N.4.5 Kennis nemen van leerstof en voorbeelden van N.4.5. De hierna volgende leerstof over 1. NULPUNTSPROGRAMMA, 2. VAN PASCAL NAAR BASIC en 3. NULPUNTSMETHODEN ALGEMEEN doornemen. Vergeet niet om het nulpuntsmethoden programma in de rekenmachine te zetten.

---

als N4.5.3. NULPUNTSPROGRAMMA

In het boek "Wiskunde voor het HBO" staan de volgende nulpuntsmethoden:
halveringsmethode;
methode van Newton-Raphson (raaklijnmethode);
koordenmethode;
methode van successieve substitutie (substitutiemethode).

Voor het berekenen van een nulpunt gebruiken we een BASIC-programmeerbare rekenmachine. In het nulpuntsprogramma moet worden vastgesteld welke methode gewenst is. Soms is dat niet zo eenvoudig, en zullen we wat willen proberen. Daartoe moet het programma de mogelijk bieden, om --- na het proberen van de ene methode en kennisname van de uitkomst daarvan --- opnieuw een keuze te maken. De nulpuntsbepaling moet dus herhaald kunnen worden totdat we klaar zijn: REPEAT nulpuntsbepaling UNTIL klaar.

Een nulpuntsmethode berekent, op grond van gegeven beginwaarden, een nulpunt x, dwz de waarde x waarvoor f(x)=0. Numeriek gezien heeft f(x)=0 nooit een oplossing, omdat het aantal cijfers van de rekenmachine beperkt is. Daarom gebruiken we een getal "nul", dat die nauwkeurigheid aan geeft (of de nauwkeurigheid die we zelf voldoende vinden). Daaruit volgt, dat de eindwaarde x een fout heeft (hoe groter "nul", hoe groter de "fout"). De nulpuntsmethode berekent ook een schatting van die fout. Resumerend: het programma heeft een invoer "nul", en een uitvoer "fout".

Het nulpunten programma wordt vastgelegd in "pseudo-PASCAL". Het "pseudo" wijst erop, dat we niet precies de taalregels van PASCAL volgen, maar wel de struktuur. Het voordeel van gestruktureerde taal is, dat we altijd kunnen kontroleren of het programma korrekt is, ook al kennen we de inhoud van gebruikte delen niet. Het is bovendien direkt vertaalbaar naar BASIC.
De programheading is: **naam(**invoerlijst**;**uitvoerlijst**)**;

```
PROGRAM Nulpunt(methode,nul,beginwaarde(n);x,fout);
  VAR nul : nauwkeurigheid in f;      { als abs(f)<=nul , dan f voldoende 0 }
methode: nulpuntsmethode;          {(halvering,raaklijn,koorde,substitutie)}
x  : eindwaarde x;              { dus f=f(x) is voldoende klein       }
      fout: nauwkeurigheid in x;     { beste foutschatting bij x           }
  FUNCTION f(x);
  FUNCTION klaar:BOOLEAN;            { bepaalt of er moet worden herhaald  }
BEGIN
  REPEAT
    read(methode);                   { lees te gebruiken methode           }
    read(nul);                       { lees "nul"                          }
    CASE methode OF
      halvering  : Halveringsm(f(x),nul,beginwaarden;x,fout)   ;
      raaklijn   : Raaklijnm(f(x),f'(x),nul,beginwaarde;x,fout);
      koorde     : Koordem(f(x),nul,beginwaarden;x,fout)       ;
      substitutie: Substitutiem(f(x),g(x),nul,beginwaarde;x,fout)
      END;
    write(x);                        { eindwaarde van x, dus nulpunt       }
    write(fout)                      { foutschatting bij x: x+fout=nulpunt }
    UNTIL klaar
END.
```

Een programma in PASCAL wordt statement voor statement, regel voor regel, vertaald naar BASIC. In BASIC worden regels opvolgend genummerd (hoeft niet aansluitend);bijvoorbeeld: 10,20,30,35,40; er mogen meerdere commando's op een regel, mits ze worden gescheiden door een ':'(dubbele punt).
Een funktie wordt in BASIC gedeclareerd als een subroutine: een programmadeel dat met RETURN wordt afgesloten.

```
FUNCTION f(x); {f(x)=sin x }      900:F= SIN (X)
                                  910:RETURN
```

