Een numerieke procedure voor het integreren van een niet-lineair viscoelastisch integraalmodel van Fung

Een numerieke procedure voor het integreren van een niet-lineair viscoëlastisch integraalmodel van Fung

Stageverslag Ben Hendriks

WFW 89.075

Begeleiders : Cees Oomens
Max Hendriks

Inhoudsopgave

Samenvatting

De constitutieve vergelijkingen voor visco–elastische materialen zijn in het algemeen niet analytisch oplosbaar. In dit verslag wordt een numerieke oplossing gepresenteerd voor het model van Fung met een willekeurig verloop van de rek als funktie van de tijd. Het model is gekoppeld aan een bestaand parameter–schat–programma. De invoer van het programma bestaat uit een rekverloop en een set materiaalparameters. Het programma geeft als uitvoer de spanning volgens het model van Fung. Met behulp van het parameter–schat–programma is het dan mogelijk het model te konfronteren met experimentele resultaten uit een trekproef.

HOOFDSTUK 1 INLEIDING.

In het kader van het projekt 'mechanische karakterisering vezelversterkte kunststoffen' wordt een nieuwe methode ontwikkeld om de mechanische eigenschappen van kunststoffen en biologische materialen te achterhalen. Een belangrijk onderdeel hiervan is het bepalen van de onbekende materiaalparameters in een gekozen materiaalmodel. Hiertoe worden experimentele resultaten gekonfronteerd met het model. Een essentieel onderdeel van de methode is, dat dit op een iteratieve wijze moet gebeuren. Hiermee wordt bedoeld dat een groot aantal modelberekeningen gemaakt wordt, met steeds andere waarden voor de materiaalparameters.

Bovengenoemd onderzoeksprojekt loopt volgens twee lijnen. De eerste lijn is gericht op anisotroop en inhomogeen materiaalgedrag. Dit gebeurt door middel van proeven en simulaties aan elastische materialen. Tijdseffekten spelen hierin geen rol. De tweede lijn, waaraan deze stage een bijdrage levert, is vooral gericht op het tijdsafhankelijk gedrag van materialen en beperkt zich voorlopig tot experimenten die met 1–dimensionale modellen te beschrijven zijn. (Lankveld[1989]).

Een bekend 1–dimensionaal visco–elastisch materiaalmodel is het "model van Fung". Dit model heeft zijn nut bewezen bij de beschrijving van biologische materialen (zie bijvoorbeeld Fung[1972] en Huyghe[1986]).

De opdracht van de stage luidt:

- Programmeer het visco–elastische model van Fung in Turbo Pascal.
- Let hierbij op rekensnelheid en nauwkeurigheid.
- Implementeer dit programma in een bestaand parameter–schat–programma.

De indeling van dit verslag is als volgt. In hoofdstuk 2 wordt het model van Fung beschreven. In hoofdstuk 3 volgt een korte beschrijving van de numerieke aspekten. In

hoofdstuk 4 staat een globale beschrijving van het programma. In hoofdstuk 5 staan enkele testberekeningen. In hoofdstuk 6 worden een aantal konklusies gegeven en wordt aangegeven wat de voortgang zal zijn.

HOOFDSTUK 2 HET VISCO–ELASTISCHE MODEL VAN FUNG

Het model van Fung kan gezien worden als een speciaal geval van de gemodificeerde superpositie methode (Biologische materialen ; collegediktaat TUE nr. 4589). Volgens deze methode geldt voor een willekeurige rekgeschiedenis $\epsilon(t)$, dat de spanning $\sigma(t)$ gegeven wordt door: :

$$\sigma(t) = \int_{\tau=0}^{\tau=t} \frac{\delta f}{\delta \epsilon} \frac{\delta \epsilon}{\delta \tau} \, d\tau \qquad (2.1)$$

met $f = f(\epsilon(\tau), t-\tau)$; relaxatiefunktie

In het model van Fung wordt aangenomen dat de relaxatiefunktie (stapresponsiefunktie) gesplitst kan worden in een tijdsafhankelijk en een rekafhankelijk deel:

$$f(\epsilon,t) = G(t).\sigma^e(\epsilon(t)) \quad , \quad t>0 \qquad (2.2)$$

G is de gereduceerde relaxatiefunktie ; een monotoon–niet–stijgende en kontinue funktie. σ^e is de elastische respons; een willekeurige funktie van de rek met $\sigma^e(0)=0$. Hiermee vinden we de door Fung geponeerde constitutieve vergelijking:

$$\sigma(t) = \int_{\tau=0}^{\tau=t} G(t-\tau) \frac{\delta \sigma^e}{\delta \epsilon} \frac{\delta \epsilon}{\delta \tau} \, d\tau \qquad (2.3)$$

Fung neemt als relaxatiefunktie:

$$G(t) = \frac{1+K\{ \; E_1(t/\tau_2) - E_1(t/\tau_1) \; \}}{1 + K \cdot \ln(\tau_2/\tau_1)} \text{ als } 0<\tau_1<\tau<\tau_2 \qquad (2.4)$$

$$\text{met } E_1(x) = \int_{y=x}^{\infty} \frac{e^{-y}}{y} \, dy \quad ; \quad \text{de exponentiële integraalfunktie.} \qquad (2.5)$$

Uit een gevoeligheidsanalyse (Sauren en Rousseau [1983]) bleken de materiaalparameters de volgende invloed te hebben:

K is voornamelijk bepalend voor de waarde van $G(\infty)$,

en is mede bepalend voor de initiële relaxatiesnelhied.

