.123E+01	-0.078	-0.020
100	0.915E-01	0.698	0.489
150	0.735E-01	0.729	0.531
200	0.604E-01	0.775	0.610
250	0.166E-03	1.011	1.022
300	0.880E-04	1.007	1.015
350	0.209E-04	1.004	1.009

```
NUMBER OF CALCULATED FUNCTIONVALUES  353
TIME IN SECONDS 1.2
POINT OF MINIMUM
        1.004
        1.009
WITH THE FUNCTIONVALUE 0.209E-04
```

```
ISMAX=15   KNMAX=40   KLAR= 4   SIKO=0.20
SIGMA=0.100   EPS=0.00   IX= 3847
```

NUMBER OF FUNCTION-EVALUATIONS	F(X)	X(I)	
50	0.140E+01	-0.166	0.046
100	0.134E+00	0.634	0.403
150	0.113E+00	0.665	0.445
200	0.102E+00	0.683	0.470
250	0.795E-01	0.718	0.515
300	0.687E-01	0.753	0.576
350	0.486E-01	0.781	0.612
400	0.148E-02	0.970	0.939
450	0.461E-03	0.979	0.959
500	0.207E-03	0.986	0.973
550	0.155E-03	0.988	0.977
600	0.902E-04	0.991	0.983
650	0.378E-04	0.994	0.988
700	0.221E-04	0.995	0.991
750	0.202E-04	0.996	0.992
800	0.148E-04	0.996	0.992
850	0.103E-04	0.997	0.994
900	0.545E-05	0.998	0.996
950	0.410E-05	0.998	0.996
1000	0.196E-05	0.999	0.997

```
NUMBER OF CALCULATED FUNCTIONVALUES 1028
TIME IN SECONDS 3.2
POINT OF MINIMUM
        0.999
        0.997
WITH THE FUNCTIONVALUE 0.196E-05
```

figur 2-22

```
ISMAX=15   KNMAX=40   KLAR= 4   SIKO=0.10
SIGMA=0.300   EPS=0.00   IX= 3847
```

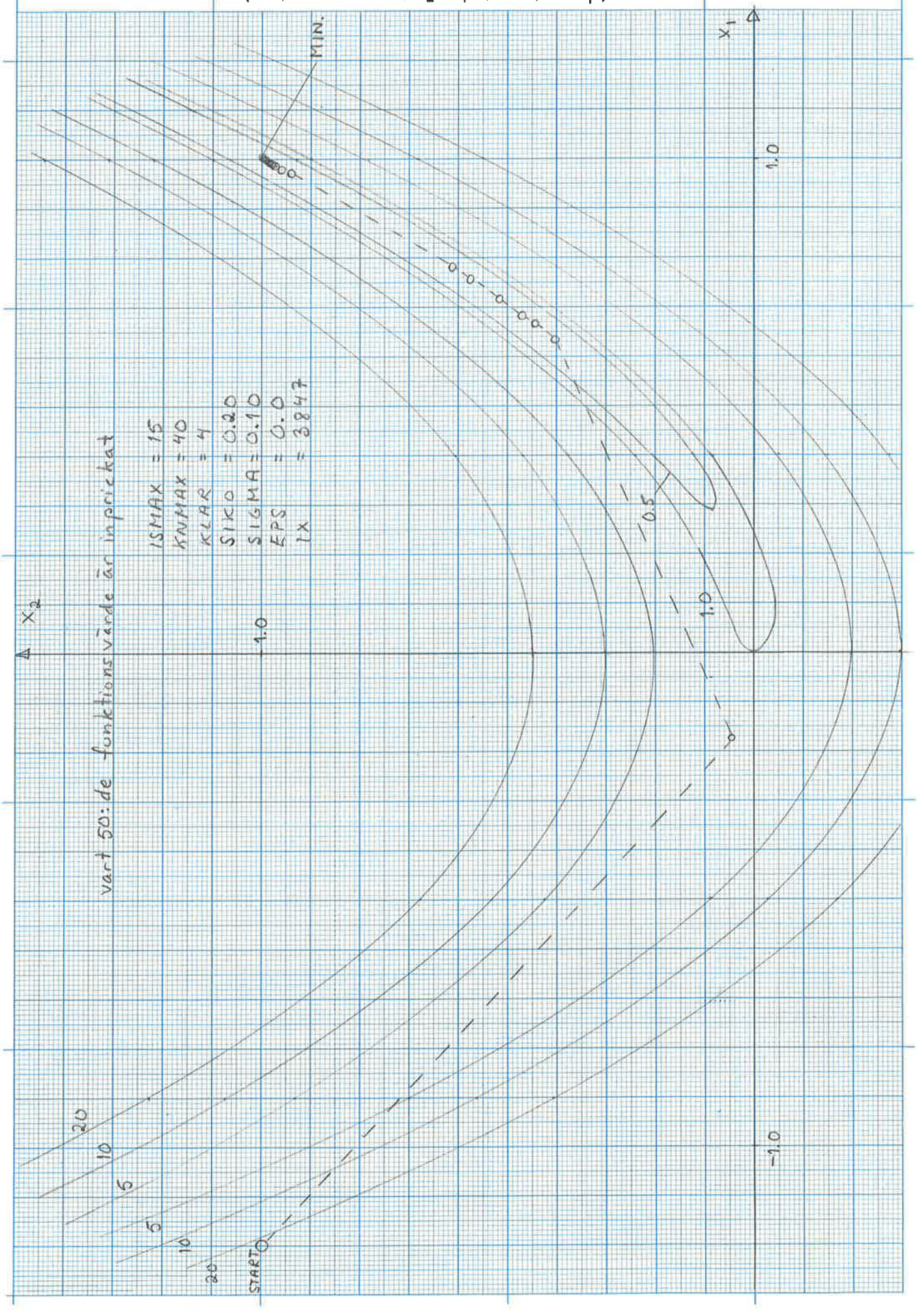
NUMBER OF FUNCTION-EVALUATIONS	F(X)	X(I)	
50	0.384E+00	0.422	0.200
100	0.260E+00	0.496	0.238
150	0.167E+00	0.600	0.351
200	0.153E+00	0.609	0.371
250	0.124E+00	0.652	0.430
300	0.961E-01	0.706	0.489
350	0.906E-01	0.699	0.490
400	0.617E-01	0.752	0.568
450	0.561E-03	0.989	0.976
500	0.198E-04	0.996	0.992
550	0.118E-04	0.997	0.994
600	0.105E-04	0.997	0.994
650	0.864E-05	0.997	0.994
700	0.688E-05	0.997	0.995
750	0.617E-05	0.998	0.995
800	0.488E-05	0.998	0.996
850	0.356E-05	0.998	0.996
900	0.238E-05	0.998	0.997
950	0.142E-05	0.999	0.998
1000	0.102E-05	0.999	0.998
1050	0.648E-06	0.999	0.999
1100	0.229E-06	1.000	0.999
1150	0.202E-06	1.000	0.999

```
NUMBER OF CALCULATED FUNCTIONVALUES 1170
TIME IN SECONDS 3.7
POINT OF MINIMUM
        1.000
        0.999
WITH THE FUNCTIONVALUE 0.202E-06
```

figur 2-23

Nivåkurvor för
Rosenbrocks curved valley
$f(x_1,x_2) = 100\,(x_2 - x_1^2)^2 + (1-x_1)^2$

Absolute Bias



figur 2-24

Beräkning av derivatan med framåtdifferenser

```
PARAMETERS
EPS1= 0.100E-04    EPS2= 0.100E-06
H= 0.50E-01    IND= 0

NUMBER OF ITERATIONS  6
TIME IN SECONDS 0.1
POINT OF MINIMUM
        -0.047
         0.010
        -0.009
WITH THE FUNCTIONVALUE 0.139E-02
```

```
PARAMETERS
EPS1= 0.100E-04    EPS2= 0.100E-06
H= 0.10E-04    IND= 0

NUMBER OF ITERATIONS 13
TIME IN SECONDS 0.1
POINT OF MINIMUM
        -0.000
        -0.000
         0.000
WITH THE FUNCTIONVALUE 0.641E-07
```

Beräkning av derivatan med centrala differenser

```
PARAMETERS
EPS1= 0.100E-04    EPS2= 0.100E-06
H= 0.50E-01    IND= 0

NUMBER OF ITERATIONS 13
TIME IN SECONDS 0.1
POINT OF MINIMUM
        -0.000
        -0.000
         0.000
WITH THE FUNCTIONVALUE 0.417E-07
```

```
PARAMETERS
EPS1= 0.100E-04    EPS2= 0.100E-06
H= 0.10E-04    IND= 0

NUMBER OF ITERATIONS 13
TIME IN SECONDS 0.1
POINT OF MINIMUM
        -0.000
        -0.000
         0.000
WITH THE FUNCTIONVALUE 0.418E-07
```

figur 2-25

```
ISMAX=15   KNMAX=40   KLAR= 3   SIKO=0.20
SIGMA=0.200   EPS=0.00   IX= 3847

NUMBER OF CALCULATED FUNCTIONVALUES  341
TIME IN SECONDS 1.5
POINT OF MINIMUM
       -0.002
        0.000
       -0.001
WITH THE FUNCTIONVALUE 0.474E-05



ISMAX=15   KNMAX=40   KLAR= 3   SIKO=0.10
SIGMA=0.500   EPS=0.00   IX= 3847

NUMBER OF CALCULATED FUNCTIONVALUES  339
TIME IN SECONDS 1.5
POINT OF MINIMUM
       -0.001
        0.000
       -0.000
WITH THE FUNCTIONVALUE 0.908E-06



ISMAX=15   KNMAX=40   KLAR= 3   SIKO=0.10
SIGMA=0.300   EPS=0.00   IX= 3847

NUMBER OF CALCULATED FUNCTIONVALUES  305
TIME IN SECONDS 1.4
POINT OF MINIMUM
        0.001
        0.001
        0.000
WITH THE FUNCTIONVALUE 0.186E-05
```

figur 2-26

```
ISMAX= 5    KNMAX=20    KLAR= 3    SIKO=0.10
SIGMA=0.300    EPS=0.00    IX= 3847


NUMBER OF CALCULATED FUNCTIONVALUES  207
TIME IN SECONDS 0.9
POINT OF MINIMUM
         0.000
        -0.002
        -0.000
WITH THE FUNCTIONVALUE 0.408E-05



ISMAX= 5    KNMAX=10    KLAR= 3    SIKO=0.10
SIGMA=0.300    EPS=0.00    IX= 3847


NUMBER OF CALCULATED FUNCTIONVALUES  177
TIME IN SECONDS 0.8
POINT OF MINIMUM
        -0.005
        -0.003
        -0.001
WITH THE FUNCTIONVALUE 0.242E-04
```

KAPITEL 3. OLIKA VÄGAR ATT NÅ MASKINKOD FÖR INTEL 8080

3.1 Inledning

Intel 8080 är en mikrodator vars CPU först beskrivs i korta drag. Därefter behandlas programmeringsspråken Assembler, PL/M och HILP. Till slut görs en jämförelse mellan PL/M och HILP m.a.p. det antal ord som genereras för några givna programexempel.

3.2 Intel 8080 CPU [8]

Registerdelen av 8080 CPU består av sex 16-bitars register:

. Programräknaren (PC)
. Stackpekaren (SP)
. Sex 8-bitars allmänna register ordnade i par kallade B,C; D,E; H,L.
. Temporärt registerpar kallat W,Z

Programräknaren innehåller adressen i minnet till den aktuella instruktionen och inkrementeras automatiskt vid varje hämtning av ny instruktion.
Stackpekaren håller reda på var någonstans i läs-skriv minnet som översta elementet i stacken ligger. Stacken styrs av instruktionerna PUSH (stoppa in i stacken) och POP (hämta ut ur stacken) Stacken växer nedåt på så sätt att SP dekrementeras vid PUSH och inkrementeras vid POP.
De sex allmänna registerna kan användas antingen som enkla register (8 bitar) eller som registerpar (16 bitar).
Det temporära registerparet W,Z används internt och kan ej nås av användaren.

Den aritmetiska, logiska enheten (ALU) består av följande register:

. 8-bitars ackumulator (ACC)
. 8-bitars temporär ackumulator (ACT)

. 5-bitars flagg-register
. 8-bitars temporärt register (TMP)

De fem flaggorna är: zero, carry, sign, parity och auxiliary flag. När någon av flaggorna sätts, kommer motsvarande bit i flagg-registret att bli 1.
Flaggorna har följande funktion:

Zero: Om resultatet av en instruktion har värdet 0 sätts flaggan annars nollställs den.

Carry: Om instruktionen resulterar i en carry (från addition) eller en borrow (från subtraktion eller jämförelse) ut från den högsta biten, så sätts flaggan annars nollställs den.

Sign: Om den mest signifikanta biten av resultatet av en operation har värdet 1, kommer flaggan att sättas annars nollställs den.

Parity: Om modulo 2 summan av bitarna av resultatet av en operation är 0, så sätts flaggan annars nollställs den.

Auxiliary carry: Om resultatet av en instruktion ger en carry från bit 3 till bit 4 så sätts flaggan, annars nollställs den.

Användaren kan testa på innehållet i flagg-registret samt använda ACC på ungefär samma sätt som de allmänna registerna. Däremot kan han ej komma åt ACT eller TMP.

3.3.1 Assembler för 8080 [8]

Assembler-instruktionerna för 8080 är uppdelade på 5 grupper:

. Överföring av data - flytta data mellan register eller mellan register och minne.
. Aritmetik - addition, subtraktion, inkrementera eller dekrementera innehållet i register eller minne.
. Logik - AND, OR, EXCLUSIVE-OR, jämförelse, skift eller komplement av innehåll i register eller i minne.
. Hopp - villkorliga eller ovillkorliga hopp, subrutinanrop och return instruktioner.
. Stack, I/O och Maskinkontroll - inkluderar I/O instruktioner samt instruktioner för behandling av stacken och interna kontrollflaggor.

Varje assembler-instruktion översätts till en maskininstruktion som i sin tur kan innehålla 1, 2 eller 3 8-bitars ord.
Det finns fyra olika sätt att adressera data lagrade i minnet eller i adressen.