De funktiewaarde kan dan worden bepaald met het commando 'GOSUB 900'. Gebruik dat bijvoorbeeld om funktiewaarden uit te proberen (of om een schets te kunen maken van de grafiek van de funktie):

Tik in de **PROmode** in : `1000 INPUT X:GOSUB 900:PRINT "F=";F:GOTO 1000 <enter>`

Tik in de **RUNmode** in : `RUN 1000 <enter>`

Dan komt er telezen : ? ,waarna de x-waarde wordt ingegeven, en f(x) bepaald.
Voortdurend wordt om nieuwe getallen (?) gevraagd, dankzij "GOTO 1000".

Sommige PASCAL procedures hebben al een direkte vertaling naar een BASIC commando; bijv

```
read(nul)                         30:INPUT "Nauwkeurigheid f=";NUL
write(x)                          60:PRINT "X=";X
```

Een PASCAL REPEAT-UNTIL herhaallus vertalen we als volgt naar BASIC:

```
REPEAT
  statement;                      30:statement
  statementlist                   ..:statementlist
UNTIL voorwaarde                  80:IF NOT voorwaarde THEN GOTO 30
```

Bij onze rekenmachine wacht de machine altijd wanneer een waarde moet worden ingevoerd. Op dat moment kan de programmauitvoering worden afgebroken. Daarvan maken we gebruik door na de statementlus altijd terug te keren.
De PASCAL keuze struktuur CASE-OF-END kan worden vertaald naar BASIC met behulp van het commando: ON telvariabele GOSUB (of GOTO)regelnummerlijst

```
CASE methode OF                    50:ON MT GOSUB 100,200
  halvering: halveringsm;          { Als MT=1 wordt '100' uitgevoerd}
  raaklijn : raaklijnm;            { als MT=2 '200' enzovoort.}
  END;
```

```
PROGRAM Nulpunt;                   | 10:CLEAR:PRINT "NULPUNT"
  FUNCTION f(x);f'(x);fs(x)        | 20:PRINT "F(X),FA,FS OP 900,920,940"
BEGIN                              |
  REPEAT                           |
    read(nul);                     | 30:INPUT "NUL=";NUL
    read(methode);                 | 40:INPUT "H=1,R=2,K=3,S=4; MT=";MT
    CASE methode OF                | 50:ON MT GOSUB 100,200,300,400
      halvering  : Halveringsm ;   |
      raaklijn   : Raaklijnm   ;   |
      koorde     : Koordem     ;   |
      substitutie: Substitutiem;   |
      END;                         |
    write(x);                      | 60:PRINT "EIND X=";X
    write(fout)                    | 70:PRINT "FOUTSCH.=";FT
    UNTIL klaar                    | 80:GOTO 30
END.                               | 90:END
```

Test het programma eerst met de 'MT=1' subroutine '100:X=1:FT=1' enz.

3.1 Gemeenschappelijke methodiek

De methoden verschillen, maar hebben ook veel gemeenschappelijk. Zo hebben alle methoden een test om te zien of een punt van de grafiek (x,f) een nulpunt van de funktie geeft: **test** abs(f)<=0 . Als de test negatief uitvalt, dan zal de nulpuntsmethode in **volgende_x** een volgende x-waarde xv aangeven, die in de procedure een **volgende_punt** wordt omgezet in een te testen 'punt' (x,f). De test zit dus in een herhaalstruktuur met test vooraf:

```
  WHILE NOT test_goed DO
    BEGIN
     volgende_x;
     volgende_punt;
    END;
```

Voordat we de herhaallus kunnen ingaan moeten we het eerste testpunt zelf aangeven. Dat doen we, door de xv-waarde in te voeren met een procedure **begin_x** , en die xv waarde met volgende_punt in een punt (x,f) om te zetten.
Nadat we de herhaallus verlaten hebben zal het programma nog de fout moeten berekenen in de procedure **Foutschatting**. Dat kan alleen maar, als we over voldoende gegevens beschikken wat betreft de laatste testpunten. Die gegevens moeten dus ook worden bijgegehouden in de procedure **volgende_x**(x,f;xv). Omdat de allereerste test direkt positief zou kunnen uitvallen moeten we er bovendien voor zorgen, dat we in het begin voldoende gegevens meegeven om de fout te kunnen bepalen; dat doen we eveneens in de procedure **begin_x**.