τ_1 bepaalt in belangrijke mate de initiële relaxatiesnelheid.

τ_2 bepaalt in mindere mate de initiële relaxatie snelheid.

Voor de elastische respons nemen wij (Huyghe [1986]) :

$$\sigma^e = A \cdot (e^{B\epsilon} - 1) \;\; ; \;\; \frac{d\,\sigma^e}{d\,\epsilon} = A \cdot B \cdot e^{B\epsilon} \tag{2.6}$$

HOOFDSTUK 3 NUMERIEKE ASPEKTEN

Bij het programmeren is gebruik gemaakt van standaardprocedures uit Numerical Recipes [1986]. Het numeriek oplossen van de constitutieve vergelijking van Fung is op te splitsen in een aantal deelproblemen; de buitenste integraal (2.3), de exponentiële integraalfunktie (2.5) en het verwerken van de invoerfile met rekken op diskrete tijdstippen.

Het oplossen van de buitenste integraal gebeurt met de Romberg integratie: De procedure QROMB samen met de vereiste procedures TRAPZD en POLINT. Met de Romberg integratie wordt het te integreren interval in tweeën gedeeld en wordt in TRAPZD m.b.v. de uitgebreide trapeziumregel (Numerical Recipes) en samen met POLINT de integraal uitgerekend. Deze uitkomst gaat samen met een schatting van de fout, is deze fout groter dan een zelf in te voeren nauwkeurigheid (eps1) dan worden de intervallen opnieuw in tweeën gedeeld en wordt het proces herhaald.

Voor een numerieke oplossing van de exponentiële integraalfunktie moet deze eerst herschreven worden, omdat deze een oneindig integratie–interval heeft. We passen de volgende overgang van variabelen toe : $y = -\ln(z)$ of $z = e^{-y}$. Hiermee geldt:

$$E_1(x) = \int_{y=x}^{\infty} \frac{e^{-y}}{y}\,dy = \int_{z=0}^{e^{-x}} \frac{e^{\ln(z)}}{-z\cdot\ln(z)}\,dt = \int_{z=0}^{e^{-x}} \frac{-1}{\ln(z)}\,dz \qquad (3.1)$$

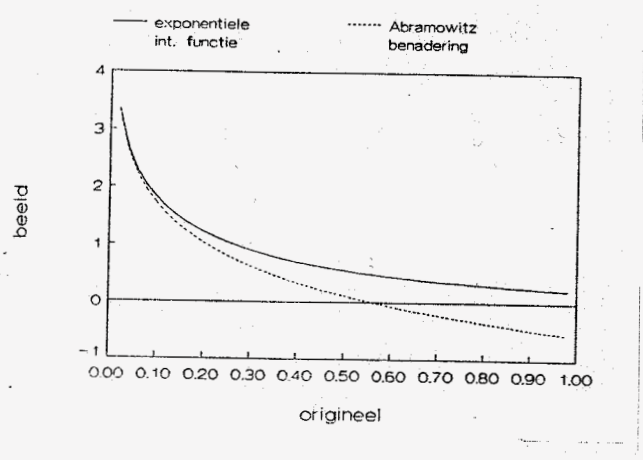
De integratie grenzen zijn nu eindig. Deze integraal kan echter niet met dezelfde procedures QROMB, TRAPZD en POLINT opgelost worden, omdat de integratiegrens t=o niet ingevuld mag worden in de integrand. In plaats van QROMB en TRAPZD wordt nu QROMO en MIDPNT gebruikt. Dit zijn aangepaste procedures voor oneigenlijke integralen. Belangrijke verschillen zijn dat de integratiegrenzen zelf niet

meer in de integrand worden ingevuld en dat bij iedere verfijning het aantal intervallen niet wordt verdubbelt maar verdrievoudigd.

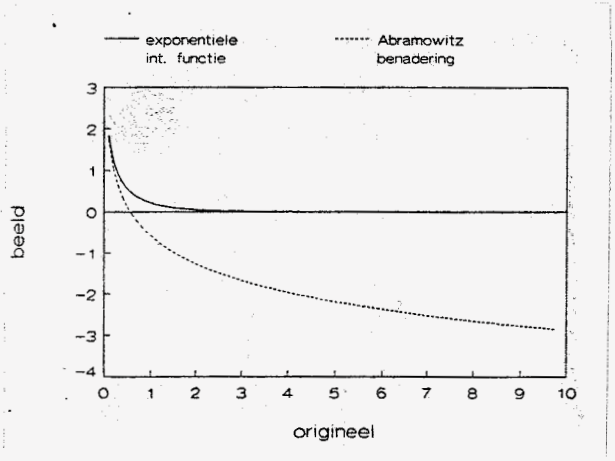
Bij het testen van dit deel van het programma bleek het oplossen veel rekentijd te kosten. Aangezien deze integraalfunktie in het programma meerdere keren wordt opgeroepen, is er een snellere oplossing gezocht. Een mogelijke oplossing is de funktie één keer voor een eindig aantal punten uit te rekenen en deze waarden dan in een tabel op te slaan. Een willekeurige waarde kan dan later door middel van interpolatie gevonden worden. Een probleem hierbij is echter dat men vantevoren niet weet binnen welk interval de originelen vallen en dat de funktie $E_1(x)$ een asymptoot heeft. Een betere oplossing is om voor kleine x de exponentiële integraalfunktie te benaderen met een afgekapte reeksontwikkeling (Abramowitz en Stegun [1973]):