. Direkt - ord 2 och 3 i instruktionen innehåller adressen till minnet.
. Register - instruktionen specificerar registret eller registerparet där data är lagrat.
. Register indirekt -instruktionen specificerar ett registerpar som i sin tur innehåller adressen till minnet.
. Data direkt - instruktionen innehåller data själv, i ord 2 eller i ord 2 och 3.

Register som kan användas är A, B, C, D, E, H, L. Registerpar betecknas B för B,C; D för D,E samt H för H,L.
Register A är ackumulatorn och används vid beräkning av aritmetiska eller logiska uttryck. Registerparet H dvs. H,L, används för att adressera minnet, exempel: MOV B,M betyder innehållet i minnet vars adress finns i register H och L flyttas till register B.
H innehåller den höga adressen (bank-adressen) och L innehåller den låga adressen (adressen i en bank). Registerna B, C, D, E kan användas som vanliga variabler, ty det är endast användaren som kan ändra innehållet i dessa.

3.3.2 Synpunkter på programmering i 8080 Assembler

Vill man ha en så effektiv kod som möjligt skall man programmera i assembler, eftersom man då direkt bestämmer vilka operationer som skall göras och vet hur lång tid dessa tar. Att skriva assemblerprogram är svårt och tar lång tid, det blir därför ganska dyra programmeringskostnader. Programmen blir också svåra att läsa för en utomstående, om de inte innehåller omfattande kommentarer. Om man har ett stort program och har gott om utrymme i mikrodatorn bör man nog överväga att programmera i ett högnivåspråk.

3.4.1 PL/M högnivåspråk för 8008 och 8080 [6]

PL/M är ett högnivåspråk utvecklat av Intel för att förenkla programmeringsarbetet för sina egna 8-bitars mikrodatorer 8008 och

8080.
Ett PL/M program består av deklarationer och därefter exekverbara satser. Språket har blockstruktur dvs. deklarerade procedurer kan i sin tur innehålla nya deklarationer. PL/M kan hantera data i två former BYTE eller ADDRESS. En BYTE variabel eller konstant består av 8 bitar. En ADDRESS variabel eller konstant består av 16 bitar. Vektorer kan också deklareras antingen som BYTE eller ADDRESS dess index går från 0 till n-1. Numeriska konstanter kan skrivas i binär, oktal, decimal eller hexadecimal kod. Koden indikeras med en bokstav efter siffrorna nämligen B, O, D eller H. D kan utelämnas. Talen 10,11,12,13,14,15 motsvaras av bokstäverna A,B,C,D,E,F i hexadecimal kod. För numeriska konstanter gäller dock att första tecknet är en siffra.
Aritmetiska operatorer: + - PLUS MINUS * / MOD
Logiska operatorer: NOT AND OR XOR
Relationsoperatorer: < <= > >= = <>
Exempel på program finns i appendix C. För närmare studium av PL/M hänvisas till referens 6.

3.4.2 Synpunkter på programmering i PL/M

PL/M är ett verkligt högnivåspråk som är lätt att lära för den som tidigare kan t. ex. ALGOL. Den assemblerkod som genereras är långt ifrån optimal. Exempel har visat att utrymmet för lagring av ett PL/M program är mellan 2 och 3 gånger större än för motsvarande assemblerprogram [1] . Detta beror på att instruktioner genereras efter fixa regler av kompilatorn, vilket gör att onödiga instruktioner ofta genereras. I PL/M finns inga möjligheter att optimera koden genom att skriva om vissa avsnitt på en lägre nivå med t. ex. registeroperationer. Det verkar som om Intel tänkt sig att användaren inte är intresserad av vilka assemblerinstruktioner som genereras. Man kan t. ex. titta på den utskrift som man kan få vid kompilering figur 3-1. Det fordras en del arbete för att klara ut vilka assemblerinstruktioner som genereras av en speciell PL/M-sats. Det kan ju vara intressant att jämföra olika alternativ för att finna ut vilket som är bäst.
Om man bara är intresserad av att så snabbt som möjligt skriva ett program för en mikrodator utan hänsyn till snabbhet hos programmet eller programmets storlek, då är PL/M ett bra språk.

figur 3-1

```
8080 PLM2 VERS 4.1


    1=0003H     8=0006H     9=000EH    12=0012H    13=001AH    14=001CH
   15=002AH    16=0040H    17=0043H    18=0048H    19=0051H    20=0066H
   21=007AH    22=0080H    23=0087H    24=008AH    25=008BH
STACK SIZE = 0 BYTES
MEMORY.....0100H
SORT.....0006H
PTR.....00FAH
N.....00FCH
I.....00FDH
SWAP.....00FEH
TEMP.....00FFH
0000H LXI SP FAH    00H     JMP     8BH     00H     LXI H   FAH     00H
0009H MOV MC INX H  MOV MB  INR L   MOV ME MOV LI FEH     MOV MI  01H
0012H LXI H  FEH    00H     MOV AA MRC      JNC     8AH     00H     MOV MI
001BH 00H    DCR L  MOV MI  00H     LXI H   FDH     00H     MOV AM  SUB I
0024H 02H    INR L  SUB M   JC      12H     00H     MOV CM  MOV BI  00H
002DH LHLD   FAH    00H     DAD B   MOV AM LXI H   FDH     00H     MOV EM
0036H INR E  MOV DI 00H     LHLD    FAH     00H     DAD D   MOV CA  MOV AM
003FH SUB C  JNC    80H     00H     LXI H   FEH     00H     MOV MI  01H
0048H DCR L  MOV CM MOV BI  00H     LHLD    FAH     00H     DAD B   MOV AM
0051H LXI H  FEH    00H     MOV MA MOV LI FDH     MOV CM  MOV BI  00H
005AH LHLD   FAH    00H     DAD D   XCHG    LXI H   FDH     00H     MOV CM
0063H INR C  PUSH D MOV BI  00H     LHLD    FAH     00H     DAD B   MOV AM
006CH POP H  MOV MA LXI H   FDH     00H     MOV CM INR C    MOV BI  00H
0075H LHLD   FAH    00H     DAD D   XCHG    LXI H   FEH     00H     MOV CM
007EH MOV AC STAX D LXI H   FDH     00H     INR M   JNZ     1FH     00H
0087H JMP    12H    00H     RET     EI      HLT
NO PROGRAM ERRORS
```

3.5.1 HILP flernivåspråk för 8080 [9]

HILP kan skrivas på olika nivåer, från högnivå ner till ren assemblerkod. Språket är utvecklat vid Institutionen för Reglerteknik LTH, som ett examensarbete av Jan-Eric Aspernäs våren 1976.
Variabler och konstanter kan bestå av antingen 8 eller 16 ord.
Numeriska konstanter kan skrivas oktalt eller decimalt. En nolla som första siffra indikerar att talet är oktalt. Vektorer kan deklareras, deras index går från 0 till n-1.
Aritmetiska operatorer: + - .ADDC .SUBB
Logiska operatorer: .XOR .OR .AND
Relationsoperatorer: > <= = /=
Exempel på program finns i appendix B och C.
Ytterligare utveckling av språket pågår. Man strävar efter att göra en högnivådel som passar för olika typer av mikrodatorer. Språket är emellertid redan fullt användbart. För närmare upplysning hänvisas till referens 9.

3.5.2 Synpunkter på programmering i HILP

HILP står ungefär på samma höga nivå som PL/M och har den fördelen att användaren själv bestämmer vilken nivå han vill skriva på. Dessutom kan HILP ge en utmärkt utskrift, figur 3-2, där användaren direkt kan se assemblerkoden för varje HILP-sats. För den som kan assembler är detta speciellt värdefullt eftersom han då kan se var och hur han bör optimera programmet. HILP är speciellt avsett för den som vill skriva ett någorlunda optimalt program utan att han skall behöva skriva hela programmet i assembler.
Det går också bra att översätta direkt från t. ex. FORTRAN till högsta möjliga nivå hos HILP vilket visas i avsnitt 4.4. Vi ser att HILP-programmen innehåller många subrutinanrop och då dessa subrutiner får sina parametrar genom registeroperationer så blir programmen ändå ganska optimala. Detta visas tydligt i nästa avsnitt där vi jämför HILP med PL/M.

3.6.1 Jämförelse nr. 1 HILP - PL/M

Vid den första jämförelsen utgick vi ifrån några satser hämtade från ett FORTRAN-program nämligen:

```
/                                   IF Z[I]>Z[I+1] THEN     ← HILP-sats
        MVI     L,I+0
        MOV     A,M
        ADI     01
        ADI     Z
        MOV     L,A
        MOV     A,M
        PUSH    PSW
        MVI     L,I+0
        MOV     A,M
        ADI     Z
        MOV     L,A
        POP     PSW
        CMP     M
        JP      X4
/                                     TEMP=Z[I]
        MOV     A,M
        MVI     L,TEMP+0
        MOV     M,A
/                                     Z[I]=Z[I+1]
        MVI     L,I+0
        MOV     A,M
        ADI     01
        ADI     Z
        MOV     L,A
        MOV     A,M
        PUSH    PSW
        MVI     L,I+0
        MOV     A,M
        ADI     Z
        MOV     L,A
        POP     PSW
        MOV     M,A
/                                     Z[I+1]=TEMP
        MVI     L,TEMP+0
        MOV     A,M
        PUSH    PSW
        MVI     L,I+0
        MOV     A,M
        ADI     01
        ADI     Z
        MOV     L,A
        POP     PSW
        MOV     M,A
/                                     SWAP=1
        MVI     L,SWAP+0
        MVI     M,01
/                                   END
X4=.
/                                   I=I+1
        MVI     L,I+0
        INR     M
/                                UNTIL I=N-1
        MVI     L,N+0
        MOV     A,M
        SUI     01
        MVI     L,I+0
        CMP     M
        JNZ     XR3
/                                ENDWHILE
        JMP     XW1
X2=.
/                   ENDPROC
        RET
```

```
      DO 10 I=1,3
      X(I)=X(I)+HH
      F1=X(1)*X(2)+X(3)*X(3)
      G(I)=(F1-F)/HH
10    X(I)=X(I)-HH
      DXNEW=0.0
      DO 20 I=1,3
20    DXNEW=DXNEW+G(I)*G(I)
      ALPHA=DXNEW/DXOLD
      IF(K.EQ.0) ALPHA=0.0
      DXOLD=DXNEW
```

Avsikten var dels att jämföra antalet ord som genererades av respektive programspråk och dels att jämföra hur lätt det var att översätta ett program skrivet i FORTRAN.
I PL/M var det nästan bara att kopiera satserna.
HILP har inga subrutiner för multiplikation eller division, sådana måste man alltså skriva själv. Det kan vara en fördel att skriva subrutinerna själv ty då kan man bestämma hur dessa skall optimeras. När man väl har subrutinerna är det bara att skriva ner HILP-satserna direkt. Slutsats: Det var något lättare att programmera i PL/M men det gick lätt även i HILP. Programmen finns listade i appendix C på sidorna C1 och C2.
Vid jämförelse av antalet genererade ord fick vi för PL/M 351 ord varav 80 ord motsvarade subrutiner för multiplikation och division, dessa läggs ut en gång i varje program på det ställe där de först används. För jämförelse med HILP betraktades dessa som externa. Vi fick alltså 271 ord. HILP gav däremot 171 ord dvs. en avsevärd minskning. Tittar vi närmare på den genererade assemblerkoden och jämför sats för sats finner vi följande:

	antal genererade ord	
FORTRAN-satser	PL/M	HILP
X(I)=X(I)+HH	23	21
F1=X(1)*X(2)+X(3)*X(3)	59	27 !
G(I)=(F1-F)/HH	38	20 !
DXNEW=DXNEW+G(I)*G(I)	39	20 !
IF(K.EQ.0) ALPHA=0.0	13	10

Det är speciellt vid multiplikationer och divisioner som skillnaden är stor. Detta beror på att man genom sina subrutinanrop styr beräkningen av de komplicerade satserna på ett effektivt sätt i HILP. Registeroperationerna för överförande av parametrar till

subrutinen är också bra t. ex. CALL MULT(B=G(I),C=B) där vi bara beräknar G(I)'en gång. Det är ganska naturligt att vi får ett färre antal ord i dessa fall ty nivån på HILP är här lägre än för PL/M.

3.6.2 Jämförelse nr. 2 HILP - PL/M

De bägge subrutinerna FLOAT och IFIX programmerades i PL/M och HILP enligt samma grundprincip. Programmen finns listade i appendix C på sidorna C3-C5. I HILP användes registeroperationer till stor del för att programmet skulle bli någorlunda optimalt. Vid jämförelse fann vi att PL/M gav 308 ord och HILP gav 153 ord. Den stora differensen berodde delvis på att HILP ej behövde ha satser som HIGH(X) eller LOW(X). HILP-programmet är inte identiskt med motsvarande i PL/M. Skillnaderna skall nu gås igenom.
PL/M-satsen SHL(X,1) motsvaras i HILP av instruktionerna

```
.ORA A
E=E.RAL
D=D.RAL
```

där då x ligger i registerparet D,E. Den första instruktionen .ORA A är bara ett sätt att nollställa carry-flaggan men ändrar i övrigt ingenting ty $A \vee A = A$.
Satsen REG D=D+1 ger avrundning ty D och B innehåller mantissan vänsterskiftad ett steg. D skiftas nu höger och innehåller alltså mantissan avrundad. B kommer att användas för ett test liknande det i PL/M-programmet. Detta testet innebär att mantissan får värdet av M resp. MA om avrundning ger att exponenten skall ökas ett steg. Där M och MA antar värdena 0.5 eller -0.5.
Proceduren IFIX är ganska lika i PL/M och HILP.
Trots att programmen i HILP respektive PL/M ej är helt identiska och att denna typen av procedurer lämpar sig väl för registeroperationer kan vi ändå konstatera en avsevärd sänkning av antalet ord.