Het programma van de nulpuntsmethode krijgt zo de volgende struktuur.

```
PROGRAM Nulpuntsmethode(f(x),nul,beginwaarden;x,fout);
  VAR n   : nummer test;               { bij start n=0, verder n=1,2,enz     }
      x   : x-waarde testpunt;         { test: als abs(f)<=nul dan eindwaarde}
      f   : f-waarde testpunt;         { f=f(x)                              }
      xv  : volgende x;                { xv is de volgende x waarde          }
     nul  : nauwkeurigheid in f;       { bepaalt wanneer f voldoende 0  is   }
fout: nauwkeurigheid in x;       { schatting; hangt af van benadering  }
BEGIN
  n:=0;begin_x(xv);                    { lees beginwaarden in en foutgegevens}
  n:=n+1;volgende_punt(xv;x,f);        { x=xv en bepaal f=f(x)               }
  WHILE NOT abs(f)<=nul DO             { zolang f onvoldoende nul is         }
    BEGIN
    volgende_x(x,f;xv);                { bepaal xv, en houd foutgegevens bij }
    n:=n+1;volgende_punt(xv;x,f);      { stel x=xv en bepaal f =f(x)         }
    END;                               { nu is f voldoende nul geworden      }
  Foutschatting(fout)                  { bepaal fout                         }
END.
```

Feitelijk kan n worden weggelaten, maar het is voor de kontrole op de korrektheid van het programma beter om n te laten staan. Wil je de waarde van n wel afdrukken, dan moet je zelf het programma aanpassen.

Welke gegevens verder nodig zijn voor het bepalen van nulpuntsbenadering xv en voor het bepalen van de fout, hangt af van de gebruikte nulpuntsbenaderingsmethode. We zullen die nulpuntsmethoden apart behandelen. Dan zal duidelijk worden hoe de ontbrekende gedeelten moeten worden ingevuld.

Het PASCAL programma, voor een of andere nulpuntsmethode, zal nu worden vertaald naar een BASIC programma. Daartoe zullen we PASCAL programmastappen **konsekwent** vertalen naar bijbehorende BASIC commando's. De specifieke gedeelten, die voor iedere nulpuntsmethode weer anders kunnen zijn,
hebben we open gelaten. Het is een goede oefening om die **zelf** te vertalen en te vergelijken met de achter de betreffende nulpuntsmethode gegeven vertaling.

In het boven gegeven programma komt de PASCAL herhaal struktuur, met voorwaarde vooraf, de WHILE-DO opdracht voor. Deze wordt naar BASIC vertaald met behulp van de keuzestruktuur IF THEN, aangevuld met twee terugkeer commando's GOTO:

```
WHILE NOT Voorwaarde DO          | 130:IF Voorwaarde THEN GOTO 180
  BEGIN Opdracht                 | 140:Opdracht
  END;                           | 170:GOTO 130
Vervolgopdracht                  | 180:Vervolgopdracht
```

Zolang nog niet aan de voorwaarde is voldaan, moet de opdracht worden uitgevoerd; daarna wordt, tengevolge van het commando GOTO 130, gekeken of al aan de voorwaarde wordt voldaan. Als dat zo is, wordt middels het commando GOTO 180 vervolgt met de vervolgopdracht.

```
          PASCAL                                  |          BASIC
                                                  |
PROCEDURE Nulpuntsmethode(f(x),nul                |
  ,beginwaarden; eindwaarde x,fout);              |
 BEGIN                                            |
   Begin_x(xv);                                   | 100:INPUT "XV=";XV
   n:=1;Volgende_punt(xv;x,f);                    | 120:N=1:GOSUB 500
   WHILE NOT abs(f)<=nul DO                       | 130:IF ABS(F)<= NUL THEN GOTO 180
     BEGIN                                        |
      Volgende_x(x,f;xv);                         | 140: { volgende gegevens }
      n:=n+1;Volgende_punt(xv;x,f)                | 160:N=N+1:GOSUB 500
     END;                                         | 170:GOTO 130
   fout:={methode afhankelijk}                    | 180:FT= { fout }
 END.                                             | 190:RETURN
```

