CODEN:LUTFD2/(TFRT-5193)/01-028/(1977)

IMPLEMENTERING AV EXTREMALSÖKANDE ALGORITM PÅ MIKRODATOR

GUNNAR WIKTORSSON

Lunds Tekniska Högskola
Inst. för Reglerteknik
Mars 1977

IMPLEMENTERING AV EXTREMALSÖKANDE ALGORITM PÅ MIKRODATOR

AV

GUNNAR WIKTORSSON

Examensarbete i reglerteknik

Institutionen för reglerteknik
Lunds Tekniska Högskola

Handledare: Björn Wittenmark

Dokumentutgivare
04T0 Lund Institute of Technology

Handläggare
06T0 Dept of Automatic Control
Björn Wittenmark

Författare
08T0 Gunnar Wiktorsson

Dokumentnamn
04T4 REPORT

Utgivningsdatum
04T6 March 1977

Dokumentbeteckning
04T6 LUTFD2/(TFRT-5193)/01-28/(1977)

Ärendebeteckning
06T6

10T4

Dokumenttitel och undertitel
18T0 Implementering av extremalsökande algoritm på mikrodator (Implementation of an extremal searching algorithm on a micro computer)

Referat (sammandrag)
20T0 The report describes the implementation and the tests of a Fletcher-Reeves method for finding the extremal value of a function. The implementation is done on an Intel 8080 micro-computer. The high-level programming language HILP has been used with a PDP-15 as a host-computer. Memory required is 120 bytes data memory, and 2.5 K bytes program memory. The method has been tested on internally generated functions as well as on functions generated on an analogue computer

Referat skrivet av
Björn Wittenmark

Förslag till ytterligare nyckelord
44T0

Klassifikationssystem och -klass(er)
50T0

Indextermer (ange källa)
52T0

Omfång
28 58T0

Övriga bibliografiska uppgifter
56T2

Språk
Swedish

Sekretessuppgifter
60T0

ISSN
60T4

ISBN
60T6

Dokumentet kan erhållas från
62T0

Mottagarens uppgifter
62T4

Pris
66T0

SIS-DB 1 DOKUMENTDATABLAD enligt SIS 62 10 12

Blankett LU 11:25 1976–07

FÖRORD

Föreliggande examensarbete utgör en fortsättning på Inge Sixtenssons examensarbete med samma titel. Hans arbte bestod i att testa två lämpliga algoritmer på minidator och välja ut en av dem för vidare testning på mikrodatorn. Utgångsläget för mig var att det fanns ett program för algoritmen skriven i HILP. Mitt arbete har sedan varit att praktiskt testa programmet på mikrodatorn. Tyvärr fick programmet inte i sin ursprungliga form plats i RAM-minnet, vilket förutsattes av Inge. Efter hand har därför delar av programmet fått läggas i PROM:ar, vilket ju också är den normala arbetsgången vid programmering av mikrodatorer.

Jag vill här även tacka min handledare för god handledning av arbetet.

Sännö i januari 1977

Gunnar Wiktorsson

INNEHÅLL

1 SAMMANFATTNING

Ändamålet med detta examensarbete är att implementera och testa Fletcher-Reeves extremalsökande algoritm på mikrodatorn Intel 8080. Algoritmen har förut testats på minidatorn PDP-15 i ett föregående examensarbete. Minnesbehovet är 120 byte dataminne och 2.5K byte programminne. Algoritmen har här testats på en internt genererad funktion: "Rosenbrock's curved valley". Vissa försök har även gjorts på en yttre funktion uppkopplad på analogimaskin. Programmeringsspråket HILP har använts med en PDP-15 som värddator.

ABSTRACT

The main purpose of this master thesis is to implement and test on a microcomputer Intel 8080 Fletcher-Reeves method for finding the extremal value of a function. The program has beentested on a minicomputer PDP-15 in a preceeding master thesis. Memory required is 120 bytes data memory and 2.5K bytes program memory. The method has here been tested on an internally generated function: "Rosenbrock's curved valley". Some tests have been done on an outer function generated by an analog computer. The high-level programming language HILP has been used with a PDP-15 as a host-computer.

2 INLEDNING

Genom att formulera syntesproblem som optimeringsproblem, kan många av variationskalkylens kraftfulla resultat tillämpas för att dimensionera reglersystem. Ett programpaket för dimensionering av regulatorer och styrlagar för linjära system, både diskreta och kontinuerliga, har utvecklats vid institutionen för reglerteknik, LTH. Dimensioneringen kan t. ex bestå i att välja lämpliga parametrar (integrationstid, förstärkning) till en PI-regulator med kvadratiska felsignalen som kriteriefunktion.

Ett stort antal extremalsökande metoder finns att tillgå. En klass av metoder, sökmetoder, väljer mer eller mindre slumpmässigt ut nya punkter och undersöker om dessa punkter är lyckosamma eller inte. Dessa metoder har ofta utvecklats ur heuristiska resonemang och är ofta mindre effektiva. Andra metoder grundar sig på beräkning av derivator i kombination med mer eller mindre sofistikerade metoder att välja steglängd.

Den här använda metoden, Fletcher-Reeves, är en metod som baserar sig på konjugerade riktningar och utnyttjar således derivator. I ett föregående examensarbete här Inge Sixtensson förordat denna metod i konkurrens med en sökmetod, Absolute bias, där sökriktningen får av en slumpvektor. Fletcher-Reeves metod visade sig vara effektivare än denna sökmetod. Ovannämnde står även som författare till det program som utnyttjas i detta examensarbete. Programmet är skrivet i proram-meringsspråket HILP, ett högnivåspråk för mikrodatorn Intel 8080.

I kapitel 3 ges en beskrivning av programmet med de ändringar som gjorts. I kapitel 4 finns en manual och i kapitel 5 redovisas slutligen de experiment som gjorts. I appendix finns infört program-listor.

3 PROGRAMBESKRIVNING

Huvudprogram

I förarbetet till detta examensarbete hade antagits att programmet skulle ligga i ett minne om 2K ord. Emellertid kan endast 1K av RAM-minnet utnyttjas för programändamål beroende på konstruktionen av mikrodatorns operativsystem. För att få plats har därför delar av huvudprogrammet brutits loss, förvandlats till subrutiner och förlagts till PROM:ar (Subrutinerna GRAD och DRCTN). Då jag var tvingad att ändra subrutinerna för flyttal och lägga dessa i PROM, ansåg jag att dessa subrutiner borde räkna med en noggrannhet av 16 bitar i mantissan i stället för de 8 bitar som befintliga subrutiner utnyttjade. Gradienten beräknas ju med hjälp av differenser och signifikanta bitar går förlorade, speciellt om derivatan är liten. Dessutom fanns ett färdigprogrammerat PROM med dessa subrutiner inlagda. Som en följd av detta har deklarationerna av variablerna svällt ut då ju dessa subrutiner fordrar ytterligare en cell för mantissan. Samtidigt har givetvis alla subrutinanrop måst ändras. Beträffande programmets matematiska teori och funktion hänvisas till I. Sixtenssons utmärkta examensarbete.

Observera att i HILP kan man inte skriva en instruktion av typen CALL PROC(DE=XLAMB,B=XLAMBE,HL=XX[I],A=XXE[I]). Eftersom XXE är indicerad genererar STAGE2 ett assemblerprogram som vid exekvering förstör innehållet i A-registret. Genom att använda instruktionerna PUSH PSW och POP PSW kan man rädda A-registrets innehåll till stacken.