$$E_1(x) \approx -\gamma - \ln(x) \quad \text{waarin } \gamma = 0.5772 \text{ de konstante van Euler is.}$$

Uit figuur 3.1 blijkt dat voor $x<0.05$ de benadering voldoet. Voor $x>10$ stellen we de funktie gelijk aan nul. Voor overige waarden van x wordt de integraal als boven beschreven numeriek opgelost. Dit geeft een zeer grote tijdwinst, aangezien juist voor kleine x–waarden het integreren moeizaam verloopt in verband met de vertikale asymptoot voor x=0. (zie fig. 3.2)

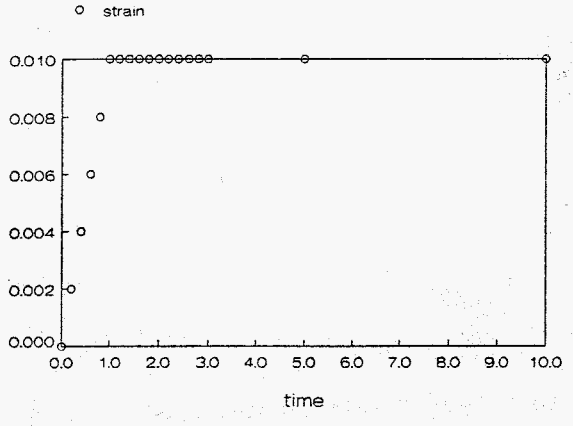


figuur 3.1: Abramowitz benadering

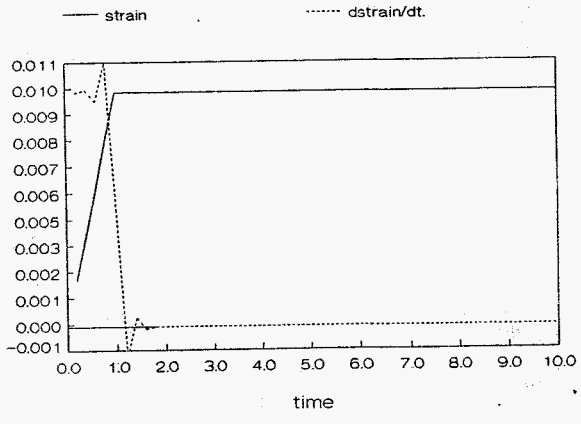


figuur 3.2: Abramowitz benadering

De invoer voor dit programma bestaat uit een array met rekken op equidistante tijdstippen. Bij het oplossingsproces zijn ook waarden voor de rek op tussenliggende tijdstippen nodig. Hiervoor wordt aan de array met rekken een array met tijdstippen toegevoegd. Een waarde van de rek op een willekeurig tijdstip kan dan gevonden worden door middel van interpolatie. Het toepassen van de "cubic spline interpolation" biedt in dit geval verschillende voordelen (Numerical Recipes). Spline interpolatie is stabieler dan interpolatie met polynomen,d.w.z. minder kans op wilde oscillaties tussen getabelleerde punten. De afgeleide van de resulterende funktie is kontinu, zodat de ook integrand van de buitenste integraal kontinu is (zie formule 2.3). Een kontinue integrand heeft als voordeel dat de uitkomst sneller konvergeert naar een konstante waarde en het rekenproces dus minder tijd vergt. De interpolatie wordt uitgevoerd door twee procedures. De tabel met tijden en rekken wordt ingevoerd in procedure SPLINE. In SPLINE wordt op ieder getabelleerd tijdstip de tweede afgeleide van de rek naar de tijd toegevoegd. Dit gebeurt eenmalig bij iedere nieuwe invoer van rekken. Procedure SPLINT geeft als uitvoer de geïnterpoleerde waarde van de rek en de eerste afgeleide, die nodig is om de afgeleide van de elastische respons uit te rekenen. In de volgende figuren wordt de werking van deze twee procedures getoond. Figuur 3.3 toont de invoer van procedure SPLINE : rekken bij equidistante tijdstippen. Figuur 3.4 geeft de uitvoer van procedure SPLINT : rekken en eerste tijdsafgeleiden hiervan op tussenliggende tijdstippen.