3.6.3 Jämförelse nr. 3 HILP - PL/M

En subrutin som sorterar en vektor enligt metoden Bubblesort har programmerats i PL/M därefter har motsvarande programmerats i HILP på tre olika nivåer från högnivå till extrem användning av registeroperationer. Programmen finns listade i appendix C

på sidorna C6-C9. Resultatet blev 137 ord för PL/M och de olika HILP-varianterna gav 101, 60 resp.37 ord. Högnivåprogrammet i HILP kräver att de vektorer som skall sorteras finns i minnet med en bestämd låg-adress. Hög-adressen (bank-adressen) varierar för de olika vektorerna. Detta betyder att vi bara kan sortera ett begränsat antal vektorer och dessa kan i sin tur inte vara hur långa som helst. Den största nackdelen är kanske att vi har BANK UNDEFINED dvs. användaren måste själv se till att register H innehåller rätt bankadress. Detta betyder att användaren måste känna till vilka instruktioner som ändrar H-värdet. I mellanvarianten av HILP har vi strukit de indexerade uttrycken och övergått till registeroperationer. Nu har vi inte längre några restriktioner på antalet vektorer eller längden hos dessa. I lågnivåprogrammet är inga variabler alls deklarerade vi använder bara registren. Vissa instruktioner kan förstöra andra register än de som verkar användas, som exempel kan nämnas addition av två registerpar. Oavsett vilka registerpar som används så kommer registerparet H,L att ändra värde ty H,L används som ackumulator vid dubbelordsräkning. Användaren måste alltså känna till vilka instruktioner som ändrar innehåll i andra register än de som medverkar i instruktionen när han programmerar på låg nivå.

3.6.4 Sammanfattning och kommentar

Jämförelse nr. 1	
PL/M	271 ord
HILP (högnivå)	171 ord
Jämförelse nr. 2	
PL/M	308 ord
HILP (lågnivå)	153 ord
Jämförelse nr. 3	
PL/M	137 ord
HILP (högnivå)	101 ord
HILP (medelnivå)	60 ord
HILP (lågnivå)	37 ord

Jämförelse mellan programspråken ger vid handen att HILP genererar mindre kod än PL/M, men att detta till stor del beror på att HILP inte ligger på lika hög nivå som PL/M. PL/M kräver inte att användaren skall känna till register eller bankadresser. Den största fördelen med HILP är att man kan programmera på olika nivåer och att man på detta sättet kan optimera vissa delar av programmet.

KAPITEL 4. PROGRAMMERING I HILP

4.1 Inledning

Först nämns lite om talrepresentation i en mikrodator, därefter något om programmering i HILP för att avslutningsvis redovisa storleken hos programmen i antalet ord.

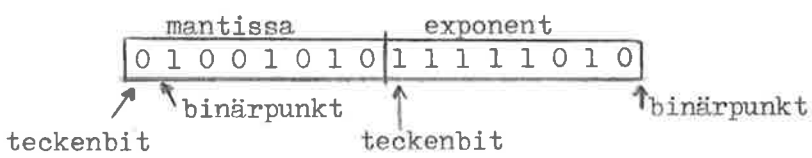
4.2 Talrepresentation i mikrodator

För att få så stor noggrannhet som möjligt vid aritmetiska beräkningar vill man att alla tal representeras i intervallet $0.5 \leq |x| < 1.0$. Detta kan man åstadkomma på två sätt, dels genom att införa skalfaktorer och dels genom att låta talet bestå av en mantissa och en exponent s. k. flyttal. Skalfaktorer kan vara svåra att bestämma eftersom man kanske inte vet storleksordningen på talet, därför väljer vi det andra sättet. Ett tal x representeras alltså av en mantissa m där $0.5 \leq |m| < 1.0$ och en exponent p där p är ett positivt eller negativt heltal. Värdet av x är:

$$x = m \cdot 2^p$$

Talets noggrannhet bestäms av antalet bitar i m, medan antalet bitar i p bestämmer storleksordningen på talet. Vi har valt att representera ett tal x med 8 bitars mantissa samt 8 bitars exponent. Då den första biten i både m och p utgörs av en teckenbit fås alltså en noggrannhet på 0.4 % och ett talområde mellan 2^{-128} och 2^{127} (ungefär 10^{-39} till 10^{38}). I figur 4-1 kan vi se hur talet placeras i ett 16 bitars ord:

figur 4-1



Talet som representeras i figur 4-1 har $m = 74/128$
exponenten är negativ, i 2-komplement blir alltså $p = -6$
dvs. $x = \frac{74}{128} \cdot 2^{-6} = \frac{74}{128 \cdot 64} = 9.03 \cdot 10^{-3}$

4.3 Subrutiner i HILP

Att vi valde 8 bitar för både m och p beror på att både HILP och PL/M kan handskas med 8 resp. 16 bitar. Det är med andra ord ganska lätt att skriva subrutiner för flytande aritmetik. I appendix B redovisas dessa subrutiner samt några andra som var nödvändiga vid översättning från FORTRAN-programmen till HILP. Subrutinerna gör inte anspråk på att vara optimala, men de är skrivna på ganska låg nivå, för att vara relativt snabba och för att inte generera alltför mycket kod. I flera av subrutinerna förekommer att ord skiftas ett antal steg åt höger eller vänster med nollor in i andra ändan. För att klara av detta har vi instruktionerna .RAR resp. .RAL som roterar innehåll + carry ett steg. Vad vi behöver göra är alltså att nollställa carry-flaggan före varje skift. Detta kan göras med instruktionerna .STC och .CTC dvs. 1-ställ carry och komplementera. Detta tar totalt 8 cykler. Ett annat alternativ är instruktionen .ORA A som tar 4 cykler och som innebär logiska operatorn .OR mellan innehållet i ackumulatorn och innehållet i ackumulatorn dvs. A förblir oförändrad, dessutom innebär instruktionen att carry-flagga nollställs.
Det senare alternativet valdes.

4.4. Huvudprogram i HILP

När subrutinerna var skrivna var det inte svårt att översätta huvudprogrammet till HILP se appendix B. Översättningen har gjorts direkt från FORTRAN så överensstämmelsen är god mellan programmen. Lägena har samma nummer och kommentarerna är lika. I HILP gäller att allt som följer efter semikolon på samma rad är kommentarer. Subrutinen FSUB utnyttjas sällan, eftersom de ingående talen måste vara positiva. Kan man inte garantera detta får man negera det andra talet och använda subrutinen FADD.
Inga försök har gjorts för att optimera programmen. Register användes dock ganska mycket, vilket beror på att subrutinerna får sina parametrar genom registeroperationer. I avsnitt 3.6 har vi sett att denna typen av parameteröverföring till subrutiner ger betydligt färre antal ord än den metod som PL/M använder. Då programmen består av ett flertal subrutinanrop får vi på detta sättet ändå en viss optimering.
Relationer mellan två flyttal återförs på att undersöka om ett uttryck är >0 dvs. $x>y$ blir $x-y>0$. Resultatet av uttrycket läggs i ett registerpar, sedan behöver man bara undersöka om det högre registret är >0.
I FORTRAN är programmet Fletcher-Reeves något större än Absolute Bias. I assemblerkod är skillnaden mycket större vilket beror på att Fletcher-Reeves använder mera dubbelordsräkning. Om vi tittar på deklarationerna så ser vi följande:

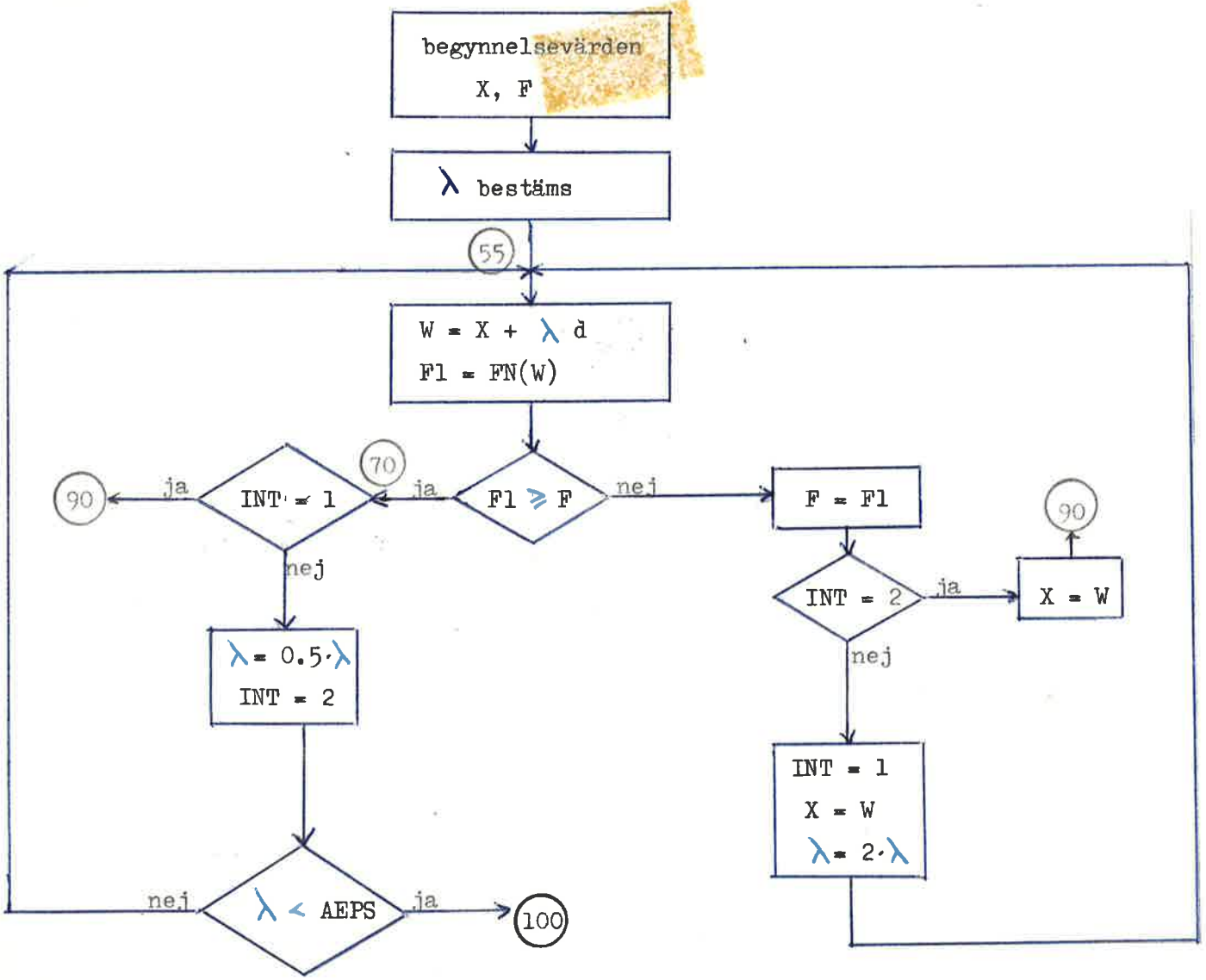
Metod	Antal variabler deklarerade		
	SINGLE	DOUBLE	Vektorer (DOUBLE)
Fletcher-Reeves	4	18	5
Absolute Bias	11	7	4

Det totala antalet ord för programmet med Fletcher-Reeves metod är 1999. Då minnesutrymmet är på 2k dvs. 2048 ord, är marginalen ganska liten. Vid körning på en verklig process kan man behöva några fler subrutiner t. ex. EXP(X). Subrutinen FVALUE kommer kanske att innehålla skalning av in- och utsignaler. Vi ser att det finns en stor risk att minnesutrymmet inte räcker till.

4.4.1. Förkortad version av Fletcher-Reeves metod

Vi kommer att ändra i den linjära minimeringen se flödesschema figur 4-2 och jämför med figur 2-1. Den kvadratiska interpólationen kommer att slopas och vi kommer i stället att halvera steget så länge vi får ett funktionsvärde som är större än det vi startade med. Effekten av denna ändring är att minimeringsprocessen blir långsammare och inte lika effektiv som tidigare. I appendix B redovisas detta programmet i HILP. Antalet ord för enbart huvud-programmet minskades från 1184 ord till 961 ord.

figur 4-2



kommentar: satsnummer markeras i ring.

(90) betyder lin. min. klar

(100) betyder abs. min. klar

4.5. Redovisning av antalet ord för huvudprogram och subrutiner

Absolute Bias (huvudprogram)	558
Fletcher-Reeves (huvudprogram)	1184
Fletcher-Reeves (huvudprogram, förkortad version)	961

Subrutiner

MULT	30
DIV	50
IFIX	88
FLOAT	93
FMULT	113
FDIV	112
FSUB	136
FADD	147
FVALUE	46
totalt för subrutiner	815

Totalt

Fletcher-Reeves	1999
Fletcher-Reeves (förkortad)	1776
Absolute Bias	1373

Referenser

1. J.-J. Østergaard,"DYNAMIC VERSION OF DIRECT OPTIMIZATION" Electrical Power Engineering Dept., Technical University of Denmark, Publication no. 7509, July 1975.

2. Douglass J. Wilde,"OPTIMUM SEEKING METHODS", Prentice-Hall, Inc. Englewood Cliffs, N.J., 1964.

3. S.L.S. Jacoby, J.S. Kowalik, J.T. Pizzo,"ITERATIVE METHODS FOR NONLINEAR OPTIMIZATION PROBLEMS", Prentice-Hall, Inc. Englewood Cliffs, N.J. 1972.

4. Håkan Ramsin,"ICKELINJÄR OPTIMERING", Institutet för tillämpad matematik, KTH, ITM arbetsrapport nr 20.

5. Birger Jansson,"RANDOM NUMBER GENERATORS", Victor Pettersons Bokindustri Aktiebolag, Stockholm 1966.

6. 8008 and 8080 PL/M Programming Manual, Intel Corporation, 1975.

7. 8080 PL/M Compiler Operators Manual, Intel Corporation, 1975.