Subrutiner för flyttal

Dessa subrutiner finns utförligt beskrivna i D.W. Clake: Floating point arithmetic routines etc. Emellertid har en smärre ändring i förhållande till detta arbete gjorts i fråga om FADD. Denna ändring finns dokumenterad på de två sista sidorna i George Kiziroglus examensarbete. Se litteraturförteckningen.

De register som används av subrutinerna framgår av nedanstående schema.

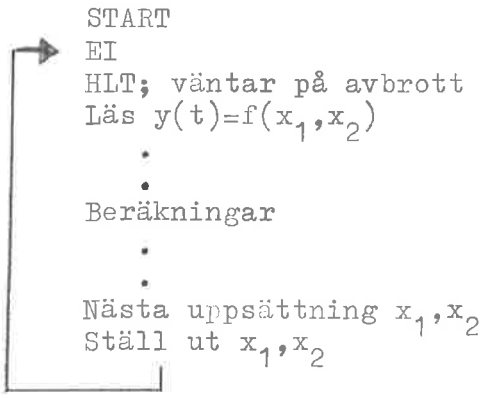
Faktor/Täljare/Term/Resultat	Mantissa: DE Exponent: B
Faktor/Nämnare/Term	Mantissa: HL Exponent: A

Talrepresentatioen är två-komplement i mantissan och 7 bitar binär offset i exponenten. Subrutinerna FLYT och FIX16 ger omvandling från ett fixtal i DE registret till ett flyttal enligt ovan och vice versa.

Kommunikation med processen

Den subrutin som används för kommunikation med den yttre processen är FVAL. När programmet behöver ett funktionsvärde ställs först argumenten ut på D/A-omvandlarna. Därefter passeras instruktionerna EI (Enable Interrupts) och HLT (HaLT), varvid programexekveringen avbryts. När en avbrottsignal kommer sker uthopp till cell 070 där instruktionen 311 (RETurn) finns. Återhopp till programcellen närmast efter HLT sker omedelbart och programmet fortsätter att exekvera. Schematiskt kan

ovanstående beskrivas:



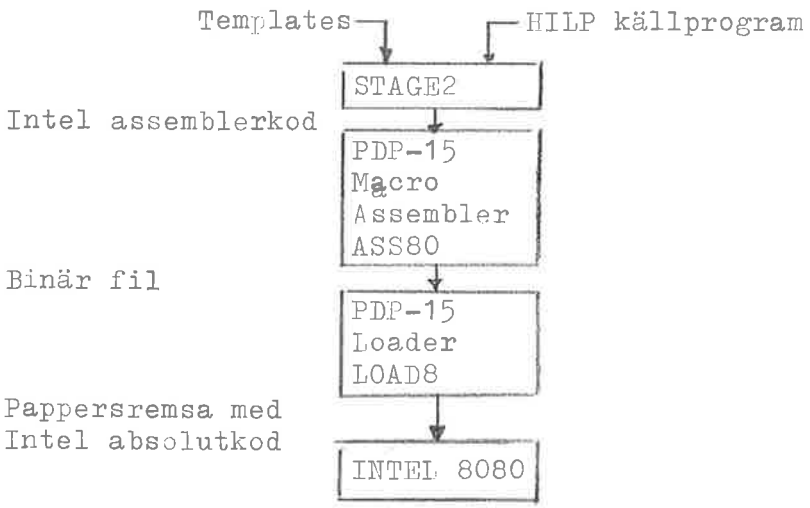
Periodtiden för avbrottsignalen måste väljas $T > T_r + T_i$, där T_r är den längsta räknetiden för programmet och T_i är processen insvängningstid. Experimentellt visade sig T=0.1 s vara lämplig.

Dataminnets organisation

Innehållet i varje cell i dataminnet framgår av tabell 1. Där finns angivet cellens nummer i oktal och decimal representation samt vilken variabel eller instruktion som cellen innehåller. Suffixet E i ett variabelnamn anger att den är en exponent till motsvarande variabel. Understrukna variabelnamn anger att dessa måste läggas in i minnet för hand med hjälp av Transintrons frontpanel. Anlednigen till att en del variabler inte ges värden av programmet är att de bör lätt kunna ändras genom att gå in på den adress som anges av tabell 1 och inte i själva programmet. Minnets tre första celler innehåller instruktionerna 303, 044,001 vilket betyder hopp till cell 044 bank 001, där huvudprogrammet börjar. Datorn börjar nämligen alltid att exekvera i cell 000. I cell 070 finns instruktionen 311=RETurn som ger omedelbart återhopp vid avbrott.

Kompilering, assemblering etc.

För att kompilera källprogrammet i HILP användes institutionens PDP-15 som värd-dator. Arbetgången framgår schematiskt av nedanstående figur.



LOAD8 ger direkt en pappersremsa innehållande ett relokerbart binärt objektprogram klar att laddas in i mikrodatorns minne genom en remsläsare. Subrutinerna har förljande startadresser:

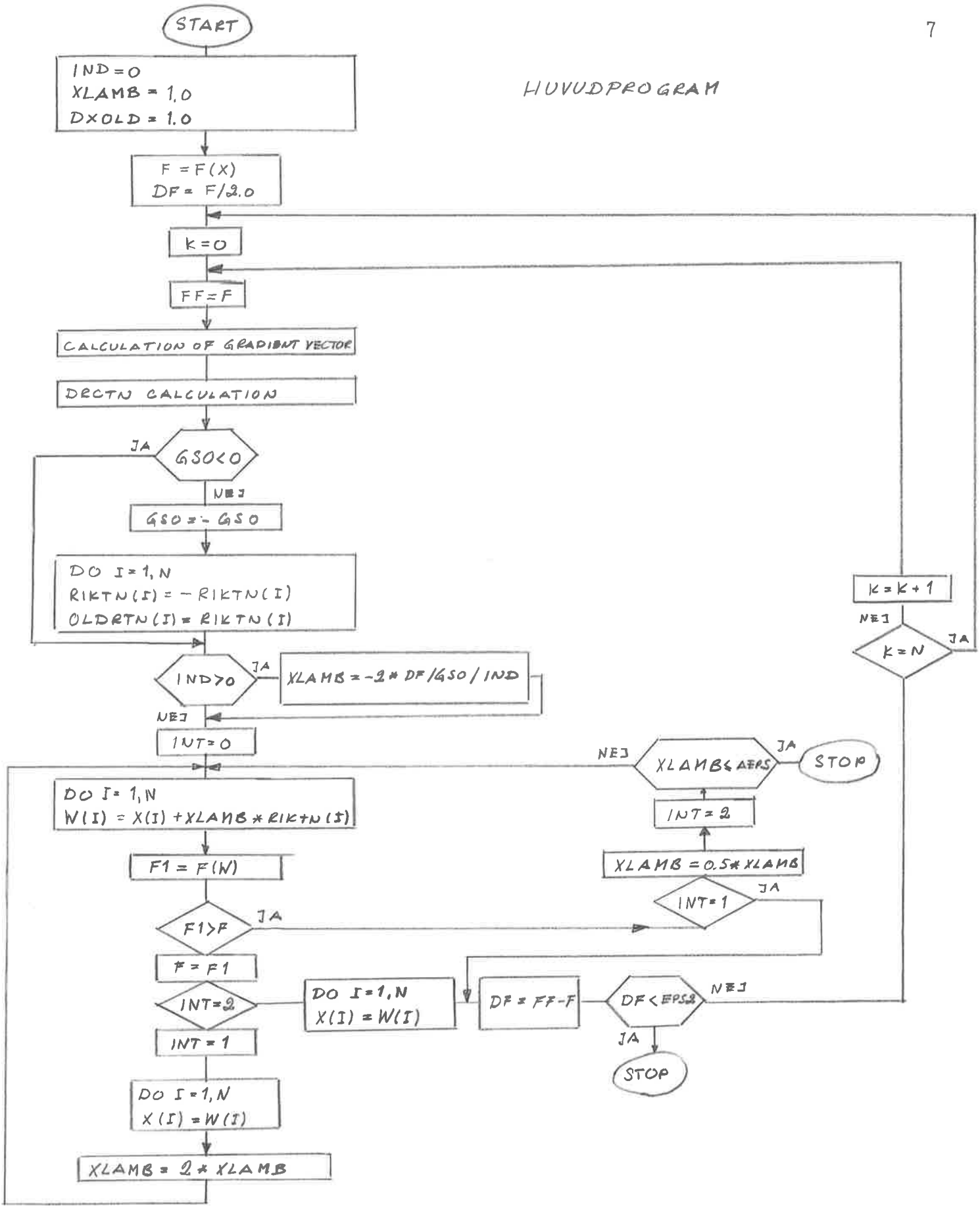
FMUL	13:317	FVAL	00:210
FDIV	14:036	GRAD	10:005
FADD	14:151	DRCTN	16:000
FLYT	13:110	FNORMO	14:231
FIX16	13:010		