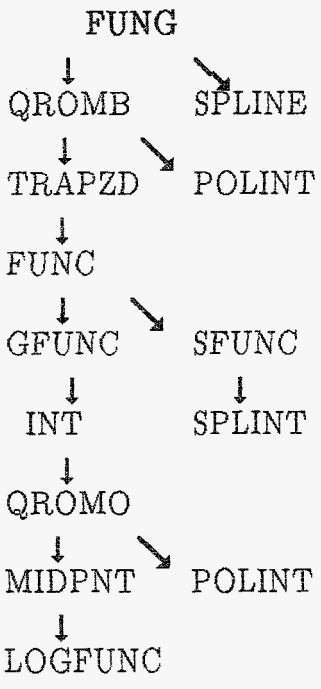
figuur 3.3: invoer SPLINE



figuur 3.4: uitvoer SPLINT

HOOFDSTUK 4 PROGRAMMA BESCHRIJVING

Het programma heeft de volgende globale struktuur:



Het programmadeel FUNG is geïmplementeerd in het bestaande parameterschat programma. De rest is ondergebracht in de unit FUNGTOOL en wordt door FUNG opgeroepen. FUNG roept èènmaal SPLINE op en berekent daarna met behulp van QROMB de spanning. Hieronder volgt een lijst met de gebruikte procedures en functions en een korte uitleg:

SPLINE — Berekent de tweede afgeleide van de rek.

QROMB — Berekent de spanning op een bepaald tijdstip met behulp van de romberg integratie.

TRAPZD — Berekent de integraal door deze in een aantal intervallen op te delen. Dit aantal wordt door QROMB bepaald.

POLINT — POLINT staat voor polynoom interpolatie. QROMB/QROMO

geeft als invoer een tabel met stapgrootten en bijbehorende integraaloplossingen uit TRAPZD. POLINT extrapoleert deze waarden naar een oplossing voor stapgrootte nul en geeft hierbij een schatting van de gemaakte fout.

FUNC Dit is de te integreren funktie, die is opgesplitst in twee delen. (formule 2.3)

GFUNC De gereduceerde relaxatiefunktie. (formule 2.4)

SFUNC De tijdsafgeleide van de elastische respons.

INT De exponentiële integraalfunktie. (formule 3.2)

SPLINT Interpoleert waarden van de rek en berekent diens afgeleiden.

QROMO Een aangepaste procedure voor de romberg integratie van oneigenlijke integralen.

MIDPNT Berekent een integraal door oppervlakten van trapeziums op te tellen. In tegen stelling tot TRAPZD gebruikt MIDPNT niet de grenzen van een interval.

LOGFUNC De integrand van de exponentiële integraalfunktie. (formule 3.1)

Enkele belangrijke variabelen:

t1par = τ_1 = x(1)
t2par = τ_2 = x(2)
apar = A = x(3)
bpar = B = x(4)
kpar = K = x(5)
} De materiaalparameters

jmax1 : Bepaalt de maximale verfijning in QROMB. Het aantal integratie–intervallen is maximaal $2^{(\text{jmax1}-1)}$.

jmax2 : Bepaalt de maximale verfijning in QROMO. Het aantal integratie–intervallen is maximaal $3^{(\text{jmax2}-1)}$.

eps1 : De gewenste geschatte nauwkeurigheid in QROMB. Het afbraak–kriterium voor verdere intervalverfijning is: {schatting van

de integratiefout} < eps1 · {uitkomst v.d. integraal}

eps2 : De gewenste geschatte nauwkeurigheid in QROMO.

yp1,ypn : De randvoorwaarden voor SPLINE ; de eerste tijdsafgeleide van de rek in begin en eindpunt. Geeft men hier een waarde in groter dan 0.99E30 dan wordt de tweede afgeleide nul gesteld.

Voor programmatuur zie bijlage A.

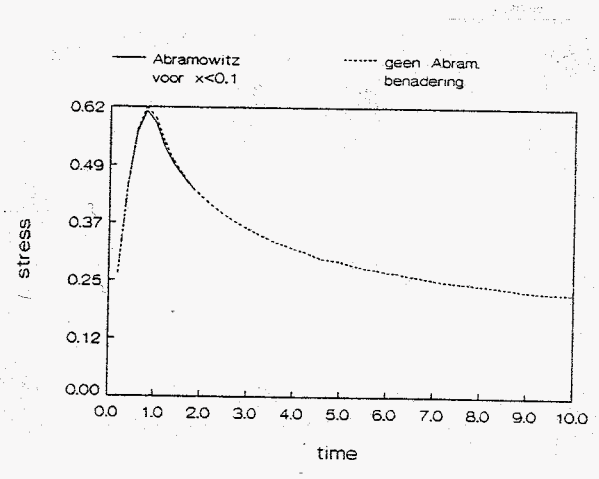
HOOFDSTUK 5 TESTBEREKENINGEN

In de figuren 5.1 en 5.2 staan de resultaten van een berekening met en zonder een Abramowitz benadering voor de exponentiële integraalfunktie. Als rekfunktie is hiervoor gekozen :