8. Intel 8080 Microcomputer Systems User´s Manual.

9. Jan-Eric Aspernäs,"MEDIUM LEVEL PROGRAMMING LANGUAGES FOR MICRO PROCESSORS", RE-177, 1976.

APPENDIX

programlistningar

```
C       PROGRAM FOR FINDING THE MINIMUM OF A FUNCTION F(X)
C        WHEN ONLY FUNCTIONVALUES ARE AVAILABLE, BY
C        FLETCHER-REEVES METHOD.
C       AUTHOR: I.SIXTENSSON 1976-03-08
C       REFERENCE: S.L.S.JACOBY,J.S.KOWALIK,J.T.PIZZO,
C        ITERATIVE METHODS FOR NONLINEAR OPTIMIZATION
C        PROBLEMS.
C
C       PARAMETERS
C
C       XLAMB    THE STEPLENGTH IN THE DESCENT DIRECTION
C       RIKTN    A REAL ARRAY OF N ELEMENTS CONTAINING
C                 THE DESCENT DIRECTION
C       IND      INDICATES THE WAY OF CALCULATING THE
C                 STARTINGVALUE OF XLAMB AT EACH NEW POINT
C       G        A REAL ARRAY OF N ELEMENTS CONTAINING
C                 THE NUMERICAL ESTIMATE OF THE GRADIENT
C                 VECTOR
C       GSO      THE DIRECTIONAL DERIVATE OF F(X) AT X
C       AEPS     AEPS=EPS1/Z WHERE Z IS MAX(ABS(RIKTN(I)))
C                 AND EPS1 A PREDETERMINED MINIMAL STEPSIZE
C                 TERMINATION IF XLAMB IS LESS THAN AEPS
C       EPS2     TERMINATION IF THE REDUCTION OF THE
C                 FUNCTIONVALUE IS LESS THAN EPS2
C       TIME10   INTERNAL FORTRAN PROCEDURE.CALCULATES
C                 THE TIME DOWN TO TENTH OF SECONDS
C
C       SUBROUTINES REQUIRED
C                NUDER2 (OR NUDER1)
C                FN (IS A FUNCTION)
C
        DIMENSION X(2),W(2),RIKTN(2),OLDRTN(2),G(2)
        N=2
        X(1)=-1.2
        X(2)=1.0
        IND=1
        ITER=0
        XLAMB=1.0
        DXOLD=1.0
        H=0.1E-04
        EPS1=0.1E-4
        EPS2=0.1E-6
        F=FN(X)
        DF=F/2.0
        WRITE(6,200) EPS1,EPS2,H,IND
200     FORMAT(' PARAMETERS'/' EPS1=',E10.3,3X,'EPS2=',
     1  E10.3/' H=',E9.2,3X,'IND=',I2//)
        CALL TIME10(IMIN,ISEC,ISEC10,IOFF)
C
C       ITERATION PROCEDURE STARTS
C
10      K=0
15      FF=F
        ITER=ITER+1
C
C       CALCULATE THE DESCENT DIRECTION
C
        CALL NUDER2 (X,G,H,N)
        DXNEW=0.0
        DO 20 I=1,N
20      DXNEW=DXNEW+G(I)*G(I)
        ALPHA=DXNEW/DXOLD
        IF(K.EQ.0) ALPHA=0.0
```

```
      DXOLD=DXNEW
      DO 30 I=1,N
      RIKTN(I)=-G(I)+ALPHA*OLDRTN(I)
30    OLDRTN(I)=RIKTN(I)
C
      Z=0.0
      GSO =0.0
      DO 40 I=1,N
      ABSRTN=ABS(RIKTN(I))
      IF(Z.GE.ABSRTN) GO TO 40
      Z=ABSRTN
40    GSO=GSO+G(I)*RIKTN(I)
      AEPS=EPS1/Z
      IF(GSO.LT.0.0) GO TO 52
      GSO=-GSO
      DO 50 I=1,N
      RIKTN(I)=-RIKTN(I)
50    OLDRTN(I)=RIKTN(I)
C
C     MINIMIZE ALONG THE DESCENT DIRECTION
C
52    RIND=IND
      IF(IND.GE.1) XLAMB=-2.0*DF/GSO/RIND
      INT=0
55    DO 60 I=1,N
60    W(I)=X(I)+XLAMB*RIKTN(I)
      F1=FN(W)
      IF(F1.GE.F) GO TO 70
      F=F1
      IF(INT.EQ.2) GO TO 87
      INT=1
      DO 63 I=1,N
63    X(I)=W(I)
      XLAMB=2.0*XLAMB
      GO TO 55
70    IF(INT.EQ.1) GO TO 90
      XLAMB=0.5*XLAMB
      DO 80 I=1,N
80    W(I)=X(I)+XLAMB*RIKTN(I)
      F2=FN(W)
      IF(F2.GE.F) GO TO 85
      F=F2
      GO TO 87
85    Z=0.1
      IF(F1+F.GT.F2+F2) Z=1.0+0.5*(F-F1)/(F+F1-F2-F2)
      IF(Z.LT.0.1) Z=0.1
      XLAMB=Z*XLAMB
      INT=2
      IF(XLAMB.LT.AEPS) GO TO 100
      GO TO 55
87    DO 88 I=1,N
88    X(I)=W(I)
90    DF=FF-F
      IF(DF.LT.EPS2) GO TO 100
C
C     TEST FOR RECYCLING
C
      IF(K.EQ.N) GO TO 10
      K=K+1
      GO TO 15
C
C     STOP THE CLOCK
C
100   IOFF=1
```

```
      WRITE(6,130) ITER
      WRITE(6,140)ISEC, ISEC10
      WRITE(6,150)
      WRITE(6,160) (X(I),I=1,N)
      WRITE(6,170) F
      STOP
130   FORMAT(//' NUMBER OF ITERATIONS',I3)
140   FORMAT(' TIME IN SECONDS',I2,'.',I1)
150   FORMAT(' POINT OF MINIMUM')
160   FORMAT(5X,F9.3)
170   FORMAT(' WITH THE FUNCTIONVALUE',E10.3)
      END
```

```
C       PROGRAM FOR FINDING THE MINIMUM OF A FUNCTION F(X)
C       WHEN ONLY FUNCTIONVALUES ARE AVAILABLE, BY
C       THE METHOD ABSOLUTE BIAS.
C       AUTHOR: I.SIXTENSSON  1976-02-22
C       REFERENCE: J.-J.OSTERGAARD, DYNAMIC VERSION OF
C       DIRECT OPTIMIZATION,PUBLICATION NO. 7509,
C       TECHNICAL UNIVERSITY OF DENMARK
C
C       PARAMETERS
C
C       IS      NUMBER OF SUCCEDING EXPERIMENTS THAT
C                WHERE ALL SUCCESSFUL
C       KN      NUMBER OF SUCCEDING EXPERIMENTS THAT
C                WHERE ALL FAILURES
C       IR      TAKES THE VALUES 0,1,2 DEPENDING ON
C                THE RESULT OF THE LAST EXPERIMENT
C       ISMAX   WHEN IS>ISMAX THEN THE STEPLENGTH
C                IS PROBABLY TOO SMALL. THE STEPLENGTH
C                WILL THEN BE DOUBLED AS LONG AS THE
C                EXPERIMENT IS SUCCESSFUL
C       KNMAX   WHEN KN=KNMAX THEN KL WILL BE INCREMENTED
C                AND SIGMA WILL BE REDUCED
C       KL      COUNTER
C       KLAR    TERMINATION WHEN KL=KLAR
C       K       K=0 WILL GIVE A RESTART OF THE PROCEDURE
C       SIKO    THE FACTOR BY WHICH SIGMA IS REDUCED
C       QM      THE BEST FUNCTIONVALUE SO FAR
C       QK      THE FUNCTIONVALUE TO BE EXAMINED
C       UM      COORDINATES CORRESPONDING TO QM
C       UK      COORDINATES CORRESPONDING TO QK
C       EPS     THE REDUCTION OF THE FUNCTIONVALUE WILL
C                BE AT LEAST EPS
C       TIME10  INTERNAL FORTRAN PROCEDURE. CALCULATES THE
C                TIME DOWN TO TENTH OF SECONDS
C
C       SUBROUTINES REQUIRED
C               GAUSS
C               FN (IS A FUNCTION)
C
      DIMENSION UK(2),UM(2),DU(2)
      N=2
      UK(1)=-1.2
      UK(2)=1.0
      KNMAX=10
      ISMAX=5
      SIGMA=0.3
      EPS=0
      IX=3847
      IS=0
      IR=0
      SIKO=0.1
      K=0
      KL=1
      KLAR=3
      KN=0
      IANT=0
      WRITE(6,200) ISMAX,KNMAX,KLAR,SIKO,SIGMA,EPS,IX
200   FORMAT(' ISMAX=',I2,3X,'KNMAX=',I2,3X,
     1 'KLAR=',I2,3X,'SIKO=',F4.2/
     1 ' SIGMA=',F5.3,3X,'EPS=',F4.2,3X,'IX=',I5)
      CALL TIME10(IMIN,ISEC,ISEC10,IOFF)
C
C       ITERATION PROCEDURE STARTS
```

```
C
10      QK=FN(UK)
        IANT=IANT+1
        IF(K.NE.0) GO TO 20
        DO 15 I=1,N
15      UM(I)=UK(I)
        QM=QK
        K=1
        GO TO 30
20      IF(QK.LT.QM-EPS) GO TO 60
C
C       FAILURE
C
        XLAMB=1.0
        IF(IS.EQ.0) GO TO 40
30      CALL GAUSS (IX,SIGMA,DU,N)
        IS=0
        IR=0
        GO TO 70
40      KN=KN+1
        IR=IR+1
        IF(KN.LT.KNMAX) GO TO 50
        IF(KL.EQ.KLAR) GO TO 80
        KL=KL+1
        KN=0
        SIGMA=SIKO*SIGMA
50      IF(IR.EQ.2) GO TO 30
        DO 55 I=1,N
55      DU(I)=-DU(I)
        GO TO 70
C
C       SUCCESS
C
60      DO 65 I=1,N
65      UM(I)=UK(I)
        QM=QK
        IS=IS+1
        KN=0
        IF(IS.GT.ISMAX) XLAMB=2.0*XLAMB
70      DO 75 I=1,N
75      UK(I)=UM(I)+XLAMB*DU(I)
        GO TO 10
C
C       STOP THE CLOCK
C
80      IOFF=1
        WRITE(6,100) IANT
        WRITE(6,110)ISEC, ISEC10
        WRITE(6,120)
        WRITE(6,130) (UM(I),I=1,N)
        WRITE(6,140) QM
        STOP
100     FORMAT(//' NUMBER OF CALCULATED FUNCTIONVALUES',I5)
110     FORMAT(' TIME IN SECONDS',I2,'.',I1)
120     FORMAT(' POINT OF MINIMUM')
130     FORMAT(5X,F9.3)
140     FORMAT(' WITH THE FUNCTIONVALUE',E10.3)
        END
```

```
      SUBROUTINE GAUSS(IX,SIGMA,V,N)
C
C     GENERATES AN ARRAY OF N ELEMENTS WITH NORMALLY
C     DISTRIBUTED PSEUDO-RANDOM REAL NUMBERS
C     WITH THE MEAN VALUE 0 AND STANDARD
C     DEVIATION SIGMA, BY ADDING 12 UNIFORM
C     DISTRIBUTED REAL NUMBERS
C
C     FORMAL PARAMETERS
C
C     IX      A RELATIVE LARGE,ODD STARTING NUMBER
C              THE SAME NUMBER WILL GENERATE THE SAME
C              SEQUENS OF RANDOM NUMBERS
C     SIGMA   STANDARD DEVIATION
C     V       GIVES THE RANDOM VECTOR
C     N       THE NUMBER OF VARIABLES
C
C     SUBROUTINES REQUIRED
C             NONE
C
      DIMENSION V(1)
      DO 60 I=1,N
      A=0.0
      DO 50 K=1,12
      IY=IX*1795
      IF(IY.LT.0) IY=IY+131072
      IX=IY
      Y=IY
50    A=A+Y/131071.0
60    V(I)=(A-6.0)*SIGMA
      RETURN
      END
```

```
      SUBROUTINE NUDER1 (X,G,F,H,N)
C
C     NUMERICAL CALCULATION OF THE GRADIENT VECTOR
C     BY FORWARD DIFFERENCES
C
C     Y'(X)=(Y(X+H)-Y(X))/H
C     ACCURACY IS O(H)
C
C     FORMAL PARAMETERS
C
C     X        THE POINT WHERE THE GRADIENT VECTOR IS
C               CALCULATED
C     G        GIVES THE GRADIENT VECTOR
C     F        FUNCTIONVALUE AT X
C     H        THE DIFFERENCE
C     N        THE NUMBER OF VARIABLES
C
C     SUBROUTINES REQUIRED
C              FN
C
      DIMENSION X(1),G(1)
      DO 10 I=1,N
      X(I)=X(I)+H
      F1=FN(X)
      G(I)=(F1-F)/H
10    X(I)=X(I)-H
      RETURN
      END



      SUBROUTINE NUDER2 (X,G,H,N)
C
C     NUMERICAL CALCULATION OF THE GRADIENT VECTOR
C     BY CENTRAL DIFFERENCES
C
C     Y'(X)=(Y(X+H)-Y(X-H))/2/H
C     ACCURACY IS O(H**2)
C
C     FORMAL PARAMETERS
C
C     X        THE POINT WHERE THE GRADIENT VECTOR
C               IS CALCULATED
C     G        GIVES THE GRADIENT VECTOR
C     H        THE DIFFERENCE
C     N        THE NUMBER OF VARIABLES
C
C     SUBROUTINES REQUIRED
C              FN
C
      DIMENSION X(1),G(1)
      DO 10 I=1,N
      X(I)=X(I)+H
      F1=FN(X)
      X(I)=X(I)-H-H
      F2=FN(X)
      G(I)=(F1-F2)/(H+H)
10    X(I)=X(I)+H
      RETURN
      END
```

```
      FUNCTION FN(X)
C
C     CALCULATES THE FUNCTIONVALUE AT THE POINT X.
C     THE FUNCTION IS CALLED 'ROSENBROCKS CURVED
C     VALLEY' AND IS A WELLKNOWN TEST PROBLEM
C
      DIMENSION X(1)
      FN=100.0*(X(2)-X(1)*X(1))**2+(1.0-X(1))**2
      RETURN
      END
```

```
      FUNCTION FN(X)
C
C      CALCULATES THE FUNCTIONVALUE AT POINT X
C
C      SUBROUTINES REQUIRED
C      NONE
C
      DIMENSION X(1)
      FN=0.5*X(1)*X(1)+X(2)*X(2)+2.0*X(3)*X(3)
     1 +5.0*(X(1)*X(2))**2
      RETURN
      END
```

```
                        ;MAIN PROGRAM
;
;       PROGRAM FOR FINDING THE MINIMUM OF A FUNCTION F(X)
;       WHEN ONLY FUNCTIONVALUES ARE AVAILABLE, BY
;       FLETCHER-REEVES METHOD.
;       AUTHOR: I.SIXTENSSON  1976-04-28
;       REFERENCE: S.L.S.JACOBY,J.S.KOWALIK,J.T.PIZZO,
;       ITERATIVE METHODS FOR NONLINEAR OPTIMIZATION
;       PROBLEMS.
;
;       PARAMETERS
;
;       XLAMB   THE STEPLENGTH IN THE DESCENT DIRECTION