Det är lämpligt att när man laddar programmet börja med FVAL i 000:210 och därefter huvudprogrammet FLR. Program och stack får då precis plats i minnet.

0	0	303=JMP	100	64	
1	1		101	65	XXE(0)
2	2		102	66	
3	3	K	103	67	XXE(2)
4	4	I	104	68	W(0)
5	5	IND	105	69	
6	6	INT	106	70	W(2)
7	7	ADR	107	71	
10	8		110	72	WE(0)
11	9		111	73	
12	10	F	112	74	WE(2)
13	11		113	75	G(0)
14	12	FE	114	76	
15	13	F1	115	77	G(2)
16	14		116	78	
17	15	F1E	117	79	GE(0)
20	16	FF	120	80	
21	17		121	81	GE(2)
22	18	FFE	122	82	RIKTN(0)
23	19	DF	123	83	
24	20		124	84	RIKTN(2)
25	21	DFE	125	85	
26	22	Z	126	86	RIKTNE(0)
27	23		127	87	
30	24	ZE	130	88	RIKTNE(2)
31	25	GS0	131	89	
32	26		132	90	RES
33	27	GSOE	133	91	
34	28	AEPS	134	92	RESE
35	29		135	93	
36	30	AEPSE	136	94	ABSRTN(0)
37	31	EPS1	137	95	
40	32		140	96	ABSRTN(2)
41	33	EPS1E	141	97	
42	34	EPS2	142	98	ABSRTNE(0)
43	35		143	99	
44	36	EPS2E	144	100	ABSRTNE(2)
45	37	HF	145	101	
46	38		146	102	OLDRTN(0)
47	39	HFE	147	103	
50	40	HH	150	104	OLDRTN(2)
51	41		151	105	
52	42	HHE	152	106	OLDRTNE(0)
53	43	ALPHA	153	107	
54	44		154	108	OLDRTNE(2)
55	45	AIPHAE	155	109	S(0)
56	46	DXNEW	156	110	
57	47		157	111	S(2)
60	48	DXNEWE	160	112	
61	49	DXOLD	161	113	SE(0)
62	50		162	114	
63	51	DXOLDE	163	115	SE(2)
64	52	XLAMB			
65	53				
66	54	XLAMBE			
67	55				
70	56	311=RET			
71	57				
72	58				
73	59				
74	60				
75	61	XX(0)			
76	62				
77	63	XX(2)			

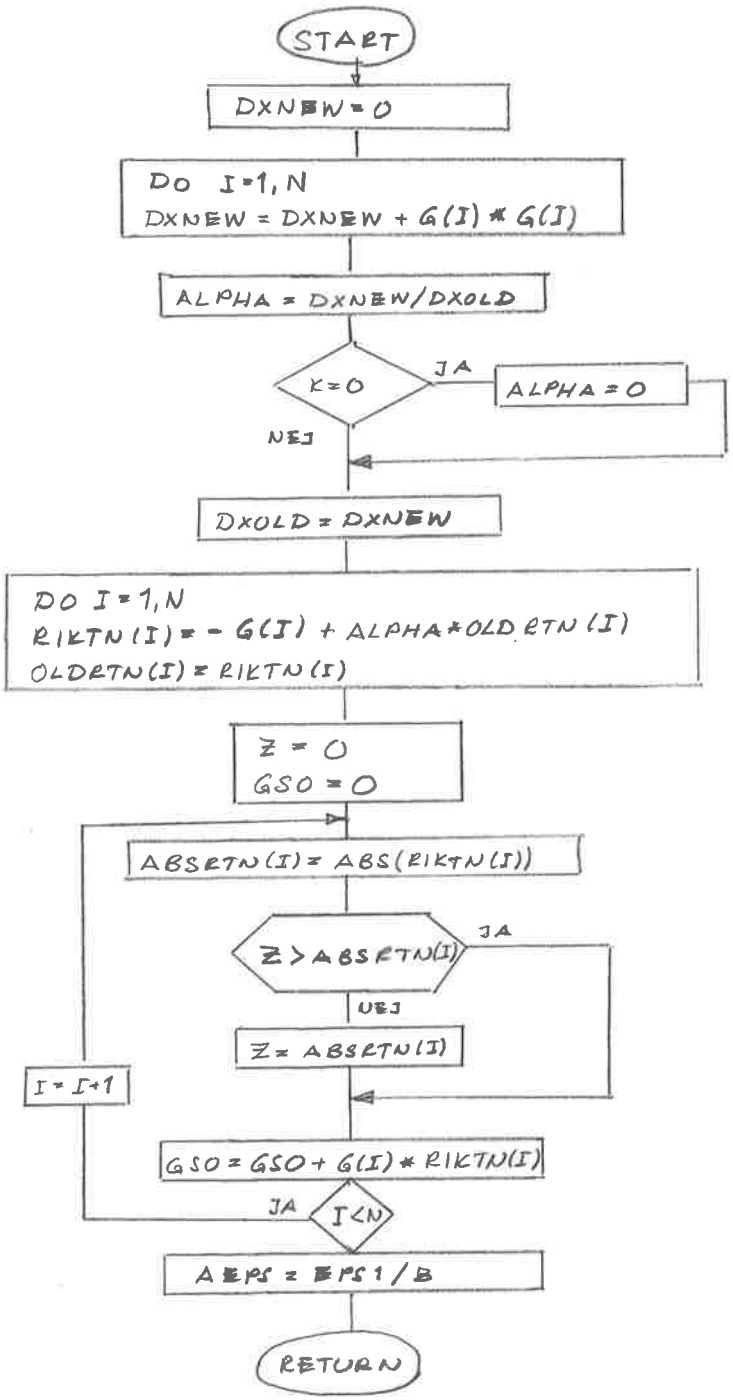
Tabell 1

HUVUDPROGRAM



FIGUR 1.