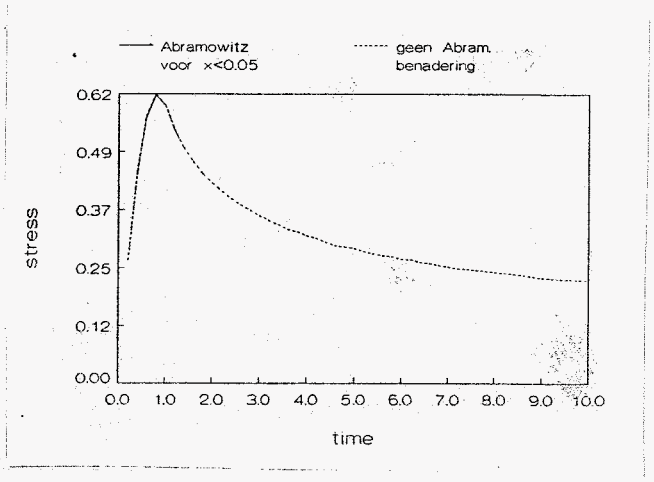
$\epsilon=\sin(t\cdot{}^1/_2\pi)\cdot 0.01$ als $t<1$

$\epsilon=0.01$ als $t\geq 1$

A=100, B=1, K=1, $\tau_1=0.1$, $\tau_2=10$.

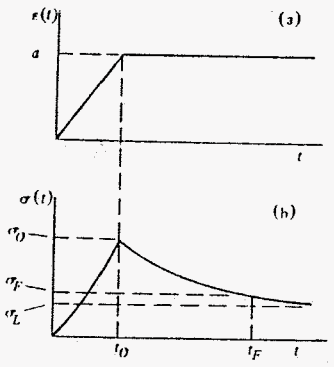


figuur 5.1: met en zonder Abramowitz benadering



figuur 5.2: met en zonder Abramowitz benadering

Vervolgens wordt de analytische oplossing van een probleem vergeleken met de numerieke. We bekijken het volgende rekverloop:



figuur 5.3 spanning en rek als funktie van de tijd (uit I Nigul & U Nigul [1987])

De analytische oplossing voor dit geval luidt: (I. Nigul en U. Nigul [1987])

$$\sigma(t) = a\cdot E_0\cdot[(t/t_0) + K\cdot\tilde{G}(t)] \quad \text{als } 0<t<t_0$$

$$\sigma(t) = a\cdot E_0\cdot[1 + K\cdot\tilde{F}(t)] \quad \text{als } t>t_0$$

$$\tilde{G}(t)=-G(\tau_1)+G(\tau_2),$$

$$\tilde{F}(t)=-F(\tau_1)+F(\tau_2),$$

$$G(\tau_j)=(t/t_0)\cdot E_1(\tau_j)-g(\tau_j),$$

$$F(\tau_j)=(t/t_o)\cdot E_1(\tau_j)-[(t/t_0-1]E_1(\tau_j-x_j)-f(\tau_j),$$

$$g(\tau_j)=x_j^{-1}\cdot(e^{-\tau_j}-1), \quad j\in 1,2,$$

$$f(\tau_j)=x_j^{-1}\cdot(e^{-\tau_j}-e^{-\tau_j+x_j}), \quad j\in 1,2,$$

$$E_0 = A/[1+K\cdot\ln(\tau_2/\tau_1)]$$

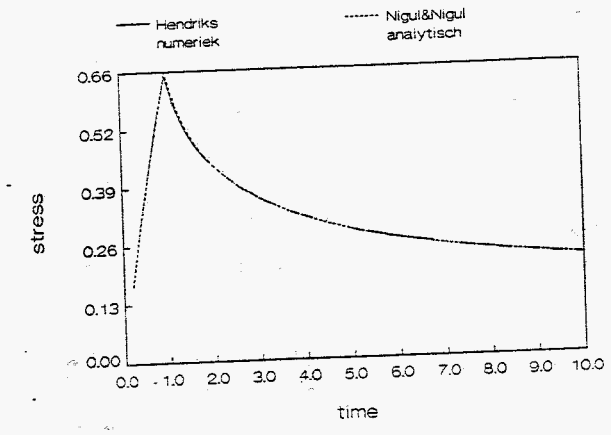
E_1 is de exponentiële integraalfunktie.

De numerieke oplossing is bepaald met twee verschillende epsl. Voor de verschillende variabelen zijn de volgende waarden gekozen:

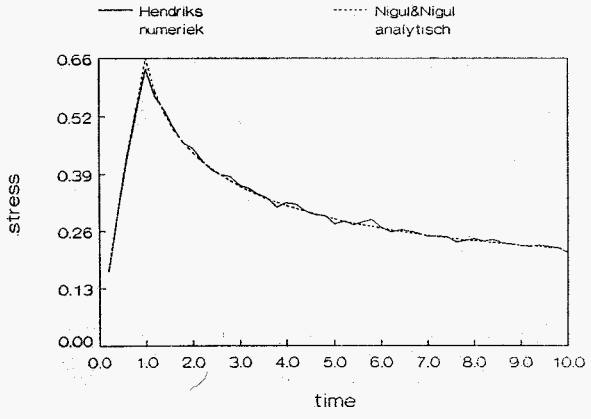
t_0=1s τ_1=0.1 τ_2=10 K=1

en het verband tussen spanning en rek is lineair gekozen: $\sigma^e=100\cdot\epsilon(t)$

In de volgende figuren staan de resultaten.