;       RIKTN   A REAL ARRAY OF N ELEMENTS CONTAINING THE
;               DESCENT DIRECTION
;       IND     INDICATES THE WAY OF CALCULATING THE
;               STARTINGVALUE OF XLAMB AT EACH NEW POINT
;       G       A REAL ARRAY OF N ELEMENTS CONTAINING THE
;               NUMERICAL ESTIMATE OF THE GRADIENT VECTOR
;       GSO     THE DIRECTIONAL DERIVATE OF F(X) AT X
;       AEPS    AEPS=EPS1/Z WHERE Z IS MAX(ABS(RIKTN(I)))
;               AND EPS1 A PREDETERMINED MINIMAL STEPSIZE
;               TERMINATION IF XLAMB IS LESS THAN AEPS
;       EPS2    TERMINATION IF THE REDUCTION OF THE
;               FUNCTIONVALUE IS LESS THAN EPS2
;       HF      THE DIFFERENCE IN CALCULATING THE
;               GRADIENT VECTOR
;
;       SUBROUTINES USED
;
;       PROC MULT       MULTIPLICATION
;                       ENTRY: BC=N (INTEGER,POSITIVE)
;                              DE=K (INTEGER,POSITIVE)
;                       EXIT:  DE=N*K (INTEGER)
;
;       PROC DIV        DIVISION WITH ROUNDING
;                       ENTRY: BC=N (INTEGER,POSITIVE)
;                              DE=K (INTEGER,POSITIVE)
;                       EXIT:  DE=N/K (INTEGER)
;
;       PROC FLOAT      CONVERTING AN INTEGER INTO
;                       A REAL NUMBER
;                       ENTRY: DE=N (INTEGER)
;                       EXIT:  DE   (REAL)
;
;       PROC IFIX       CONVERTING A REAL NUMBER INTO
;                       AN 8-BITS INTEGER
;                       ENTRY: DE=X (REAL)
;                       EXIT:  E    (INTEGER)
;
;       PROC FMULT      FLOATING MULTIPLICATION
;                       ENTRY: BC=X (REAL)
;                              DE=Y (REAL)
;                       EXIT:  DE=X*Y (REAL)
;
;       PROC FDIV       FLOATING DIVISION
;                       ENTRY: BC=X (REAL)
;                              DE=Y (REAL)
;                       EXIT:  DE=X/Y (REAL)
;
;       PROC FSUB       FLOATING SUBTRACTION
;                       ENTRY: BC=X (REAL,POSITIVE)
;                              DE=Y (REAL,POSITIVE)
```

```
;                          EXIT:  DE=X-Y (REAL)
;
;       PROC FADD          FLOATING ADDITION
;                          ENTRY: BC=X (REAL)
;                                 DE=Y (REAL)
;                          EXIT:  DE=X+Y (REAL)
;
;       PROC FVALUE        RETURNS A FUNCTIONVALUE OF AN
;                          ANALOG PROCESS
;                          ENTRY: B↑ STARTING ADDRESS OF X
;                          EXIT:  DE=FUNCTIONVALUE (REAL)
;
        CONST ¤BK=1,¤N=2
        BANK ¤BK
        SINGLE K↑10,I,IND,INT
        DOUBLE F,F1,F2,FF,DF,Z,GSO,AEPS,EPS1,EPS2,HF,HH
        DOUBLE ALPHA,DXNEW,DXOLD,XLAMB,ABSRTN,DIVFAC
        DOUBLE X[¤N],W[¤N],G[¤N],RIKTN[¤N],OLDRTN[¤N]
        SET BANK ¤BK
        SET STACK 2:255
        GLOBAL MULT,DIV,FLOAT,IFIX,FMULT,FDIV
        GLOBAL FSUB,FADD,FVALUE

        DX X[0]=0131401;          =-1.203
        DX X[1]=040001;           =1.0
        IND=0
        DX XLAMB=040001;          =1.0
        DX DXOLD=040001;          =1.0
        DX HF=051372;             =0.01
        DX EPS1=041367;           =0.101E-02
        DX EPS2=041367;           =0.101E-02

        CALL FVALUE(B↑X)
        DX F=DE;          F=F(X)
        CALL FDIV(BC=DE,DE=040002)
        DX DF=DE;         DF=F/2.0
;
;       ITERATION PROCEDURE STARTS
;
  L10:  K=0
  L15:  DX FF=F
;
;       CALCULATE THE DESCENT DIRECTION
;
;       NUMERICAL CALCULATION OF THE GRADIENT VECTOR
;
        I=¤N
        REPEAT
          I=I-1
          CALL FADD(BC=X[I],DE=HF)
          DX X[I]=DE;     X(I)=X(I)+HF
          CALL FVALUE(B↑X)
          DX F1=DE;       F1=F(X)
          CALL FADD(BC=HF,DE=BC)
          DX HH=DE;       HH=HF+HF
          D=-D
          CALL FADD(BC=X[I],DE=DE)
          DX X[I]=DE;     X(I)=X(I)-HH
          CALL FVALUE(B↑X)
          D=-D
          CALL FADD(BC=F1,DE=DE)
          CALL FDIV(BC=DE,DE=HH)
          DX G[I]=DE;     G(I)=(F1-F(X))/HH
          CALL FADD(BC=X[I],DE=HF)
```

```
    DX X[I]=DE;    X(I)=X(I)+HF
  UNTIL I=0

  DX DXNEW=0
  I=#N
  REPEAT
    I=I-1
    CALL FMULT(BC=G[I],DE=BC)
    CALL FADD(BC=DXNEW,DE=DE)
    DX DXNEW=DE;  DXNEW=DXNEW+G(I)*G(I)
  UNTIL I=0
  CALL FDIV(BC=DE,DE=DXOLD)
  DX ALPHA=DE;     ALPHA=DXNEW/DXOLD
  IF K=0 THEN
    DX ALPHA=0
  END
  DX DXOLD=DXNEW
  I=#N
  REPEAT
    I=I-1
    CALL FMULT(BC=ALPHA,DE=OLDRTN[I])
    DREG BC=G[I]
    B=-B
    CALL FADD(BC=BC,DE=DE)
    DX RIKTN[I]=DE;        RIKTN(I)=-G(I)+ALPHA*OLDRTN(I)
    DX OLDRTN[I]=DE;       OLDRTN(I)=RIKTN(I)
  UNTIL I=0


  DX Z=0
  DX GSO=0
  I=#N
  REPEAT
    I=I-1
    DREG BC=RIKTN[I]
    IF 0>B THEN
      B=-B
    END
    DX ABSRTN=BC
    CALL FSUB(BC=BC,DE=Z)
    IF 0<=D THEN
      DX Z=ABSRTN;         Z WILL BE MAX(ABS(RIKTN(I)))
    END
    CALL FMULT(BC=G[I],DE=RIKTN[I])
    CALL FADD(BC=GSO,DE=DE)
    DX GSO=DE;     GSO=GSO+G(I)*RIKTN(I)
  UNTIL I=0
  CALL FDIV(BC=EPS1,DE=Z)
  DX AEPS=DE;      AEPS=EPS1/Z
  DREG DE=GSO
  GOTO L52 IF 0>D;         GOTO L52 IF GSO<0
  D=-D
  DX GSO=DE;       GSO=-GSO
  I=#N
  REPEAT
    I=I-1
    DREG DE=RIKTN[I]
    D=-D
    DX RIKTN[I]=DE;        RIKTN(I)=-RIKTN(I)
    DX OLDRTN[I]=DE;       OLDRTN(I)=RIKTN(I)
  UNTIL I=0
;
;       MINIMIZE ALONG THE DESCENT DIRECTION
;
```

```
L52:  IF IND>0 THEN
        CALL FLOAT(DE=IND)
        CALL FDIV(BC=0140002,DE=DE)
        CALL FMULT(BC=DF,DE=DE)
        CALL FDIV(BC=DE,DE=GSO)
        DX XLAMB=DE;  XLAMB=-2.0*DF/GSO/IND
      END
      INT=0
L55:  I=□N
      REPEAT
        I=I-1
        CALL FMULT(BC=XLAMB,DE=RIKTN[I])
        CALL FADD(BC=X[I],DE=DE)
        DX W[I]=DE;   W(I)=X(I)+XLAMB*RIKTN(I)
      UNTIL I=0
      CALL FVALUE(B↑W)
      DX F1=DE;       F1=F(W)
      D=-D
      CALL FADD(BC=F,DE=DE)
      GOTO L70 IF 0>D;                GOTO L70 IF F1>F
      DX F=F1
      GOTO L87 IF INT=2
      INT=1
      I=□N
      REPEAT
        I=I-1
        DX X[I]=W[I]
      UNTIL I=0
      CALL FADD(BC=XLAMB,DE=BC)
      DX XLAMB=DE;    XLAMB=2.0*XLAMB
      GOTO L55
L70:  GOTO L90 IF INT=1
      CALL FMULT(BC=040000,DE=XLAMB)
      DX XLAMB=DE;    XLAMB=0.5*XLAMB
      I=□N
      REPEAT
        I=I-1
        CALL FMULT(BC=XLAMB,DE=RIKTN[I])
        CALL FADD(BC=X[I],DE=DE)
        DX W[I]=DE;   W(I)=X(I)+XLAMB*RIKTN(I)
      UNTIL I=0
      CALL FVALUE(B↑W)
      DX F2=DE;       F2=F(W)
      D=-D
      CALL FADD(BC=F,DE=DE)
      GOTO L85 IF D<=0;       GOTO L85 IF F2>=F
      DX F=F2
      GOTO L87
L85:  DX Z=063375;    Z=0.996
      CALL FADD(BC=F2,DE=BC)
      D=-D
      CALL FADD(BC=F1,DE=DE)
      CALL FADD(BC=F,DE=DE)
      DX DIVFAC=DE;   DIVFAC=F+F1-F2-F2
      IF D>0 THEN
        DREG BC=F1
        B=-B
        CALL FADD(BC=BC,DE=F)
        CALL FDIV(BC=DE,DE=DIVFAC)
        CALL FMULT(BC=040000,DE=DE)
        CALL FADD(BC=040001,DE=DE)
        DX Z=DE;      Z=1.0+0.5*(F-F1)/(F+F1-F2-F2)
      END
      CALL FADD(BC=Z,DE=0115375);     0115375=-0.996
```

```
          IF D<=0 THEN
            DX Z=063375
          END
          CALL FMULT(BC=Z,DE=XLAMB)
          DX XLAMB=DE;     XLAMB=Z*XLAMB
          INT=2
          CALL FSUB(BC=XLAMB,DE=AEPS)
          GOTO L100 IF D<=0;       GOTO L100 IF XLAMB<=AEPS
          GOTO L55
    L87:  I=□N
          REPEAT
            I=I-1
          DX X[I]=W[I]
          UNTIL I=0
          DREG DE=F
          D=-D
    L90:  CALL FADD(BC=FF,DE=DE)
          DX DF=DE;        DF=FF-F
          CALL FSUB(BC=EPS2,DE=DF)
          GOTO L100 IF D>0;        GOTO L100 IF DF<EPS2
;
;         TEST FOR RECYCLING
;
          GOTO L10 IF K=□N
          K=K+1
          GOTO L15
   L100:
          FINISH
```

```
                        ;MAIN PROGRAM
;
;       PROGRAM FOR FINDING THE MINIMUM OF A FUNCTION F(X)
;       WHEN ONLY FUNCTIONVALUES ARE AVAILABLE, BY
;       FLETCHER-REEVES METHOD,
;       AUTHOR: I.SIXTENSSON  1976-04-28
;       REFERENCE: S.L.S.JACOBY,J.S.KOWALIK,J.T.PIZZO,
;       ITERATIVE METHODS FOR NONLINEAR OPTIMIZATION
;       PROBLEMS.
;
;       PARAMETERS
;
;       XLAMB    THE STEPLENGTH IN THE DESCENT DIRECTION
;       RIKTN    A REAL ARRAY OF N ELEMENTS CONTAINING THE
;                DESCENT DIRECTION
;       IND      INDICATES THE WAY OF CALCULATING THE
;                STARTINGVALUE OF XLAMB AT EACH NEW POINT
;       G        A REAL ARRAY OF N ELEMENTS CONTAINING THE
;                NUMERICAL ESTIMATE OF THE GRADIENT VECTOR
;       GSO      THE DIRECTIONAL DERIVATE OF F(X) AT X
;       AEPS     AEPS=EPS1/Z WHERE Z IS MAX(ABS(RIKTN(I)))
;                AND EPS1 A PREDETERMINED MINIMAL STEPSIZE
;                TERMINATION IF XLAMB IS LESS THAN AEPS
;       EPS2     TERMINATION IF THE REDUCTION OF THE
;                FUNCTIONVALUE IS LESS THAN EPS2
;       HF       THE DIFFERENCE IN CALCULATING THE
;                GRADIENT VECTOR
;
;       SUBROUTINES USED
;
;       PROC MULT        MULTIPLICATION
;                        ENTRY: BC=N (INTEGER,POSITIVE)
;                               DE=K (INTEGER,POSITIVE)
;                        EXIT:  DE=N*K (INTEGER)
;
;       PROC DIV         DIVISION WITH ROUNDING
;                        ENTRY: BC=N (INTEGER,POSITIVE)
;                               DE=K (INTEGER,POSITIVE)
;                        EXIT:  DE=N/K (INTEGER)
;
;       PROC FLOAT       CONVERTING AN INTEGER INTO
;                        A REAL NUMBER
;                        ENTRY: DE=N (INTEGER)
;                        EXIT:  DE   (REAL)
;
;       PROC IFIX        CONVERTING A REAL NUMBER INTO
;                        AN 8-BITS INTEGER
;                        ENTRY: DE=X (REAL)
;                        EXIT:  E    (INTEGER)
;
;       PROC FMULT       FLOATING MULTIPLICATION
;                        ENTRY: BC=X (REAL)
;                               DE=Y (REAL)
;                        EXIT:  DE=X*Y (REAL)
;
;       PROC FDIV        FLOATING DIVISION
;                        ENTRY: BC=X (REAL)
;                               DE=Y (REAL)
;                        EXIT:  DE=X/Y (REAL)
;
;       PROC FSUB        FLOATING SUBTRACTION
;                        ENTRY: BC=X (REAL,POSITIVE)
;                               DE=Y (REAL,POSITIVE)
```

```