PROC DRCTN



FIGUR 2.

4 MANUAL

Uppstart av programmet

1. Kontrollera att interface-modulen är ansluten till minnet, samt att teletypen är ansluten.

2. Slå på spänningen.

3. Sätt EXT/INT-omkopplaren i läge EXT. Tryck på SINGLE STEP en gång.

4. Ställ in adress 005 i switchregistret.

5. Sätt teletypen i läge LINE.

6. Tryck på RESTART samt därefter RUN på Transintron. Teletypen skriver då ut en asterisk.

7. Skriv på teletypen LOADB:(stopadress):(startadress):. Exempel: LOADB:033:375:030:210:. Observera att bank 000 heter här 030 etc.

8. Applicera hålremsan och starta remsläsaren.

9. Programmet läses nu in i RAM-minnet. När hela programmet är inläst, skriver inläsningsprogrammet ut en asterisk på teletypen. Stäng av remsläsaren.

10. Tryck på STOP på Transintron samt en gång på SINGLE STEP.

11. Lägg in instruktioner enligt tabell 1 i RAM -minnet. Se nedan.

12. Sätt EXT/INT i läge INT. Tryck på RESET.

13. Sätt RUN/STOP/SINGLE STEP-omkopplaren i läge RUN. Nu exekveras programmet.

14. När man vill stanna programmet slår man över omkopplaren i läge STOP. För att komma åt att läsa och skriva i RAM-minnet sätt EXT/INT i läge EXT samt tryck på SINGLE STEP. Observera att om EXT/INT-omkopplaren varit i läge INT, måste man alltid trycka på SINGLE STEP en gång.

Skrivning i RAM-minnet

1. Kontrollera att EXT/INT-omkopplaren är i läge EXT.

2. Ställ in adressen på switchregistret.

3. Tryck på LOAD ADRESS.

4. Ställ in önskat minnesinnehåll i switchregistret.

5. Tryck på LOAD MEMORY.

6. Om man vill gå till nästföljande minnescell tryck på STEP ADRESS och gå till 4 annars gå till 2.

5 EXPERIMENT

Rosenbrock's curved valley

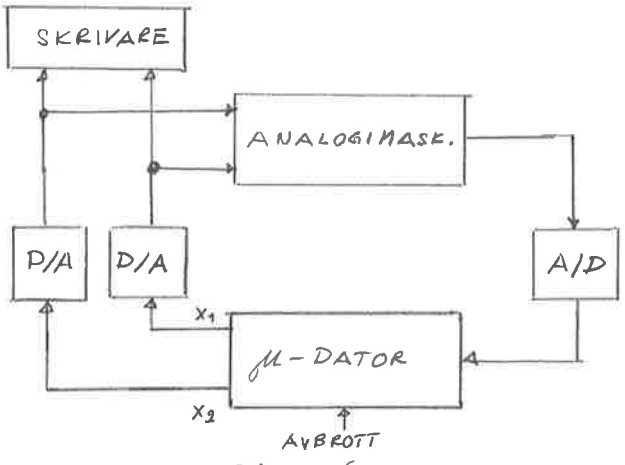
Det första försök som gjordes var på en funktion inlagd som en subrutin i programmet. x_1 och x_2-värdena ställdes ut på D/A-omvandlare. På så sätt kunde man åskådliggöra hur optimeringen förflöt på en oscilloskopskärm eller tvåkanalsskrivare. Den inlagda funktionen är "Rosenbrocks curved valley", ett vanligt testproblem vid optimering. Det analytiska uttrycket för funktionen är $f(x_1,x_2)=100(x_2-x_1^2)^2+(1-x_1)^2$. Funktionen har formen av en banan med en minimipunkt i (1,1) med funktionsvärdet 0.0. Med startvärdet (-1.2,1.0) rör man sig i"dalen" runt "hörnet" i origo mot minimipunkten (1,1). Figurer med nivåkurvor över funtionen finns i Sixtenssons examensarbete.

Resultatet av några försök finns i figur 3 - 5. De långa topparna i figurerna beror på att programmet gör en omstart vid vissa förhållanden (se flödesschema). Programmet kan då hamna i en punkt långt bort vilket får till förljd att D/A-omvandlaren blir mättad under ett kort ögonblick.

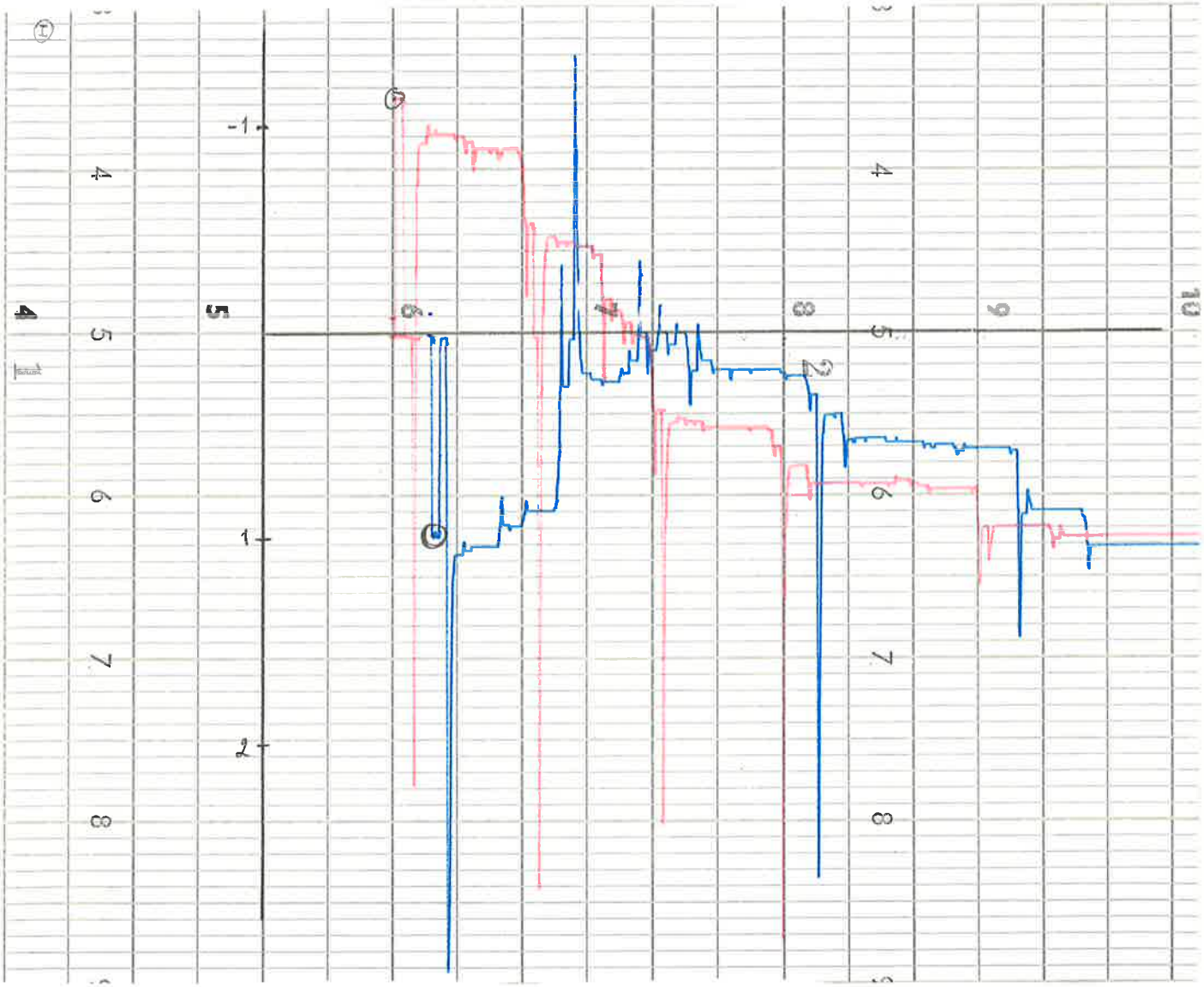
Valet av parametrarna HF, EPS1, EPS2 är ett något kinkigt problem, eftersom de är starkt kopplade till varandra. Väljer man ett för litet värde på steglängden vid derivataberäkningen (HF) kan man få numeriska problem på grund av kavnäseringnen. Ett för stort värde på HF ger dålig noggrannhet på derivatavärdet. För stora värden på EPS1 respektive EPS2 gör att programmet betraktar minimeringen som avslutad för tidigt. Vid för små värden stannar programmaet kanske aldrig. Vid en inställning gör man klokt i att välja HF till ett lämpligt värde mellan ytterligheterna. Därefter väljer man EPS1 resp EPS2 till programmet fungerar bra. Fungerar inte programmet tillfredställande väljer man ett nytt HF o.s.v.