figuur 5.4: epsl = 10^{-4}



figuur 5.5: epsl = 10^{-5}

HOOFDSTUK 6 KONKLUSIES EN VOORTGANG

De benadering volgens Abramowitz kan toegepast worden voor originelen kleiner dan 0.05. De rekentijd is dan aanzienlijk kleiner en de nauwkeurigheid is voldoende. De procedures SPLINE en SPLINT werken snel en maken het mogelijk rekken op willekeurige tijdstippen te bepalen. De totale integraalberekening verloopt nauwkeurig en voldoende snel. De nauwkeurigheid eps1 moet met enige voorzichtigheid gekozen worden. Een te kleine waarde leidt al snel tot lange rekentijden. De werkelijke relatieve afwijking zal groter zijn dan de eps1. Dit komt doordat de integrand niet exact bepaald kan worden. Hiervoor zijn twee oorzaken: Het rekverloop moet ge'nterpoleerd worden tussen de tijdstippen waar deze wel bekend is. Een onderdeel van de integrand is zelf een integraal die numeriek benaderd wordt.

In de toekomst zal m.b.v. het parameterschatprogramma het model gekonfronteerd worden met resultaten van experimenten. Het kan noodzakelijk blijken het model van Fung aan te passen met een extra term, zoals is voorgesteld door Oomens [1989].

Literatuur

M Abramowitz en J.A. Stegun; Handbook of mathematical functions 1973

Y.C.B. Fung; Stress–strain–history relations of soft tissues in simple elongation. Hfdst.7 in : Biomechanics, its foundations and objectives.

J.M.R.J. Huyghe; Non–lineair finite element models of the beating left ventricle and the intramyocardial coronary circulation, proefschrift TUE, 1986

M.A.M. van Lankveld; Parameterschatten aan niet–lineaire mengselmodellen m.b.v. het Kalmanfilter, afstudeerverslag TUE 1989

I. Nigul & U. Nigul; On algorithms of evaluation of Fung's relaxation function parameters, Journal of Biomechanics, Vol. 20, no. 4, pp 343–352, 1987

C.W.J.Oomens; mondelinge kommunikatie juli 1989

W.H. Press, B.P. Flannery, S.A. Teukolsky en W.T. Vetterling; Numerical Recipes the art of scientific computing

A.A.H.J. Sauren en C.W.J. Oomens; Biologische materialen, collegediktaat TUE nr 4589