De procedure **Volgende_punt** is bij alle nulpuntsmethoden gelijk, en kan dus als een globale procedure van het programma nulpunt worden gedeclareerd. Dat is al door de regelnummering aangegeven. Het testpunt (x,f) moet worden berekend, uitgaande van de waarde xv. Programma volgende_punt ,en de vertaling, luiden dan als volgt:

```
PROCEDURE Volgende_punt(xv;x,f)                  |
BEGIN                                            |
   x:=xv; write(x)                               | 500:X=XV:PRINT "X=";X
   f:=f(x);                                      | 510:GOSUB 900
END;                                             | 520:RETURN
```

De x-waarde wordt afgedrukt, omdat het prettig is als programmagebruiker om te weten hoe het nulpunt wordt benaderd. Als een grote snelheid wordt vereist kan de schrijfopdracht vervallen.

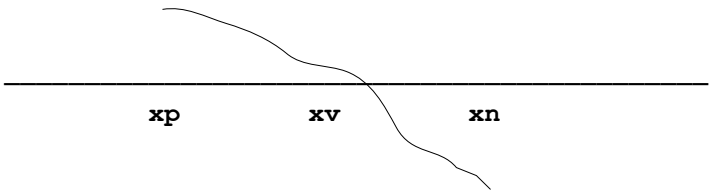
N.4.6 Leerstof 4.6.1 bestuderen en in voorbeeld vooral de tabel doornemen. Maken opgaven: 1, 2, en 3.
Neem de volgende leerstof 4.1 Halveringsmethode door.
Maken opgaven: 4 en 5.

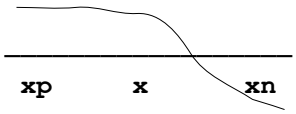
---

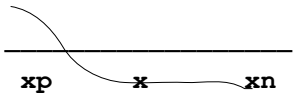
als N4.6.4 NULPUNTSMETHODEN

4.1 Halveringsmethode

```
PROGRAM  nulpuntHalvering (f(x),nul,xp,xn;x,fout);

{Dit programma bepaalt de oplossing x van de vgl f(x)=0, met nauwkeurigheid }
{nul in f,uitgaande van de x-waarden xp resp. xn, waar de funktie positief  }
{resp negatief is, en die daarom tenminste een nulpunt insluiten.           }
{De volgende testwaarde xv wordt halverwege xp en xn genomen, waardoor het  }
{ nulpunt steeds verder wordt ingesloten.                                   }

  VAR x  : testpunt x-waarde;
      f  : testpunt f-waarde;          { f=f(x) ; getest op f voldoende nul }
      n  : indexnummer testpunt;       { n=0,1,2,..                         }
      nul: nauwkeurigheid in f;        { f voldoende 0 als abs(f) <= nul    }
      xp : positief punt x-waarde;     { sign(f(xp)) = +1 dicht bij nulpunt }
      xn : negatief punt x-waarde;     { sign(f(xn)) = -1 dicht bij nulpunt }
      xv : volgende x-waarde;          { xv=(xp+xn)/2 halverwege xp en xn   }
     fout : foutschatting voor x;      { als f>0 , dan is het nulpunt tussen}
                                       { x en xn, dus geschat fout=(xn-x)/2;}
                                       { anders is het nulpunt tussen x enxp}
                                       { met: fout=(x-xp)/2=-(xn-x)/2       }

  FUNCTION  f=f(x);
```








```
BEGIN
  {Begin_x}
  read(xp);                        { f(xp) > 0 noodzakelijk            }
  read(xn):                        { f(xn) < 0 noodzakelijk            }
  xv:=(xn+xp)/2;                   { berekende xv waarde!              }
  n:=1; Volgende_punt(xv;x,f);
  WHILE NOT abs(f) <= nul DO
    BEGIN
      {Volgende_x}
      IF sign(f)= +                { als het teken van f positief is,  }
        THEN  xp:=x                { dan is x een positiefpunt dat     }
        ELSE  xn:=x;               { dichter bij nulpunt is dan xp;    }
      xv:=(xn+xp)/2;
      n:=n+1; Volgende_punt(xv;x,f)
    END;
  {Foutschatting}
  fout:=sign(f)(xn-x)/2            { werkelijke fout is hooguit 2*fout }
END.
```