Parabol uppkopplad på analogimaskin

Några mer realistiska försök med en yttre funktion gjordes även. Den enkla funktionen $f(x_1,x_2)=\alpha x_1^2+\beta x_2^2$ kopplades upp på analogimaskin enligt figur 7. I praktiken tar man givetvis bort inverterarna 8 och 10 och byter samtidigt +REF till -REF i summatorn 9. I figuren är de dock med för tydlighetens skull. Schematiskt beskrivs uppställnigen av figur 6.

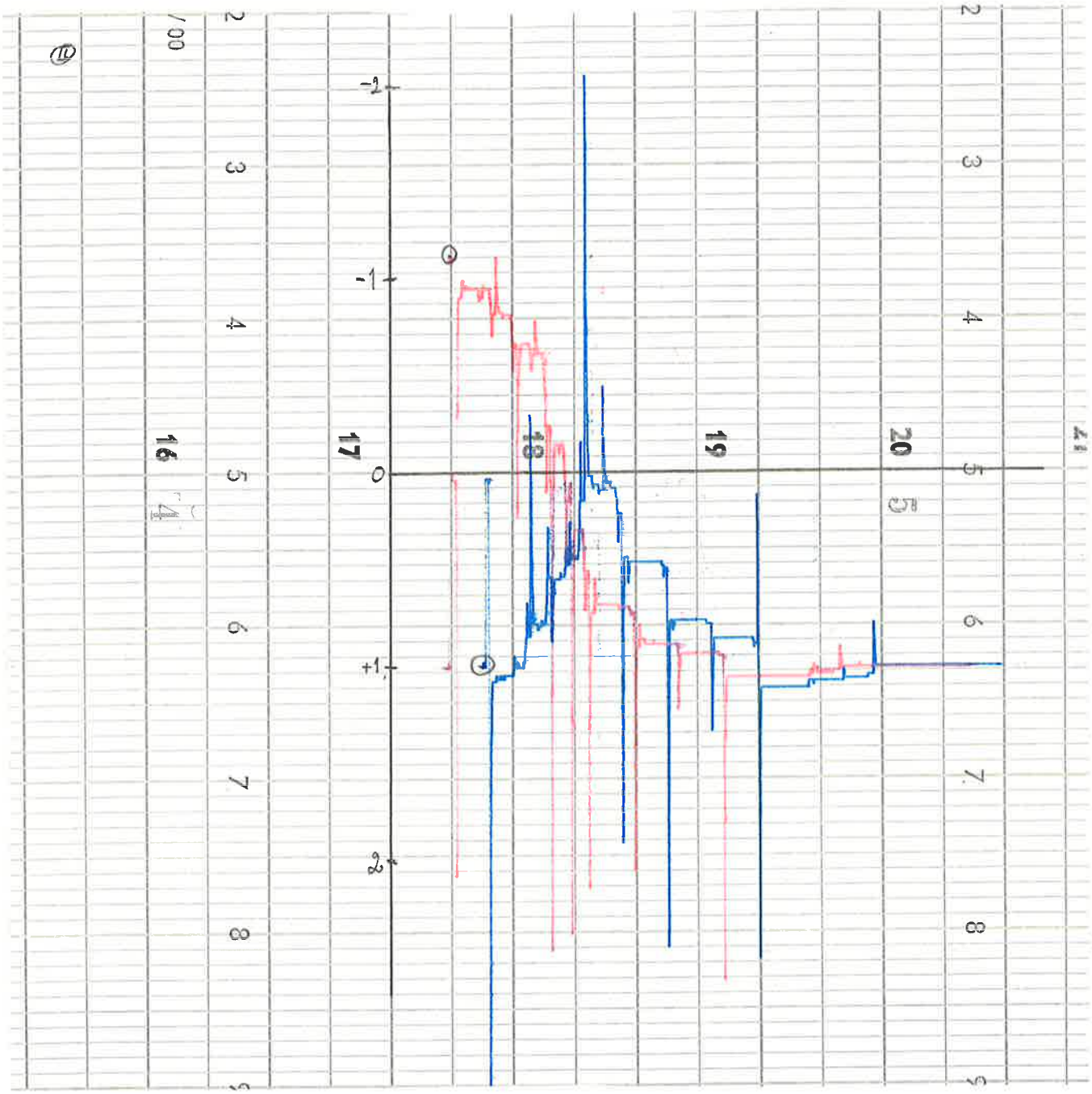


Figur 6.