BIJLAGE PROGRAMMATUUR

```
UNIT FUNGTOOL;
{$N+}

INTERFACE

CONST
   n         = 10;
   maxtim    =200;
TYPE
   glnarray = array [1..n] of double;
   arrtim = array [0..maxtim] of double;

VAR
   apar,bpar,t1par,t2par,kpar : double;
   tbegin,tend                : double;
   stress                     : double ;
   eps1,eps2                  : double;
   jmax1,jmax2,glit1,glit2    : integer;
   stress_file,int_file       : text;
   strain_file                : text;

PROCEDURE spline(x,y       : arrtim;
                 n         : integer;
                 yp1,ypn   : double;
                 VAR y2    : arrtim);

PROCEDURE qromb(a,b                                    : double;
                jmax,jmax2                             : integer;
                eps,eps2                               : double ;
                t1par,t2par,apar,bpar,kpar             : double ;
                m                                      : integer ;
                xa,ya,y2a                              : arrtim ;
                VAR  ss                                : double   );


IMPLEMENTATION
```

```
PROCEDURE spline(x,y        : arrtim;
                 n          : integer;
                 yp1,ypn    : double;
                 VAR y2     : arrtim);
(* Programs using routine SPLINE must define the type
TYPE
   arrtim = ARRAY [1..m] OF double;
in the main routine. *)
(* procedure spline berekent een array met tweede afgeleiden *)
VAR
   i,k: integer;
   p,qn,sig,un: double;
   u: arrtim;
BEGIN
   IF (yp1 > 0.99e30) THEN BEGIN
      y2[1] := 0.0;
      u[1] := 0.0
   END ELSE BEGIN
      y2[1] := -0.5;
      u[1] := (3.0/(x[2]-x[1]))*((y[2]-y[1])/(x[2]-x[1])-yp1)
   END;
   FOR i := 2 to n-1 DO BEGIN
      sig := (x[i]-x[i-1])/(x[i+1]-x[i-1]);
      p := sig*y2[i-1]+2.0;
      y2[i] := (sig-1.0)/p;
      u[i] := (y[i+1]-y[i])/(x[i+1]-x[i])
         -(y[i]-y[i-1])/(x[i]-x[i-1]);
      u[i] := (6.0*u[i]/(x[i+1]-x[i-1])-sig*u[i-1])/p
   END;
   IF (ypn > 0.99e30) THEN BEGIN
      qn := 0.0;
      un := 0.0
   END ELSE BEGIN
      qn := 0.5;
      un := (3.0/(x[n]-x[n-1]))*(ypn-(y[n]-y[n-1])/(x[n]-x[n-1]))
   END;
   y2[n] := (un-qn*u[n-1])/(qn*y2[n-1]+1.0);
   FOR k := n-1 DOWNTO 1 DO BEGIN
      y2[k] := y2[k]*y2[k+1]+u[k]
   END
END;
```

```
PROCEDURE splint(xa,ya,y2a: arrtim;
                 n        : integer;
                 x        : double;
                 VAR y,y1 : double);
(* Programs using routine SPLINT must define the type
TYPE
   arrtim = ARRAY [1..m] OF double;
in the main routine. *)
(* procedure splint berekent de functiewaarde en de eerste afgeleide *)
VAR
   klo,khi,k: integer;
   h,b,a: double;
BEGIN
   klo := 1;
   khi := n;
   WHILE (khi-klo > 1) DO BEGIN
      k := (khi+klo) DIV 2;
      IF (xa[k] > x) THEN khi := k ELSE klo := k
   END;
   h := xa[khi]-xa[klo];
   IF (h = 0.0) THEN BEGIN
      writeln ('pause in routine SPLINT');
      writeln (' ... bad XA input'); readln END;
   a := (xa[khi]-x)/h;
   b := (x-xa[klo])/h;
   y := a*ya[klo]+b*ya[khi]+
      ((a*a*a-a)*y2a[klo]+(b*b*b-b)*y2a[khi])*(h*h)/6.0;
   y1:=(ya[khi]-ya[klo])/h -
       (3*a*a-1)*h*y2a[klo]/6 +
       (3*b*b-1)*h*y2a[khi]/6;
END;
```

```
FUNCTION logfunc(x : double ) : double;
VAR
   res : double;
BEGIN
   res := -1.0/(ln(x));
   logfunc := res
END;
```

```
PROCEDURE midpnt(a,b: double; VAR s: double; n: integer);
(* Programs using routine MIDPNT must supply a function
logfunc(x:double):double which is to be integrated. They must also
declare an iteration counter
VAR
   glit2: integer; *)
VAR
   j: integer;
   x,tnm,sum,del,ddel: double;
BEGIN
   IF  (n = 1)  THEN BEGIN
      s := (b-a)*logfunc(0.5*(a+b));
      glit2 := 1
   END ELSE BEGIN
      tnm := glit2;
      del := (b-a)/(3.0*tnm);
      ddel := del+del;
      x := a+0.5*del;
      sum := 0.0;
      FOR j := 1 to glit2 DO BEGIN
         sum := sum+logfunc(x);
         x := x+ddel;
         sum := sum+logfunc(x);
         x := x+del
      END;
      s := (s+(b-a)*sum/tnm)/3.0;
      glit2 := 3*glit2
   END
END;
```

```
PROCEDURE polint(xa,ya: glnarray; n: integer;
       x: double; VAR y,dy: double);
(* Programs using routine POLINT must define the type
TYPE
   glnarray = ARRAY [1..n] OF double;
in the main routine. *)
VAR
   ns,m,i: integer;
   w,hp,ho,dift,dif,den: double;
   c,d: glnarray;
BEGIN
   ns := 1;
   dif := abs(x-xa[1]);
   FOR i := 1 to n DO BEGIN
      dift := abs(x-xa[i]);
      IF (dift < dif) THEN BEGIN
         ns := i;
         dif := dift
      END;
      c[i] := ya[i];
      d[i] := ya[i]
   END;
   y := ya[ns];
   ns := ns-1;
   FOR m := 1 to n-1 DO BEGIN
      FOR i := 1 to n-m DO BEGIN
         ho := xa[i]-x;
         hp := xa[i+m]-x;
         w := c[i+1]-d[i];
         den := ho-hp;
         IF (den = 0.0) THEN BEGIN
            writeln ('pause in routine POLINT'); readln END;
         den := w/den;
         d[i] := hp*den;
         c[i] := ho*den
      END;
      IF ((2*ns) < (n-m)) THEN BEGIN
         dy := c[ns+1]
      END ELSE BEGIN
         dy := d[ns];
         ns := ns-1
      END;
      y := y+dy
   END
END;
```

```
PROCEDURE qromo(a,b    : double ;
                jmax   : integer;
                eps    : double;
                VAR ss: double);
(* Programs using routine QROMO must define the type
TYPE
   glnarray = ARRAY [1..n] OF double;
in the main routine, where n is equal to the constant k below. The routir
expfunc(x:double):double in the calling routine must return the value of