;                       EXIT:  DE=X-Y (REAL)
;
;       PROC FADD       FLOATING ADDITION
;                       ENTRY: BC=X (REAL)
;                              DE=Y (REAL)
;                       EXIT:  DE=X+Y (REAL)
;
;       PROC FVALUE     RETURNS A FUNCTIONVALUE OF AN
;                       ANALOG PROCESS
;                       ENTRY: B↑ STARTING ADDRESS OF X
;                       EXIT:  DE=FUNCTIONVALUE (REAL)
;
        CONST ¤BK=1,¤N=2
        BANK ¤BK
        SINGLE K↑10,I,IND,INT
        DOUBLE F,F1,FF,DF,Z,GSQ,AEPS,EPS1,EPS2,HF,HH
        DOUBLE ALPHA,DXNEW,DXOLD,XLAMB,ABSRTN
        DOUBLE X[¤N],W[¤N],G[¤N],RIKTN[¤N],OLDRTN[¤N]
        SET BANK ¤BK
        SET STACK 2:255
        GLOBAL MULT,DIV,FLOAT,IFIX,FMULT,FDIV
        GLOBAL FSUB,FADD,FVALUE

        DX X[0]=0131401;        =-1.203
        DX X[1]=040001;         =1.0
        IND=0
        DX XLAMB=040001;        =1.0
        DX DXOLD=040001;        =1.0
        DX HF=051372;           =0.01
        DX EPS1=041367;         =0.101E-02
        DX EPS2=041367;         =0.101E-02

        CALL FVALUE(B↑X)
        DX F=DE;            F=F(X)
        CALL FDIV(BC=DE,DE=040002)
        DX DF=DE;           DF=F/2.0
;
;       ITERATION PROCEDURE STARTS
;
  L10:  K=0
  L15:  DX FF=F
;
;       CALCULATE THE DESCENT DIRECTION
;
;       NUMERICAL CALCULATION OF THE GRADIENT VECTOR
;
        I=¤N
        REPEAT
          I=I-1
          CALL FADD(BC=X[I],DE=HF)
          DX X[I]=DE;     X(I)=X(I)+HF
          CALL FVALUE(B↑X)
          DX F1=DE;       F1=F(X)
          CALL FADD(BC=HF,DE=BC)
          DX HH=DE;       HH=HF+HF
          D=-D
          CALL FADD(BC=X[I],DE=DE)
          DX X[I]=DE;     X(I)=X(I)-HH
          CALL FVALUE(B↑X)
          D=-D
          CALL FADD(BC=F1,DE=DE)
          CALL FDIV(BC=DE,DE=HH)
          DX G[I]=DE;     G(I)=(F1-F(X))/HH
          CALL FADD(BC=X[I],DE=HF)
```

```
      DX X[I]=DE;    X(I)=X(I)+HF
    UNTIL I=0

    DX DXNEW=0
    I=ON
    REPEAT
      I=I-1
      CALL FMULT(BC=G[I],DE=BC)
      CALL FADD(BC=DXNEW,DE=DE)
      DX DXNEW=DE;   DXNEW=DXNEW+G(I)*G(I)
    UNTIL I=0
    CALL FDIV(BC=DE,DE=DXOLD)
    DX ALPHA=DE;     ALPHA=DXNEW/DXOLD
    IF K=0 THEN
      DX ALPHA=0
    END
    DX DXOLD=DXNEW
    I=ON
    REPEAT
      I=I-1
      CALL FMULT(BC=ALPHA,DE=OLDRTN[I])
      DREG BC=G[I]
      B=-B
      CALL FADD(BC=BC,DE=DE)
      DX RIKTN[I]=DE;        RIKTN(I)=-G(I)+ALPHA*OLDRTN(I)
      DX OLDRTN[I]=DE;       OLDRTN(I)=RIKTN(I)
    UNTIL I=0


    DX Z=0
    DX GSO=0
    I=ON
    REPEAT
      I=I-1
      DREG BC=RIKTN[I]
      IF 0>B THEN
        B=-B
      END
      DX ABSRTN=BC
      CALL FSUB(BC=BC,DE=Z)
      IF 0<=D THEN
        DX Z=ABSRTN;         Z WILL BE MAX(ABS(RIKTN(I)))
      END
      CALL FMULT(BC=G[I],DE=RIKTN[I])
      CALL FADD(BC=GSO,DE=DE)
      DX GSO=DE;     GSO=GSO+G(I)*RIKTN(I)
    UNTIL I=0
    CALL FDIV(BC=EPS1,DE=Z)
    DX AEPS=DE;      AEPS=EPS1/Z
    DREG DE=GSO
    GOTO L52 IF 0>D;         GOTO L52 IF GSO<0
    D=-D
    DX GSO=DE;       GSO=-GSO
    I=ON
    REPEAT
      I=I-1
      DREG DE=RIKTN[I]
      D=-D
      DX RIKTN[I]=DE;        RIKTN(I)=-RIKTN(I)
      DX OLDRTN[I]=DE;       OLDRTN(I)=RIKTN(I)
    UNTIL I=0
;
;       MINIMIZE ALONG THE DESCENT DIRECTION
;
```

```
L52:  IF IND>0 THEN
        CALL FLOAT(DE=IND)
        CALL FDIV(BC=0140002,DE=DE)
        CALL FMULT(BC=DF,DE=DE)
        CALL FDIV(BC=DE,DE=GSO)
        DX XLAMB=DE;  XLAMB=-2.0*DF/GSO/IND
      END
      INT=0
L55:  I=□N
      REPEAT
        I=I-1
        CALL FMULT(BC=XLAMB,DE=RIKTN[I])
        CALL FADD(BC=X[I],DE=DE)
        DX W[I]=DE;   W(I)=X(I)+XLAMB*RIKTN(I)
      UNTIL I=0
      CALL FVALUE(B↑W)
      DX F1=DE;       F1=F(W)
      D=-D
      CALL FADD(BC=F,DE=DE)
      GOTO L70 IF 0>D;                GOTO L70 IF F1>F
      DX F=F1
      GOTO L87 IF INT=2
      INT=1
      I=□N
      REPEAT
        I=I-1
        DX X[I]=W[I]
      UNTIL I=0
      CALL FADD(BC=XLAMB,DE=BC)
      DX XLAMB=DE;    XLAMB=2.0*XLAMB
      GOTO L55
L70:  GOTO L90 IF INT=1
      CALL FMULT(BC=040000,DE=XLAMB)
      DX XLAMB=DE;    XLAMB=0.5*XLAMB
      INT=2
      CALL FSUB(BC=XLAMB,DE=AEPS)
      GOTO L100 IF D<=0;      GOTO L100 IF XLAMB<=AEPS
      GOTO L55
L87:  I=□N
      REPEAT
        I=I-1
      DX X[I]=W[I]
      UNTIL I=0
      DREG DE=F
      D=-D
L90:  CALL FADD(BC=FF,DE=DE)
      DX DF=DE;       DF=FF-F
      CALL FSUB(BC=EPS2,DE=DE)
      GOTO L100 IF D>0;       GOTO L100 IF DF<EPS2
;
;     TEST FOR RECYCLING
;
      GOTO L10 IF K=□N
      K=K+1
      GOTO L15
L100:
      FINISH
```

```
                        ;MAIN PROGRAM
;
;       PROGRAM FOR FINDING THE MINIMUM OF A FUNCTION F(X)
;       WHEN ONLY FUNCTIONVALUES ARE AVAILABLE, BY
;       THE METHOD ABSOLUTE BIAS.
;       AUTHOR: I.SIXTENSSON  1976-04-29
;       REFERENCE: J.-J.OSTERGAARD,DYNAMIC VERSION OF
;       DIRECT OPTIMIZATION, PUBLICATION NO. 7509,
;       TECHNICAL UNIVERSITY OF DENMARK
;
;       PARAMETERS
;
;       IS      NUMBER OF SUCCEDING EXPERIMENTS THAT
;               WHERE ALL SUCCESSFUL
;       KN      NUMBER OF SUCCEDING EXPERIMENTS THAT
;               WHERE ALL FAILURES
;       IR      TAKES THE VALUES 0,1,2 DEPENDING ON
;               THE RESULT OF THE LAST EXPERIMENT
;       ISMAX   WHEN IS>ISMAX THEN THE STEPLEGTH
;               IS PROBABLY TOO SMALL. THE STEPLENGTH
;               WILL THEN BE DOUBLED AS LONG AS THE
;               EXPERIMENT IS SUCCESSFUL
;       KNMAX   WHEN KN=KNMAX THEN KL WILL BE
;               INCREMENTED AND SIGMA WILL BE REDUCED
;       KL      COUNTER
;       KLAR    TERMINATION WHEN KL=KLAR
;       K       K=0 WILL GIVE A RESTART OF THE PROCEDURE
;       SIKO    THE FACTOR BY WHICH SIGMA IS REDUCED
;       QM      THE BEST FUNCTIONVALUE SO FAR
;       QK      THE FUNCTIONVALUE TO BE EXAMINED
;       UM      COORDINATES CORRESPONDING TO QM
;       UK      COORDINATES CORRESPONDING TO QK
;       EPS     THE REDUCTION OF THE FUNCTIONVALUE
;               WILL BE AT LEAST EPS
;       IX      A RELATIVE LARGE,ODD STARTING NUMBER FOR
;               THE RANDOM PROCESS
;       SIGMA   STANDARD DEVIATION
;
;       SUBROUTINES USED
;
;       PROC MULT       MULTIPLICATION
;                       ENTRY: BC=N (INTEGER,POSITIVE)
;                              DE=K (INTEGER,POSITIVE)
;                       EXIT:  DE=N*K (INTEGER)
;
;       PROC DIV        DIVISION WITH ROUNDING
;                       ENTRY: BC=N (INTEGER,POSITIVE)
;                              DE=K (INTEGER,POSITIVE)
;                       EXIT:  DE=N/K (INTEGER)
;
;       PROC FLOAT      CONVERTING AN INTEGER INTO
;                       A REAL NUMBER
;                       ENTRY: DE=N (INTEGER)
;                       EXIT:  DE   (REAL)
;
;       PROC IFIX       CONVERTING A REAL NUMBER INTO
;                       AN 8-BITS  INTEGER
;                       ENTRY: DE=X (REAL)
;                       EXIT:  E    (INTEGER)
;
;       PROC FMULT      FLOATING MULTIPLICATION
;                       ENTRY: BC=X (REAL)
;                              DE=Y (REAL)
```

```
;                       EXIT:  DE=X*Y (REAL)
;
;       PROC FDIV       FLOATING DIVISION
;                       ENTRY: BC=X (REAL)
;                              DE=Y (REAL)
;                       EXIT:  DE=X/Y (REAL)
;
;       PROC FSUB       FLOATING SUBTRACTION
;                       ENTRY: BC=X (REAL,POSITIVE)
;                              DE=Y (REAL,POSITIVE)
;                       EXIT:  DE=X-Y (REAL)
;
;       PROC FADD       FLOATING ADDITION
;                       ENTRY: BC=X (REAL)
;                              DE=Y (REAL)
;                       EXIT:  DE=X+Y (REAL)
;
;       PROC FVALUE     RETURNS THE FUNCTIONVALUE OF
;                       AN ANALOG PROCESS
;                       ENTRY: B↑ STARTING ADDRESS OF X
;                       EXIT:  DE=FUNCTIONVALUE (REAL)
;
        CONST □BK=1,□N=2
        BANK □BK
        SINGLE K↑10,I,KK,KL,KN,KLAR,IR,IS,IX
        SINGLE ISMAX,KNMAX
        DOUBLE AA,QM,QK,SIGMA,SIKO,EPS,XLAMB
        DOUBLE UM[□N],UK[□N],DU[□N]
        SET BANK □BK
        SET STACK 2:255
        GLOBAL MULT,DIV,FLOAT,IFIX,FMULT,FDIV
        GLOBAL FSUB,FADD,FVALUE

        DX UK[0]=0131401;       =-1.203
        DX UK[1]=040001;        =1.0
        IS=0
        IR=0
        K=0
        KN=0
        KL=1
        KLAR=3
        ISMAX=10
        KNMAX=40
        IX=5719
        DX SIGMA=040000;        =0.5
        DX SIKO=063675;         =0.1
        DX EPS=041367;          =0.101E-02
;
;       ITERATION PROCEDURE STARTS
;
  L10:  CALL FVALUE(B↑UK)
        DX QK=DE;       QK=F(UK)
        GOTO L20 IF K/=0
        I=□N
        REPEAT
          I=I-1
          DX UM[I]=UK[I]
        UNTIL I=0
        DX QM=QK
        K=1
        GOTO L30
  L20:  DREG DE=QK
        D=-D
        CALL FADD(BC=QM,DE=DE)
```

```
        CALL FADD(BC=EPS,DE=DE)
        GOTO L60 IF 0>D;          GOTO L60 IF QK<QM-EPS
;
;               FAILURE
;
        DX XLAMB=040001;          =1.0
        GOTO L40 IF IS=0
;
;       GENERATES AN ARRAY OF NORMALLY DISRIBUTED
;       PSEUDO-RANDOM REAL NUMBERS WITH THE MEAN VALUE 0
;       AND STANDARD DEVIATION SIGMA
;
  L30:  I=@N
        REPEAT
          I=I-1
          DX AA=0
          KK=12
          REPEAT
            KK=KK-1
            CALL MULT(BC=IX,DE=899)
            IF 0>D THEN
              DX DE=DE+32767+1
            END
            DX IX=DE
            CALL FLOAT(DE=DE)
            CALL FDIV(BC=DE,DE=040020); 040020=32768
            CALL FADD(BC=AA,DE=DE)
            DX AA=DE;    AA=AA+IX/32768
          UNTIL KK=0
          CALL FSUB(BC=AA,DE=060003);    060003=6
          CALL FMULT(BC=SIGMA,DE=DE)
          DX DU[I]=DE;  DU[I]=(AA-6.0)*SIGMA
        UNTIL I=0

        IS=0
        IR=0
        GOTO L70
  L40:  KN=KN+1
        IR=IR+1
        GOTO L50 IF KNMAX>KN
        GOTO L80 IF KL=KLAR
        KL=KL+1
        KN=0
        CALL FMULT(BC=SIKO,DE=SIGMA)
        DX SIGMA=DE;     SIGMA=SIKO*SIGMA
  L50:  GOTO L30 IF IR=2
        I=@N
        REPEAT
          I=I-1
          DREG DE=DU[I]
          D=-D
          DX DU[I]=DE