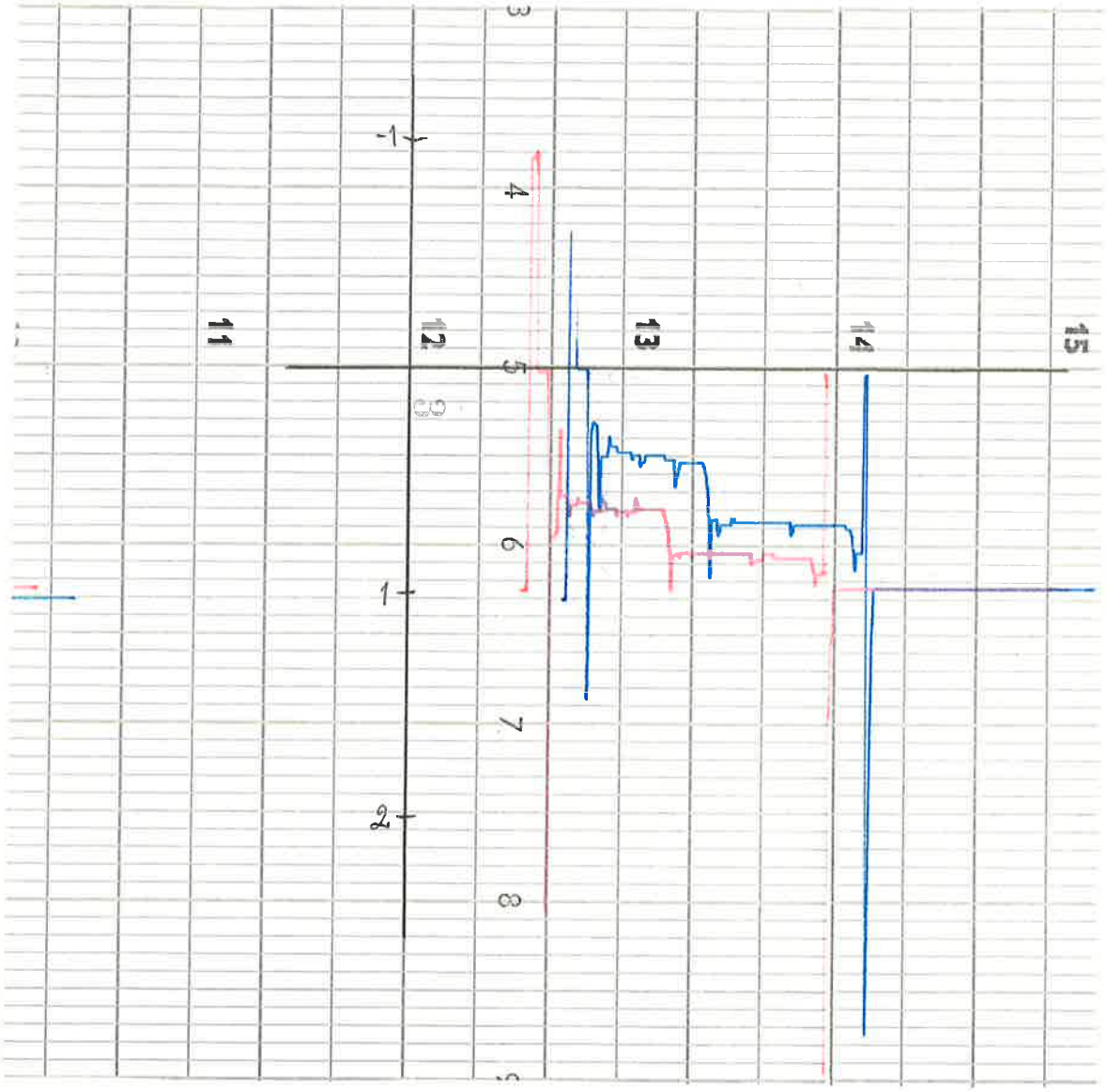
Figur 3

Startpunkt:	(-1.2;1)
HF:	$7.81\ 10^{-5}$
EPS1:	$3.13\ 10^{-5}$
EPS2:	$9.78\ 10^{-7}$
Pappershastighet:	160 mm/min
Pulsfrekvens:	0.1 sek



Figur 4

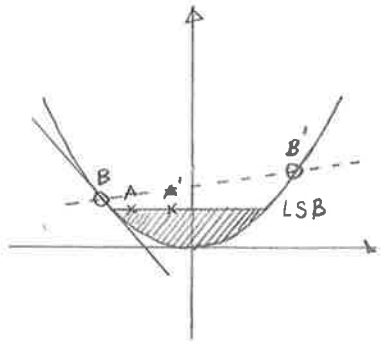
Startpunkt:	(-1.2;1)
HF:	$3.12\ 10^{-4}$
EPS 1:	$6.26\ 10^{-5}$
EPS2:	$9.78\ 10^{-7}$
Pappershastighet:	80 mm/min
Pulsfekvens:	0.1 sek



Figur 5

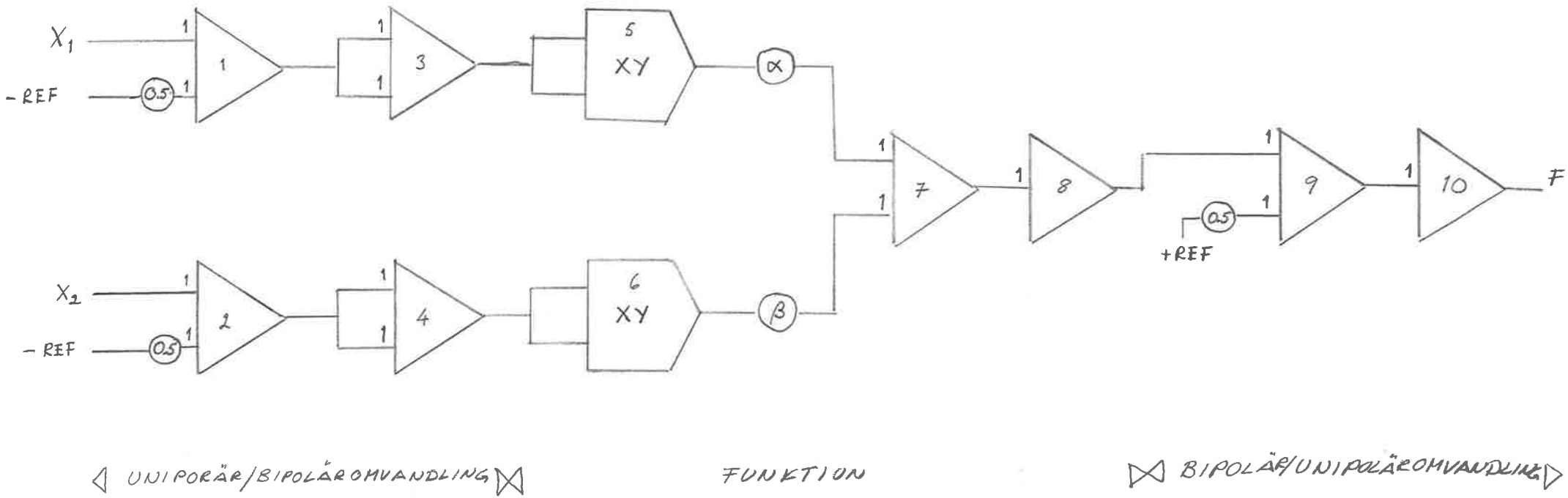
Startpunkt:	(-1;-1)
HF:	$7.81\ 10^{-5}$
EPS1:	$3.13\ 10^{-5}$
EPS2:	$9.78\ 10^{-7}$
Pappershastighet:	160 mm/min
Pulsfrekvens:	0.1 sek

Problemet visade sig bli numeriskt illa konditionerat. Man måste i programmet lägga in begränsningar så att man inte hamnar utanför omvandlarnas utsyrningsområde. Detta får till följd att man får en"bottensats" i parabolen om 16 bitar där utsignalen blir lika med den minst signifikanta biten (=1/256). Detta medför i sin tur att minimipunkten kan bestämmas med endast 4 bitars noggrannhet. Det blir även svårt att finna ett bra värde på HF. Gör man HF för litet får man derivatan lika med noll i "bottensatsen" (A - A' i figur) och det kan programmet inte hantera på något bra sätt. Ett för stort HF kan ge helt fel tecken på derivatan (B - B' i figur)



Ovanstående problem skulle kunna elimineras med 16 bitars D/A- och A/D-omvandlare. Programtekniskt är det dessutom lämpligt då vi ju har subrutiner för flyttal med 16 bitars mantissa. Några sådana omvandlare fanns dock ej att tillgå.

De odokumenterade försök som gjordes gav samma erfarenhet beträffande parameterval som förut. Några ytterligare försök gjordes inte.


X1
- REF
0.5
1
3
5
XY
α
X2
- REF
0.5
2
4
6
XY
β
7
8
9
10
F
0.5
+REF
UNIPOLÄR/BIPOLÄROMVANDLING
FUNKTION
BIPOLÄR/UNIPOLÄROMVANDLING

FIGUR 7.