function to be integrated.  MIDPNT is choosen at the indicated point belc
QROMO integreert een oneigenlijke integraal samen met POLINT en MIDPNT. *
LABEL 99;
CONST
   k=2;  (*deze was 5*)
   km=1;       (* km=k-1 *)
   jmaxmax=15;        (*jmax2 < jmaxmax*)
VAR
   i,j,jmaxp      : integer;
   dss            : double;
   h,s            : ARRAY [1..jmaxmax] OF double;
   c,d            : glnarray;
BEGIN
   jmaxp := jmax + 1;
   h[1] := 1.0;
   FOR j := 1 to jmax DO BEGIN
(* Here you must choose the appropriate integration method *)
      midpnt(a,b,s[j],j);
      IF  (j >= k)  THEN BEGIN
         FOR i := 1 to k DO BEGIN
            c[i] := h[j-k+i];
            d[i] := s[j-k+i]
         END;
         polint(c,d,k,0.0,ss,dss);
         IF (abs(dss) < (eps*abs(ss))) THEN GOTO 99
      END;
      s[j+1] := s[j];
      h[j+1] := h[j]/9.0
   END;
   writeln('warning in QROMO - too many steps');
99: END;
```

```
FUNCTION int(x      : double;
             jmax2 : integer;
             eps2   : double   ) : double;
VAR
   res : double;
BEGIN
   if x>10 then res:=0 else
      BEGIN
         if x < 0.05 then res := -0.5772 - ln(x)   (*Abramowitz*) else
         qromo(0,exp(-x),jmax2,eps2,res);
      END;
   int := res;
END;
```

```
FUNCTION gfunc(t         : double ;
               jmax2     : integer;
               eps2      : double ;
               t1,t2,k   : double  ) : double;
(*GFUNC is de gereduceerde relaxatiefunctie.*)
VAR
   res : double;
BEGIN
   IF t=0 THEN
      res := 1 / ( 1+k*ln(t2/t1) )
   ELSE
   BEGIN
      res := 1 +  k*( int(t/t2,jmax2,eps2)-int(t/t1,jmax2,eps2) );
      res := res/(1+k*ln(t2/t1))
   END;
   gfunc := res ;
END;
```

```
FUNCTION sfunc( t          : double   ;
                apar,bpar  : double   ;
                xa,ya,y2a  : arrtim ;
                m          : integer) : double;
(*SFUNC is de elastische respons.*)
VAR
   res             : double;
   strain,dstrain  : double;
BEGIN
   splint(xa,ya,y2a,m,t,strain,dstrain);
   res := apar*bpar*exp(bpar*strain);
   res := res*dstrain;
   sfunc := res;
END;
```

```
FUNCTION func( t,tend                                  : double  ;
               jmax2                                   : integer ;
               eps2                                    : double  ;
               t1par,t2par,apar,bpar,kpar              : double  ;
               m                                       : integer ;
               xa,ya,y2a                               : arrtim ) : double;
VAR
   res  : double;
BEGIN
   res := gfunc(tend-t,jmax2,eps2,t1par,t2par,kpar);
   res := res * sfunc(t,apar,bpar,xa,ya,y2a,m) ;
   func := res
END;
```

```
PROCEDURE trapzd(a,b                                    : double  ;
                 jmax2                                  : integer ;
                 eps2                                   : double    ;
                 t1par,t2par,apar,bpar,kpar             : double  ;
                 m                                      : integer ;
                 xa,ya,y2a                              : arrtim;
                 VAR     s : double    ;       n : integer );
(* Programs calling TRAPZD must provide a function
func(x:double):double which is to be integrated. They must
also define the variable
VAR
   glit1: integer;
in the main routine. *)
VAR
   j: integer;
   x,tnm,sum,del: double;
BEGIN
   IF (n = 1) THEN BEGIN
      s := 0.5*(b-a)*(func(a,b,jmax2,eps2,t1par,t2par,apar,bpar,kpar,
                          m,xa,ya,y2a)
                     +func(b,b,jmax2,eps2,t1par,t2par,apar,bpar,kpar,
                          m,xa,ya,y2a));
      glit1 := 1
   END
   ELSE BEGIN
      tnm := glit1;
      del := (b-a)/tnm;
      x := a+0.5*del;
      sum := 0.0;
      FOR j := 1 to glit1 DO BEGIN
         sum := sum+func(x,b,jmax2,eps2,t1par,t2par,apar,bpar,kpar,
                         m,xa,ya,y2a);
         x := x+del
      END;
     s := 0.5*(s+(b-a)*sum/tnm);
      glit1 := 2*glit1
END;
```

```
PROCEDURE qromb(a,b                                          : double  ;
                jmax,jmax2                                   : integer ;
                eps,eps2                                     : double  ;
                t1par,t2par,apar,bpar,kpar                   : double  ;
                m                                            : integer ;
                xa,ya,y2a                                    : arrtim  ;
                VAR ss                                       : double  );
(* Programs using routine QROMB must define type
TYPE
   glnarray = ARRAY [1..n] OF double;
just as for routine POLINT which it calls. In this case
n should have a value no smaller than the constant k below.
qromb integreert de functie func m.b.v. trapzd en polint. *)
LABEL 99;
CONST
   k=5;  (* deze was 5 *)
   jmaxmax=20;   (*jmax1<jmaxmax ; vaste grens van array *)
VAR
   i,j,jmaxp       : integer;
   dss             : double;
   h,s             : ARRAY[1..jmaxmax] OF double;
   c,d             : glnarray;
BEGIN
   jmaxp := jmax + 1;
   h[1] := 1.0;
   FOR j := 1 to jmax DO BEGIN
      trapzd(    a,    b,jmax2, eps2,t1par,t2par, apar, bpar,
              kpar,    m,   xa,   ya,  y2a, s[j],    j);
      IF  (j >= k)  THEN BEGIN
         FOR i := 1 to k DO BEGIN
            c[i] := h[j-k+i];
            d[i] := s[j-k+i]
         END;
         polint(c,d,k,0.0,ss,dss);
         IF (abs(dss) < eps*abs(ss)) THEN GOTO 99
      END;
      s[j+1] := s[j];
      h[j+1] := 0.25*h[j]
   END;
   writeln('warning in QROMB - too many steps');
99:   END;

END.
```