        UNTIL I=0
        GOTO L70
;
;       SUCCESS
;
  L60:  I=@N
        REPEAT
          I=I-1
          DX UM[I]=UK[I]
        UNTIL I=0
        DX QM=QK
        IS=IS+1
```

```
        KN=0
        IF IS>ISMAX THEN
          CALL FADD(BC=XLAMB,DE=BC)
          DX XLAMB=DE;  XLAMB=2.0*XLAMB
        END
L70:    I=ON
        REPEAT
          I=I-1
          CALL FMULT(BC=XLAMB,DE=DU[I])
          CALL FADD(BC=UM[I],DE=DE)
          DX UK[I]=DE;  UK(I)=UM(I)+XLAMB*DU(I)
        UNTIL I=0
        GOTO L10
L80:
        FINISH
```

```
    PROC MULT
                        ;THE PROCEDURE MULTIPLIES TWO POSITIVE
                        ;16-BITS INTEGERS N AND K, THE RESULT
                        ;WILL BE AN INTEGER,
                        ;IF N<K THEN THE PROCEDURE WILL BE
                        ;OPTIMAL. **WARNING** THE RESULT
                        ;MUST BE LESS THAN 65536.
                        ;SUBROUTINES USED: NONE
                        ;CALL IS: CALL MULT(BC=N,DE=K)
                        ;RESULT IN: DE
                        ;
      GLOBAL MULT

      DREG HL=DE
      DREG DE=0
LM1:  RETURN IF 0=B.OR C
      .XCHG;            EXCHANGE (HL) AND (DE)
      .ORA A;           CLEARS CARRY
      B=B.RAR;          BC WILL BE SHIFTED ONE STEP
      C=C.RAR;          RIGHT THROUGH CARRY
      GOTO LM2 IF CARRY FALSE
      DX HL=HL+DE
LM2:  .XCHG
      DX HL=HL+HL
      GOTO LM1
    ENDPROC
    FINISH
```

```
        PROC DIV
                            ;THE PROCEDURE DIVIDES TWO POSITIVE
                            ;16-BITS INTEGERS N AND K,THE RESULT
                            ;WILL BE AN INTEGER, ROUNDED
                            ;SUBROUTINES USED: NONE
                            ;CALL IS: CALL DIV(BC=N,DE=K)
                            ;RESULT IN: DE
                            ;
          GLOBAL DIV

          DREG DE=-DE
          DREG HL=0
          REG A=17
    LD1:  PUSH PSW
          PUSH HL
          DX HL=HL+DE
          GOTO LD2 IF CARRY FALSE
          .XTHL;            EXCHANGE (HL) AND STACK TOP
    LD2:  POP HL
          C=C.RAL;          HLBC WILL BE SHIFTED ONE STEP
          B=B.RAL;          LEFT THROUGH CARRY
          L=L.RAL;
          H=H.RAL;
          POP PSW
          REG A=A-1
          GOTO LD1 IF 0/=A
;               ROUNDING PROCEDURE STARTS
;               BC  CONTAINS THE RESULT
;               HL  CONTAINS  2*REMAINDER
;               DE  CONTAINS THE DIVISOR NEGATED
          DX HL=HL+DE;  MEANS  HL=2*REM-DIV
          DREG DE=BC
          RETURN IF 0>H
          DREG DE=DE+1
        ENDPROC
        FINISH
```

```
PROC IFIX
                    ;THE PROCEDURE CONVERTS A REAL
                    ;NUMBER X INTO AN 8-BITS INTEGER
                    ;IF X>127 THEN E=127, IF X<-128
                    ;THEN E=-128, IF X<0.5 AND X>=-0.5
                    ;THEN E=0,
                    ;SUBROUTINES USED: MULT,DIV
                    ;CALL IS: CALL IFIX(DE=X)
                    ;RESULT IN: E
                    ;
  BANK 1
  SINGLE I
  GLOBAL MULT DIV
  GLOBAL IFIX

  IF 0>E THEN
    REG E=0
    RETURN
  END
  IF E>7 THEN
    IF 0>D THEN
      REG E=-128
    ELSE
      REG E=127
    END
    RETURN
  END
  I=1
  IF 0>D THEN
    D=-D
    I=2
  END
  REG L=1
  REPEAT
    L=L.RLC;     ROTATE LEFT
    REG E=E-1
  UNTIL E=0
  CALL MULT(B=0,C=L,E=D,D=B)
  CALL DIV(BC=DE,DE=128)
  IF I=2 THEN
    E=-E
  END
ENDPROC
FINISH
```

```
PROC FLOAT
                    ;THE PROCEDURE CONVERTS A 16-BITS
                    ;INTEGER N INTO A REAL NUMBER, WITH
                    ;8 BITS OF MANTISSA IN  D  AND
                    ;8 BITS OF CHARACTERISTIC IN  E.
                    ;SUBROUTINES USED: NONE
                    ;CALL IS: CALL FLOAT(DE=N)
                    ;RESULT IN: DE
                    ;
  BANK 1
  SINGLE I,LO
  GLOBAL FLOAT

  RETURN IF 0=D.OR E
  I=1
  LO=16
  IF 0>D THEN
    DREG DE=-DE
    I=2
  END
  WHILE 0<=D DO
    .ORA A;       CLEARS CARRY
    E=E.RAL
    D=D.RAL
    LO=LO-1
  ENDWHILE;       SHIFT LEFT UNTIL SIGN FLAG=1
  REG B=D
  REG D=D+1;      GIVES ROUNDING
  REG C=0100;     IF C MANTISSA THEN C=0.5
  .ORA A
  D=D.RAR
  IF I=2 THEN
    D=-D
    REG C=0300;   IF C MANTISSA THEN C=-0.5
  END
  IF B>0376 THEN; 0376=1111 1110
    REG D=C
    REG E=LO+1
  ELSE
    REG E=LO
  END
ENDPROC
FINISH
```

```
PROC FMULT
                    ;THE PROCEDURE MULTIPLIES TWO REAL
                    ;NUMBERS X AND Y OF STANDARDFORM
                    ;AND RETURNS THE REAL NUMBER RESULT
                    ;IN STANDARDFORM
                    ;SUBROUTINES USED: MULT
                    ;CALL IS: CALL FMULT(BC=X,DE=Y)
                    ;RESULT IN: DE
                    ;
  BANK 1
  SINGLE I,LO
  GLOBAL MULT
  GLOBAL FMULT

  I=0
  IF 0>B THEN
    B=-B
    I=I+1
  END
  IF 0>D THEN
    D=-D
    I=I+1
  END
  LO=2+C+E
  CALL MULT(C=B,B=0,E=D,D=R)
  RETURN IF 0=D.OR E
  WHILE 0<=D DO
    .ORA A;       CLEARS CARRY
    E=E.RAL
    D=D.RAL
    LO=LO-1
  ENDWHILE;       SHIFT LEFT UNTIL SIGN FLAG=1
  REG B=D
  REG D=D+1;       GIVES ROUNDING
  REG C=0100;     IF C MANTISSA THEN C=0.5
  .ORA A
  D=D.RAR
  IF I=1 THEN
    D=-D
    REG C=0300;   IF C MANTISSA THEN C=-0.5
  END
  IF B>0376 THEN; 0376=1111 1110
    REG D=C
    REG E=LO+1
  ELSE
    REG E=LO
  END
ENDPROC
FINISH
```

```
PROC FDIV
                    ;THE PROCEDURE DIVIDES TWO REAL
                    ;NUMBERS X AND Y OF STANDARDFORM,
                    ;AND RETURNS THE REAL NUMBER RESULT
                    ;IN STANDARDFORM
                    ;SUBROUTINES USED: DIV
                    ;CALL IS: CALL(BC=X,DE=Y)
                    ;RESULT IN: DE
                    ;
  BANK 1
  SINGLE I,LO
  GLOBAL DIV
  GLOBAL FDIV

  I=0
  LO=C-E+8
  IF 0>B THEN
    B=-B
    I=I+1
  END
  IF 0>D THEN
    D=-D
    I=I+1
  END
  CALL DIV(B=B,C=0,E=D,D=C)
  RETURN IF 0=B,OR C
  WHILE 0<=D DO
    .ORA A;       CLEARS CARRY
    E=E.RAL
    D=D.RAL
    LO=LO-1
  ENDWHILE;       SHIFT LEFT UNTIL SIGN FLAG=1
  REG B=D
  REG D=D+1;       GIVES ROUNDING
  REG C=0100;     IF C MANTISSA THEN C=0,5
  .ORA A
  D=D.RAR
  IF I=1 THEN
    D=-D
    REG C=0300;   IF C MANTISSA THEN C=-0,5
  END
  IF B>0376 THEN; 0376=1111 1110
    REG D=C
    REG E=LO+1
  ELSE
    REG E=LO
  END
ENDPROC
FINISH
```

```
      PROC FSUB
                        ;THE PROCEDURE SUBTRACTS TWO POSITIVE
                        ;REAL NUMBERS X AND Y OF STANDARD
                        ;FORM, AND RETURNS THE REAL NUMBER
                        ;RESULT X-Y IN STANDARD FORM.
                        ;SUBROUTINES USED: NONE
                        ;THE CALL IS: CALL FSUB(BC=X,DE=Y)
                        ;RESULT IN: DE
                        ;
        BANK 1
        SINGLE I,DIFF*2
        GLOBAL FSUB

        IF B=0 THEN
          D=-D
          RETURN
        END
        IF D=0 THEN
          DREG DE=BC
          RETURN
        END
        DIFF=C-E;       DIFFERENCE IN CHARACTERISTICS
        IF 0<=DIFF THEN
          REG E=C
          REG A=B;      B IS EXCHANGED WITH
          REG B=D;      D USING A AS TEMPORARY
          REG D=A;      STORAGE
          DIFF=-DIFF
        END
        GOTO LS1 IF DIFF=0
        DIFF=DIFF+1
        WHILE 0>DIFF DO
          DIFF=DIFF+1
          .ORA A;       CLEARS CARRY
          B=B.RAR
        ENDWHILE
        REG B=B+1;      GIVES ROUNDING
        .ORA A
        B=B.RAR
LS1:    IF E=C THEN
          D=D-B
        ELSE
          D=B-D
        END
        RETURN IF D=0
        I=1
        IF 0>D THEN
          D=-D
          I=2
        END
        WHILE 0<=D DO
          .ORA A
          D=D.RAL
          REG E=E-1
        ENDWHILE
        .ORA A
        D=D.RAR
        IF I=2 THEN
          D=-D
        END
        REG E=E+1
      ENDPROC
      FINISH
```

```
      PROC FADD
                          ;THE PROCEDURE ADDS TWO REAL NUMBERS
                          ;X AND Y OF STANDARDFORM, AND
                          ;RETURNS THE REAL NUMBER RESULT X+Y
                          ;IN STANDARDFORM
                          ;SUBROUTINES USED: FSUB
                          ;CALL IS: CALL FADD(BC=X,DE=Y)
                          ;RESULT IN: DE
                          ;
        BANK 1
        SINGLE I,DIFF↑2
        GLOBAL FSUB
        GLOBAL FADD

        RETURN IF B=0
        IF D=0 THEN
          DREG DE=BC
          RETURN
        END
        I=1
        IF 0>B THEN
          IF 0>D THEN
            I=2
            B=-B
            D=-D
            GOTO LA1
          END
          H=-B
          REG L=C
          CALL FSUB(BC=DE,DE=HL)
          RETURN
        END
        IF 0>D THEN
          D=-D
          CALL FSUB(BC=BC,DE=DE)
          RETURN
        END
LA1:    DIFF=C-E;       DIFFERENCE IN CHARACTERISTICS
        IF 0<=DIFF THEN
          REG E=C
          REG A=B;      B IS EXCHANGED WITH
          REG B=D;      D USING A AS TEMPORARY
          REG D=A;      STORAGE
          DIFF=-DIFF
        END
        GOTO LA2 IF DIFF=0
        DIFF=DIFF+1
        WHILE 0>DIFF DO
          DIFF=DIFF+1
          .ORA A;       CLEARS CARRY
          B=B.RAR
        ENDWHILE
        REG B=B+1;      GIVES ROUNDING
        .ORA A
        B=B.RAR
LA2:    D=D+B
        IF 0>D THEN
          REG D=D+1
          .ORA A
          D=D.RAR
          REG E=E+1
        END
        IF I=2 THEN
```

```
      D=-D
    END
ENDPROC
FINISH
```

```
PROC FVALUE
                    ;THE PROCEDURE RETURNS THE FUNCTION-
                    ;VALUE F(X) OF AN ANALOG PROCESS
                    ;WITH TWO VARIABLES
                    ;WHERE X IS AN ARRAY OF 2 ELEMENTS
                    ;SUBROUTINES USED: IFIX,FLOAT
                    ;THE CALL IS: CALL FVALUE(B↑X)
                    ;RESULT IN: DE
                    ;
  BANK 1
  SINGLE ADR↑3
  GLOBAL IFIX,FLOAT
  GLOBAL FVALUE

  ADR=B
  REG E=↑B;       L=B, E WILL BE LOADED FROM MEMORY
  REG D=↑B+1;     L=B+1, D WILL BE LOADED FROM MEMORY
  CALL IFIX(DE=DE)
  OUTPUT(1)=E
  B=ADR+2
  SET BANK 1
  REG E=↑B
  REG D=↑B+1
  CALL IFIX(DE=DE)
  OUTPUT(2)=E
  E=INPUT(1)
  CALL FLOAT(D=0,E=E)
ENDPROC
FINISH
```