De vertaling van PASCAL IF-THEN-ELSE naar BASIC commando IF THEN, aangevuld met GOTO commando's, gaat als volgt:

```
  IF voorwaarde                          |  n1  IF NOT Voorwaarde THEN GOTO n3
    THEN Opdracht_ja                     |  n2  Opdracht_ja:GOTO n4
    ELSE Opdracht_nee;                   |  n3  Opdracht_nee
  Vervolgopdracht                        |  n4  Vervolgopdracht
```

Kontroleer, dat de ontkenning NOT bij de voorwaarde moet, als we de volgorde van de opdrachten gelijk willen houden. Funkties voor het teken sign(getal) en voor de absolute waarde abs(getal) van een getal, kent BASIC als de funkties SGN en ABS. Het teken wordt weergegeven door de waarden 1 voor + en -1 voor -.

```
PROCEDURE nulpuntHalvering               |
                                         |
BEGIN                                    |
  {Begin_x}                              |
  read(xp);                              |  100:INPUT "XP=";XP
  read(xn);                              |  105:INPUT "XN=";XN
  xv:=(xn+xp)/2;                         |  110 XV=(XN+XP)/2
  n:=1; Volgende_punt(xv;x,f);           |  120:N=1:GOSUB 500
  WHILE NOT abs(f) <= nul DO             |  130:IF ABS(F)<= NUL THEN GOTO 180
    BEGIN                                |
      {Volgende_x}                       |
      IF sign(f)= +                      |  140:IF SGN(F)=-1 THEN GOTO 150
        THEN  xp:=x                      |  145:XP=X:GOTO 155
        ELSE  xn:=x;                     |  150:XN=X
      xv:=(xn+xp)/2;                     |  155:XV=(XP+XN)/2
      n:=n+1;Volgende_punt(xv;x,f)       |  160:N=N+1:GOSUB 500
    END;                                 |  170:GOTO 130
  {Foutschatting}                        |
  fout:= sgn(f)(xn-x)/2                  |  180:FT=SGN(F)*(XN-X)/2
END.                                     |  190:RETURN
```

Merk op, dat in BASIC de variabele-naam fout niet letterlijk is vertaald. Op de meeste rekenmachines zijn namelijk van de namen alleen de eerste twee letters bepalend. Dus: FOUT en FO worden als dezelfde naam opgevat. We korten om misverstanden te voorkomen fout af als FT.

Het programma is voor jezelf wat vriendelijker wat betreft de invoer van de waarden xp en xn als de funktiewaarde wordt bepaald, en x pas wordt geaccepteerd als f werkelijk >0 resp <0 is:

```
REPEAT read(x);                          |  100:INPUT "XP=";X
  f:= f(x);                              |  101:GOSUB 900
  UNTIL f>0;                             |  102:IF NOT F>0 THEN GOTO 100
xp := x                                  |  103:XP=X
```