6 LITTERATURFÖRTECKNING

Aspernäs,J.E. HILP 80 - A high level programming language for Intel 8080. Report 7626(C). Institutionen för Reglerteknik, LTH. Lund 1976.

Clarke, D.W. Floating point arithmetic routines and macros for an Intel 8080 microprocessor. Department of Engineering Science. University of Oxford.

Kizi roglu,G. Implementation of a self-tuning regulator on a microcomputer. RE-182. Institutionen för Reglerteknik, Lund 1976.

Ramsin,H. Ickelinjär optimering. Matematiska institutionen, KTH. Stockholm 1975.

Sixtensson,I. Impementering av extremalsökande algoritm på mikrodator. Institutionen för Reglerteknik, LTH. Lund 1976.

```
                        ;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
;       REVISED: G.WIKTORSSON 1976-12-21
;       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
;
;
        CONST ¤BK=0,¤N=2
        BANK ¤BK
        WORDS K↑3,I,IND,INT
        WORDS F↑10,FE↑12,F1↑13,F1E↑15,FF↑16,FFE↑18
        WORDS DF↑19,DFE↑21,Z↑22,ZE↑24,GS0↑25,GSOE↑27
        WORDS AEPS↑28,AEPSE↑30,EPS1↑31,EPS1E↑33,EPS2↑34,EPS2E↑36
        WORDS HF↑37,HFE↑39,HH↑40,HHE↑42,ALPHA↑43,ALPHAE↑45
        WORDS DXNEW↑46,DXNEWE↑48,DXOLD↑49,DXOLDE↑51,XLAMB↑52,XLAMBE↑54
        WORDS ABSRTN[2]↑94,ABSRTE[2]↑98,OLDRTN[2]↑102,OLDRTE[2]↑106
        WORDS XX[¤N]↑61,XXE[¤N]↑65,W[¤N]↑68,WE[¤N]↑72
        WORDS G[¤N]↑75,GE[¤N]↑79,RIKTN[¤N]↑82,RIKTNE[¤N]↑86
        GLOBAL FADD,FMUL,FDIV,FVAL,DRCTN,GRAD
        GLOBAL FLYT,FIX16
        SET STACK 3:255

        IND=0
        DX XLAMB=040000;        =1.0
        XLAMBE=0101
        DX DXOLD=040000;        =1.0
        DXOLDE=0101

        CALL FVAL(DE=XX[0],B=XXE[0],A=XXE[2],HL=XX[2])
        DX F=DE;        F=F(X)
        FE=B
        CALL FDIV(A=0102,HL=040000)
        DX DF=DE;       DF=F/2.0
        DFE=B
;
;       ITERATION PROCEDURE STARTS
;
L10:    K=0
L15:    DX FF=F
        FFE=FE
```

```
;
;       NUMERICAL CALCULATION OF THE GRADIENT VECTOR
;
        CALL GRAD
;
;       CALCULATE THE DESCENT DIRECTION
;
        CALL DRCTN

        DREG DE=GSO
        GOTO L52 IF 0>D;GOTO L52 IF GSO<0
        DREG DE=-DE
        DX GSO=DE
        I=4
        REPEAT
          I=I-2
          RIKTN[I]=-RIKTN[I]
          OLDRTN[I]=RIKTN[I]
          OLDRTE[I]=RIKTNE[I]
        UNTIL I=0
;
;       MINIMIZE ALONG THE DESCENT DIRECTION
;
  L52:  IF IND>0 THEN
          CALL FLYT(DE=IND)
          CALL FDIV(HL=DE,A=B,DE=0140000,B=0102)
          CALL FMUL(HL=DF,A=DFE)
          CALL FDIV(HL=GSO,A=GSOE)
          DX XLAMB=DE;  XLAMB=-2.0*DF/GSO/IND
          XLAMBE=B
        END
        INT=0
  L55:  I=¤N+2
        REPEAT
          I=I-2
          DREG DE=XLAMB
          REG B=XLAMBE
          REG A=RIKTNE[I]
          PUSH PSW;    SAVE A REGISTER
          DREG HL=RIKTN[I]
          POP PSW
          CALL FMUL
          REG A=XXE[I]
          PUSH PSW;   SAVE A REGISTER
          DREG HL=XX[I]
          POP PSW
          CALL FADD
          DX W[I]=DE;   W(I)=X(I)+XLAMB*RIKTN(I)
          WE[I]=B
        UNTIL I=0
  L56:  I=¤N+2
        REPEAT
          I=I-2
          CALL FADD(DE=W[I],B=WE[I],HL=040000,A=0101)
          GOTO L60 IF 0>D
          CALL FADD(DE=W[I],B=WE[I],HL=0140000,A=0101)
          GOTO L60 IF D>0
        UNTIL I=0;   JUMP IF W>1
        GOTO L65
  L60:  CALL FMUL(DE=XLAMB,B=XLAMBE,HL=040000,A=0100)
        DX XLAMB=DE
        XLAMBE=B
        GOTO L55
```

```
L65:
      CALL FVAL(DE=W[0],B=WE[0],A=WE[2],HL=W[2])
      DX F1=DE;        F1=F(W)
      F1E=B
      DREG DE=-DE
      CALL FADD(HL=F,A=FE)
      GOTO L70 IF 0>D;      GOTO L70 IF F1>F
      DX F=F1
      FE=F1E
      GOTO L87 IF INT=2
      INT=1
      I=□N+2
      REPEAT
        I=I-2
        DX XX[I]=W[I]
        XXE[I]=WE[I]
      UNTIL I=0
      CALL FADD(DE=XLAMB,B=XLAMBE,A=B,HL=DE)
      DX XLAMB=DE;     XLAMB=2.0*XLAMB
      XLAMBE=B
      GOTO L55
L70:  GOTO L90 IF INT=1
      CALL FMUL(DE=040000,B=0100,A=XLAMBE,HL=XLAMB)
      DX XLAMB=DE;     XLAMB=0.5*XLAMB
      XLAMBE=B
      INT=2
      DREG DE=AEPS
      DREG DE=-DE
      CALL FADD(B=AEPSE,A=XLAMBE,HL=XLAMB)
      GOTO L100 IF D<=0;      GOTO L100 IF XLAMB<AEPS
      GOTO L55
L87:  I=□N+2
      REPEAT
        I=I-2
        DX XX[I]=W[I]
        XXE[I]=WE[I]
      UNTIL I=0
L90:  DREG DE=F
      DREG DE=-DE
      CALL FADD(B=FE,A=FFE,HL=FF)
      DX DF=DE;        DF=FF-F
      DFE=B
      DREG DE=-DE
      CALL FADD(A=EPS2E,HL=EPS2)
      GOTO L100 IF D>0;     GOTO L100 IF DF<EPS2