TESTPH 00:56:03 05/21/76

```
100 $0=1 $1=1
110 /* TESTPROGRAM FOR COMPARISON ASSEMBLER CODE
120    GENERATED BY PL/M AND HELP.
130    THIS PROGRAM IS ONLY A PART OF A LARGER PROGRAM */
140
150 DECLARE (X,G)(3) BYTE;
160 DECLARE (I,K,F,F1,HH,DXNEW,DXOLD,ALPHA) BYTE;
170
180 DO I=0 TO 2;
190   X(I)=X(I)+HH;
200   F1=X(0)*X(1)+X(2)*X(2);
210   G(I)=(F1-F)/HH;
220   X(I)=X(I)-HH;
230 END;
240 DXNEW=0;
250 DO I=0 TO 2;
260   DXNEW=DXNEW+G(I)*G(I);
270 END;
280 ALPHA=DXNEW/DXOLD;
290 IF K=0 THEN ALPHA=0;
300 DXOLD=DXNEW;
310 EOF;
```

```
;TESTPROGRAM FOR COMPARISON ASSEMBLER CODE GENERATED
;BY PL/M AND HILP.
;THIS PROGRAM IS ONLY A PART OF A LARGER PROGRAM.
;THIS HILP-PROGRAM IS WRITTEN AT THE HIGHEST POSSIBLE
;LEVEL. NOTHING HAS BEEN DONE TO OPTIMATE THE CODE.
;
;SUBROUTINES USED
;
;       PROC MULT       MULTIPLICATION
;                       ENTRY: B=N (INTEGER)
;                              C=K (INTEGER)
;                       EXIT:  B=N*K (INTEGER)
;
;       PROC DIV        DIVISION
;                       ENTRY: B=N (INTEGER)
;                              C=K (INTEGER)
;                       EXIT:  B=N/K (INTEGER)
;REGISTERS AFFECTED
;
        BANK 1
        SINGLE X[3],G[3]
        SINGLE I,K,F,F1,HH,DXNEW,DXOLD,ALPHA
        SET BANK 1
        SET STACK 2:255
        GLOBAL MULT,DIV

        I=0
        REPEAT
          X[I]=X[I]+HH
          CALL MULT(B=X[0],C=X[1])
          F1=B; TEMPORARY STORAGE
          CALL MULT(B=X[2],C=B)
          F1=F1+B;      F1=X[0]*X[1]+X[2]*X[2]
          B=F1-F
          CALL DIV(B=B,C=HH)
          G[I]=B
          X[I]=X[I]-HH
          I=I+1
        UNTIL I=3
        DXNEW=0
        I=0
        REPEAT
          CALL MULT(B=G[I],C=B)
          DXNEW=DXNEW+B
          I=I+1
        UNTIL I=3
        CALL DIV(B=DXNEW,C=DXOLD)
        ALPHA=B
        IF K=0 THEN
          ALPHA=0
        END
        DXOLD=DXNEW
        FINISH
```

```
TEST2        18:55EDT     05/12/76

100 $0=1 $1=1
110 FLOAT:PROCEDURE(X) ADDRESS;
120
130 /*THE PROCEDURE CONVERTS AN ADDRESS VARIABLE X INTO*/
140 /*A REAL NUMBER WITH 8 BITS OF MANTISSA AND 8 BITS */
150 /*OF CHARACTERISTIC. THE RESULT IS AN ADDRESS-     */
160 /*VARIABLE WHERE THE 8 MOST SIGNIFICANT BITS ARE   */
170 /*THE MANTISSA, AND THE 8 LEAST SIGNIFICANT BITS   */
180 /*ARE THE CHARACTERISTIC                           */
190
200     DECLARE(X,MA) ADDRESS;
210     DECLARE(I,LO,HI) BYTE;
220     IF X=0 THEN GOTO R;
230     I=1; LO=16; MA=4000H;
240     IF X>32767 THEN DO;
250       X=-X; I=2;
260     END;
270     DO WHILE X <= 32767;
280       X=SHL(X,1); LO=LO-1;
290     END;
300 R:  HI=HIGH(SHR(X,1));
310     IF I=2 THEN DO; HI=-HI; MA=0C000H; END;
315     IF X>1111111101111111B THEN RETURN 1+LOW(MA);
320     RETURN LO+(256 AND X)+256*HI;
340 END FLOAT;
350 IFIX:PROCEDURE(X) BYTE;
360
370 /*THE PROCEDURE CONVERTS A REAL NUMBER X INTO AN   */
380 /*INTEGER, AND RETURNS THE RESULT IN A BYTE-        */
390 /*VARIABLE. THE PROCEDURE RETURNS 0 IF X<0.5        */
400 /*AND 255 IF X>=255                                 */
410
420     DECLARE X ADDRESS;
430     DECLARE (HI,LO) BYTE;
440     IF (HI:=HIGH(X))>127 OR (LO:=LOW(X))>127 THEN RETURN 0;
450     IF LO>8 THEN RETURN 255;
460     RETURN LOW((HI*SHL(01H,LO))/128);
470 END IFIX;
480 EOF;
```

```
PROC FLOAT
                    ;THE PROCEDURE CONVERTS A 16-BITS
                    ;INTEGER N INTO A REAL NUMBER, WITH
                    ;8 BITS OF MANTISSA IN  D  AND
                    ;8 BITS OF CHARACTERISTIC IN  E.
                    ;SUBROUTINES USED: NONE
                    ;CALL IS: CALL FLOAT(DE=N)
                    ;RESULT IN: DE
                    ;
  BANK 1
  SINGLE I,LO,MA
  GLOBAL FLOAT
  RETURN IF 0=D.OR E
  I=1
  LO=16
  MA=0100
  IF 0>D THEN
    DREG DE=-DE
    I=2
  END
  WHILE 0<=D DO
    .ORA A;      CLEARS CARRY
    E=E.RAL
    D=D.RAL
    LO=LO-1
  ENDWHILE
  REG B=D
  REG D=D+1;     GIVES ROUNDING
  .ORA A
  D=D.RAR
  IF I=2 THEN
    D=-D
    MA=0300;  MEANS MA=-MA
  END
  IF B>0376 THEN; 0376=1111 1110  B HAS
        ;THE VALUE OF D BEFORE SHIFTING RIGHT
    REG D=MA
    REG E=LO+1
  ELSE
    REG E=LO
  END
ENDPROC
FINISH
```

```
PROC IFIX
                    ;THE PROCEDURE CONVERTS A REAL
                    ;NUMBER X INTO AN 8-BITS INTEGER
                    ;IF X<0.5 THEN E=0. IF X>=-128
                    ;THEN E=-128
                    ;SUBROUTINES USED: MULT DIV
                    ;CALL IS: CALL IFIX(DE=X)
                    ;RESULT IN: E
                    ;
  GLOBAL MULT,DIV
  GLOBAL IFIX
  IF 0>D THEN
    REG E=0
    RETURN
  END
  IF 0>E THEN
    REG E=0
    RETURN
  END
  IF E>8 THEN
    REG E=-128
    RETURN
  END
  REG L=1
  REPEAT
    L=L.RLC
    REG E=E-1
  UNTIL E=0
  CALL MULT(B=0,C=L,E=D,D=B)
  CALL DIV(BC=DE,DE=128)
ENDPROC
FINISH
```

```
SORT          19:16EDT     05/12/76

100 $O=1 $I=1
110 SORT: PROCEDURE(PTR,N);
120
130 /* BUBBLESORT, SORTS A VECTOR LOCATED AT PTR, WITH
140     THE LENGTH N, INTO ASCENDING ORDER,
150     SWAP=(BOOLEAN) HAVE WE DONE ANY SWITCHING
160             YET ON THIS SCAN */
170
180 DECLARE PTR ADDRESS,Z BASED PTR BYTE;
190 DECLARE (I,N,SWAP,TEMP) BYTE;
200
210 SWAP=1; /* SWAP=TRUE MEANS NOT DONE YET */
220 DO WHILE SWAP;
230   SWAP=0;  /*BEGIN NEXT SCAN OF Z */
240   DO I=0 TO N-2;
250     IF Z(I)>Z(I+1) THEN
260     DO;  /* FOUND A PAIR OUT OF ORDER */
270       SWAP=1;  /* SWITCH THEM INTO ORDER */
280       TEMP=Z(I);
290       Z(I)=Z(I+1);
300       Z(I+1)=TEMP;
310     END;
320   END; /*HAVE NOW COMPLETED A SCAN */
330 END; /* WHILE */
340 /* HAVE NOW COMPLETED A SCAN WITH NO SWITCHING */
350 RETURN;
360 END SORT;
370 EOF;
```

```
;BUBBLESORT IS DIRECTLY TRANSLATED FROM A HIGH-LEVEL
;LANGUAGE INTO HILP. THE TRANSLATION TIME IS SHORT
;BUT A LOT OF CODE IS GENERATED
;
;
PROC SORT
;
;THE PROCEDURE WILL SORT A VECTOR WITH AT MOST
;50 ELEMENTS INTO ASCENDING ORDER ACCORDING TO BUBBLESORT.
;IN THE MAIN PROGRAM, ALL VECTORS WHICH ARE TO BE
;SORTED MUST BE LOCATED AT THE SAME LOW ADDRESS,
;NAMELY 200, BUT DIFFERENT BANK ADDRESSES.
;IT IS CONVENIENT TO SAVE THE LAST CELLS IN EVERY
;BANK. THESE CELLS CAN BE USED IN SUBROUTINES
;WHERE THE BANK IS UNDEFINED. IN THIS EXAMPLE
;CELL 250-255 IS NOT FOR NORMAL USE.
;
;ENTRY: B=VECTOR LENGTH
;       H=BANK WHERE VECTOR IS LOCATED
;EXIT:    -
;REGISTERS AFFECTED
;SUBROUTINES USED
;       NONE
;
        BANK UNDEFINED
        SINGLE Z[50]↑200,I,N,SWAP,TEMP
        GLOBAL SORT
;
;WHEN WE HAVE BANK UNDEFINED, THE USER MUST GUARANTEE
;THAT H HAS THE RIGHT BANK ADDRESS WHEN NEEDED.
;HERE H GETS THE RIGHT BANK ADDRESS IN THE CALL.
;THE SUBROUTINE DOES NOT CHANGE H-VALUE
;
        N=B
        SWAP=1
        WHILE SWAP=1 DO
          SWAP=0
          I=0
          REPEAT
            IF Z[I]>Z[I+1] THEN
              TEMP=Z[I]
              Z[I]=Z[I+1]
              Z[I+1]=TEMP
              SWAP=1
            END
            I=I+1
          UNTIL I=N-1
        ENDWHILE
ENDPROC
FINISH
```

```
;BUBBLESORT IS HERE ALMOST DIRECTLY TRANSLATED FROM
;HIGH-LEVEL LANGUAGE INTO HILP. REGISTER
;OPERATIONS HAS TAKEN CARE OF EXPRESSIONS WITH
;INDEX. THIS WILL GENERATE A MORE EFFECTIVE CODE
;
;
PROC SORT
;
;THE PROCEDURE WILL SORT A VECTOR INTO ASCENDING
;ORDER ACCORDING TO BUBBLESORT, WITHOUT MOVING
;THE VECTOR FROM ITS PLACE IN MEMORY.
;
;ENTRY: R=VECTOR LENGTH
;       C↑ VECTOR
;       H= BANK WHERE VECTOR IS LOCATED
;EXIT     -
;REGISTERS AFFECTED
;SUBROUTINES USED
;       NONE
;
        BANK UNDEFINED
        SINGLE I↑250,N,SWAP
        GLOBAL SORT
;
;WHEN WE HAVE BANK UNDEFINED, THE USER MUST GUARANTEE
;THAT H HAS THE RIGHT BANK ADDRESS WHEN NEEDED.
;HERE H GETS THE RIGHT BANK ADDRESS IN THE CALL.
;THE SUBROUTINE DOES NOT CHANGE  H-VALUE
;
        N=B
        SWAP=1
        WHILE SWAP=1 DO
          SWAP=0
          I=0
          REPEAT
            REG D=↑C;   L=C, D WILL BE LOADED FROM MEMORY
                      ; HL CONTAINS THE ADDRESS, D=Z[I]
            REG E=↑C+1; L=C, L=L+1 THEN E WILL
                      ; BE LOADED FROM MEMORY, E=Z[I+1]
            IF D>E THEN
              SWAP=1
              REG ↑C=E; L=C,   THEN THE VALUE
                      ; OF E IS MOVED TO MEMORY
              REG ↑C+1=D; L=C,  L=L+1 THEN THE VALUE
                      ; OF D IS MOVED TO MEMORY
              REG C=C+1
            END
            I=I+1
          UNTIL I=N-1
        ENDWHILE
ENDPROC
FINISH
```

```
;BUBBLESORT IS HERE PROGRAMMED IN HILP WITH EXTREM
;USE OF REGISTER OPERATIONS. REGISTERS ARE USED
;INSTEAD OF ADDRESS REFERENCES. *WARNING* THE USER
;MUST KNOW WHEN CERTAIN REGISTERS ARE DESTROYED
;E.G. DOUBLE REGISTER STATEMENTS CAN DESTROY
;THE REGISTERPAIR HL
;
;
PROC SORT
;
;THE PROCEDURE WILL SORT A VECTOR INTO ASCENDING
;ORDER ACCORDING TO BUBBLESORT, WITHOUT MOVING
;THE VECTOR FROM ITS PLACE IN MEMORY.
;
;ENTRY: B= VECTOR LENGTH
;       L↑ VECTOR
;       H= BANK WHERE VECTOR IS LOCATED
;EXIT:   -
;REGISTERS AFFECTED
;SUBROUTINES USED
;       NONE
;
        GLOBAL SORT
;
;H GETS THE RIGHT BANK ADDRESS IN THE CALL
;THE SUBROUTINE DOES NOT CHANGE H-VALUE
;
        REG C=1;  C=SWAP
        WHILE C=1 DO
          REG C=0
          REG A=1; A IS USED AS COUNTER
          REPEAT
            REG D=↑L; D WILL BE LOADED FROM MEMORY
                    ; D=Z(I)
            REG E=↑L+1; E WILL BE LOADED FROM MEMORY
                    ; E=Z(I+1)
            IF D>E THEN
              REG C=1
              REG ↑L-1=E; THE VALUE OF E IS
                        ; MOVED TO MEMORY
                        ; AFTER L=L-1
              REG ↑L+1=D; THE VALUE OF D IS
                        ; MOVED TO MEMORY
                        ; AFTER L=L+1
            END
            REG A=A+1
          UNTIL B=A; THE LOOP IS DONE N-1 TIMES
        ENDWHILE
ENDPROC
FINISH
```