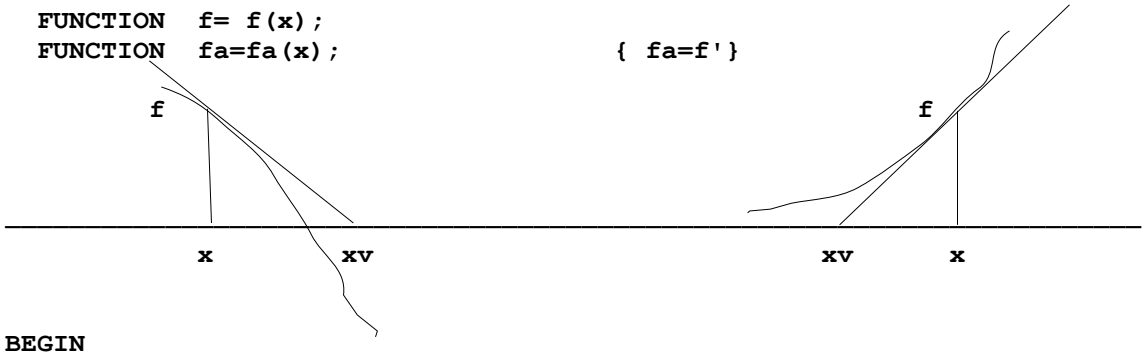
Het is verstandig om de achtereenvolgende nulpuntsbenaderingen x te laten afdrukken. Daardoor is beter te volgen wat er precies gebeurt, en hoe snel het nulpunt wordt benaderd. Het afdrukken van de x gaat het eenvoudigst, door bij iedere funktieaanroep eerst de x-waarde te laten afdrukken:

```
900:PRINT "X=";X:F=f(X)
910:RETURN
```

N.4.7 Leerstof 4.7.1 bestuderen en voorbeeld 1 doornemen.
Maken opgaven: 1, 2, 3 en 4.
Leerstof 4.7.2 doorlezen en daarbij vooral letten op de rij absolute fouten onderaan blz 323. Merk op, dat de fout bij iedere stap kwadratisch kleiner wordt. Voorbeeld overslaan, maar de Voorbeelden 1 en 2 doornemen. Bestudeer de figuren 4.39 en 4.40 om te zien hoe deze methode geen resultaat oplevert.
Leerstof 4.2 Raaklijnmethode doornemen; het programma invoeren.
Maken opgaven: 7, 8, 9 en 10.

4.2 Raaklijnmethode

```
PROCEDURE nulpuntRaaklijn (f(x),fa(x),nul,x;x,fout);

{ Bepaalt de oplossing x van de vgl f(x)=0, met een nauwkeurigheid nul in f.}
{ De volgende testwaarde xv wordt bepaald door het snijpunt van de x-as en  }
{ de raaklijn in het laatste testpunt aan de grafiek van de funktie. De rc  }
{ van de raaklijn is gelijk aan de afgeleide in het raakpunt. De geschatte  }
{ fout is onzeker als bij het nulpunt f'=0.                                  }

  VAR x : testpunt x-waarde;
      f : testpunt f-waarde;          { getest op f=f(x) voldoende nul      }
      n : nummerindex testpunten;     { n=0,1,2,...                         }
     fa : afgeleidewaarde in testpunt;{ fa=fa(x) mag niet 0 zijn, wel klein }
     xv : volgende_punt x-waarde;     { xv=x-f/fa , bepaald door raaklijn   }
     dx : verandering in x;           { dx=xv-x=-f/fa                       }
     dxo: oude dx;                    { dxo=x-xo                            }
      r : rede van de dx rij;         { r=dx/dxo ; meestal is |r|<<1        }
    nul: nauwkeurigheid in f;
   fout: nauwkeurigheid in x;         { fout=sign(r)r²dx mits f'<>0 is      }

  FUNCTION  f= f(x);
  FUNCTION  fa=fa(x);                 { fa=f'}
```




```
BEGIN
  {Begin_x}
  n:=0;read(xv);
  dx:=10;r:=1;                      { willekeurig gekozen, zodat fout=10   }
  Volgende_punt(xv;x,f);
  WHILE NOT abs(f)<=nul DO
    BEGIN
      {Volgende_x}
      fa:=fa(x);
      xv:=x-f/fa;                    { xv is snijpunt raaklijn met x-as    }
      dxo=dx; dx:=xv-x;  r:=dx/dxo; { let op de volgorde van gegevenopslag}
      Volgende_punt(xv;x,f)
    END;
  {Foutschatting}
  fout:=r²dx; write(fout);          { foutschatting is kwadratisch in r   }
END.
```

De vertaling van PASCAL naar BASIC voor nulpuntraaklijn volgt hieronder.