;
;     TEST FOR RECYCLING
;
      GOTO L10 IF K=□N
      K=K+1
      GOTO L15
L100: .HLT
      FINISH
```

```
PROCEDURE GRAD
CONST □N=2
BANK 0
WORDS I↑4,F1↑13,F1E↑15,HF↑37,HFE↑39,HH↑40,HHE↑42
WORDS XX[2]↑61,XXE[2]↑65,G[2]↑75,GE[2]↑79
GLOBAL GRAD,FVAL,FADD,FDIV
I=□N+2
REPEAT
  I=I-2
  CALL FADD(DE=XX[I],B=XXE[I],A=HFE,HL=HF)
  DX XX[I]=DE;  X(I)=X(I)+HF
  XXE[I]=B
  CALL FVAL(DE=XX[0],B=XXE[0],A=XXE[2],HL=XX[2])
  DX F1=DE;     F1=F(X)
  F1E=B
  CALL FADD(DE=HF,B=HFE,A=B,HL=DE)
  DX HH=DE;     HH=HF+HF
  HHE=B
  DX DE=-DE
  REG A=XXE[I]
  PUSH PSW
  DREG HL=XX[I]
  POP PSW
  CALL FADD
  DX XX[I]=DE;  X(I)=X(I)-HH
  XXE[I]=B
  CALL FVAL(DE=XX[0],B=XXE[0],A=XXE[2],HL=XX[2])
  DX DE=-DE
  CALL FADD(A=F1E,HL=F1)
  CALL FDIV(A=HHE,HL=HH)
  DX G[I]=DE;   G(I)=(F1-F(X))/HH
  GE[I]=B
  CALL FADD(DE=XX[I],B=XXE[I],A=HFE,HL=HF)
  DX XX[I]=DE;  X(I)=X(I)+HF
  XXE[I]=B
UNTIL I=0
ENDPROC
FINISH
```

```
PROCEDURE DRCTN
CONST □N=2
BANK 0
GLOBAL DRCTN,FMUL,FADD,FDIV
WORDS K↑3,I↑4,Z↑22,ZE↑24,GSO↑25,GSOE↑27,AEPS↑28,AEPSE↑30
WORDS ALPHA↑43,ALPHAE↑45,DXNEW↑46,DXNEWE↑48,DXOLD↑49,DXOLDE↑51
WORDS EPS1↑31,EPS1E↑33,G[2]↑75,GE[2]↑79
WORDS RIKTN[2]↑82,RIKTNE[2]↑86,ABSRTN[2]↑94,ABSRTE[2]↑98
WORDS OLDRTN[2]↑102,OLDRTE[2]↑106
DX DXNEW=0
DXNEWE=0
I=□N+2
REPEAT
  I=I-2
  CALL FMUL(DE=G[I],B=GE[I],A=B,HL=DE)
  CALL FADD(DE=DE,B=B,A=DXNEWE,HL=DXNEW)
  DX DXNEW=DE;  DXNEW=DXNEW+G(I)*G(I)
  DXNEWE=B
UNTIL I=0
CALL FDIV(DE=DE,B=B,A=DXOLDE,HL=DXOLD)
DX ALPHA=DE;    ALPHA=DXNEW/DXOLD
ALPHAE=B
IF K=0 THEN
  DX ALPHA=0
  ALPHAE=0
END
DX DXOLD=DXNEW
DXOLDE=DXNEWE
I=□N+2
REPEAT
  I=I-2
  DREG DE=ALPHA
  REG B=ALPHAE
  REG A=OLDRTE[I]
  PUSH PSW;  SAVE A REGISTER
  DREG HL=OLDRTN[I]
  POP PSW
  CALL FMUL
  REG A=GE[I]
  PUSH PSW
  DREG HL=G[I]
  DREG HL=-HL
  POP PSW
  CALL FADD
  DX RIKTN[I]=DE;       RIKTN(I)=-G(I)+ALPHA*OLDRTN(I)
  RIKTNE[I]=B
  DX OLDRTN[I]=DE;      OLDRTN(I)=RIKTN(I)
  OLDRTE[I]=B
UNTIL I=0


DX Z=0
ZE=0
DX GSO=0
GSOE=0
I=□N+2
REPEAT
  I=I-2
  DREG DE=RIKTN[I]
  IF 0>D THEN
    DREG DE=-DE
  END
  DX ABSRTN[I]=DE
```

```
    ABSRTE[I]=RIKTNE[I]
    DREG DE=Z
    DREG DE=-DE
    REG B=ZE
    REG A=ABSRTE[I]
    PUSH PSW
    DREG HL=ABSRTN[I]
    POP PSW
    CALL FADD
    IF 0<=D THEN
      DX Z=ABSRTN[I];      Z WILL BE MAX(ABS(RIKTN(I)))
      ZE=ABSRTE[I]
    END
    DREG DE=G[I]
    REG B=GE[I]
    REG A=RIKTNE[I]
    PUSH PSW
    DREG HL=RIKTN[I]
    POP PSW
    CALL FMUL
    CALL FADD(A=GSOE,HL=GSO)
    DX GSO=DE;     GSO=GSO+G(I)*RIKTN(I)
    GSOE=B
  UNTIL I=0
  CALL FDIV(DE=EPS1,B=EPS1E,A=ZE,HL=Z)
  DX AEPS=DE;      AEPS=EPS1/Z
  AEPSE=B
  ENDPROC
  FINISH
```

```
        PROC FVAL
;               THE PROCEDURE GENERATES THE FUNCTION
;               ROSENBROCKS CURVED VALLEY.
;               CALL IS:CALL FVAL(DE=XX[0],B=XXE[0],HL=XX[2],A=XXE[2])
;               RESULT IN:DE B
;
        BANK 0
        WORDS RES↑90,RESE↑92
        WORDS S[2]↑109,SE[2]↑113
        GLOBAL FADD,FMUL,FVAL,FIX16
        DX S[2]=HL
        SE[2]=A
        DX S[0]=DE
        SE[0]=B
        CALL FMUL(HL=040000,A=0116);SCALE FACTOR
        CALL FIX16
        .MOV A,D
        .XRI 200;   CONVERT TO BINARY OFFSET
        .OUT 374
        CALL FMUL(DE=S[2],B=SE[2],HL=040000,A=0116)
        CALL FIX16
        .MOV A,D
        .XRI 200
        .OUT 375
        .EI
        .HLT

        CALL FMUL(DE=S[0],B=SE[0],A=B,HL=DE)
        DX DE=-DE
        CALL FADD(A=SE[2],HL=S[2])
        CALL FMUL(A=B,HL=DE)
        CALL FMUL(A=0107,HL=062000)
        DX RES=DE
        RESE=B
        DREG DE=S[0]
        DX DE=-DE
        CALL FADD(B=SE[0],A=0101,HL=040000)
        CALL FMUL(A=B,HL=DE)
        CALL FADD(A=RESE,HL=RES)
        ENDPROC
FINISH
```

```
        PROC FVAL
;       THE PROCEDURE COMMUNICATES WITH AN EXTERNAL
;       PROCESS.
        RANK 0
        WORDS S↑109,SE↑113
        GLOBAL FIX16,FLYT,FDIV,FMUL,FVAL
        DX S=HL
        SE=A
        CALL FMUL(HL=040000,A=0115);SCALE FACTOR
        CALL FIX16
        .MOV A,D
        .XRI 200;CONVERT TO BINARY OFFSET
        .OUT 374
        CALL FMUL(DE=S,B=SE,HL=040000,A=0115)
        CALL FIX16
        .MOV A,D
        .XRI 200
        .OUT 375
        .EI
        .HLT
        .OUT 376
        .XRA A
WAIT:    .IN 377
        .ANI 1
        .CPI 0
        .JNZ WAIT
        .IN 376
        .XRI 200;   CONVERT TO TWO-COMPLEMENT
        .MOV D,A
        .MVI E,0
        CALL FLYT
        CALL FDIV(HL=040000,A=0115)
        ENDPROC
FINISH
```