De afgeleide funktie wordt 'direkt in de buurt' van de funktie gedefinieerd. Daardoor staan de bij elkaar behorende funkties F (f) en FA (f') direkt bij elkaar.

```
PROCEDURE nulpuntRaaklijn (f(x),fa(x),nul,x;x,fout);
  FUNCTION fa(x);                              |  920:FA=f'(X)
                                               |  930:RETURN
BEGIN                                          |
  {Begin_x}                                    |
  read(xv);                                    |  200:INPUT "XV=";XV
  dx=10;r:=1;                                  |  210:DX=10:R=1
  n:=1; Volgende_punt(xv;x,f);                 |  220:N=1:GOSUB 500
  WHILE NOT abs(f)<=nul DO                     |  230:IF ABS(F)<= NUL THEN GOTO 280
    BEGIN                                      |
     {Volgende_x}                              |
     fa:=fa(x);                                |  240:GOSUB 920
     xv:=x-f/fa;                               |  250:XV=X-F/FA
     dxo=dx; dx:=xv-x;  r:=dx/dxo;             |  255:DO=DX:DX=XV-X:R=DX/DO
     n:=n+1; Volgende_punt(xv;x,f)             |  260:N=N+1:GOSUB 500
    END;                                       |  270:GOTO 230
  {Foutschatting}                              |
  fout:=r^2 dx;                                |  280:FT=SQU(X)*DX
END.                                           |
```

Desgewenst kan de geschatte fout worden gebruikt voor een **allerbeste** benadering van het nulpunt:

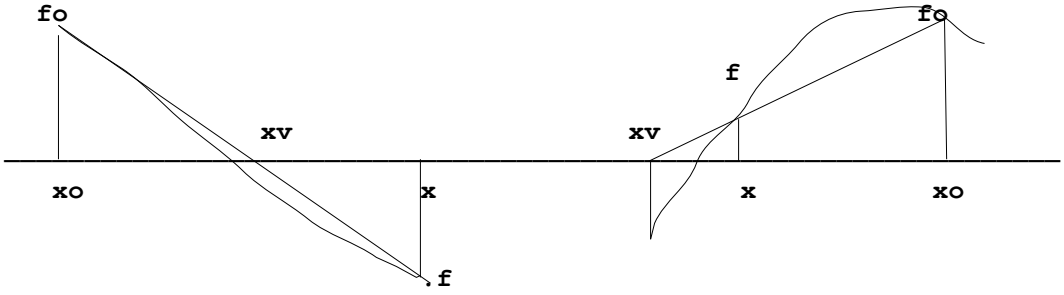
```
allerbeste benadering = benadering + foutschatting
            nulpunt   =     x      + fout
```

De foutschatting voor die allerbeste benadering is helaas gelijk aan de foutschatting voor de benadering, maar dan als absolute waarde (het echte nulpunt kan aan beide kanten liggen). Kontroleer aan de hand van de funktiewaarde, en het verloop van de funktie, of de richting van de foutschatting (+ of -) korrekt is.

N.4.8 Leerstof 4.8.1 bestuderen, maar alleen de echte koordenmethode met steeds opschuivende punten x0 en x1. Laat de OPMERKING aan het eind goed tot je doordringen.
Leerstof 4.8.2 doorlezen.
Maken opgaven: 1 en 2.
Leerstof 4.3 Koordenmethode doornemen en programma invoeren.
Maken opgaven: 3 en 5.

4.3 Koordenmethode

```
PROCEDURE nulpuntKoorde (f(x),nul,x,xv;x,fout);

  {   Geeft de oplossing x van de vgl f(x)=0, met een nauwkeurigheid       }
  {   nul in f volgens de koorde methode. Schatting fout alleen korrekt    }
  {   als in het nulpunt niet f'=0.                                        }

  VAR x  : testpunt x-waarde;
       f  : testpunt f-waarde;          { f=f(x) getest op voldoende nul zijn}
       n  : nummerindex testpunten;     { n=0,1,2,...                         }
       xo : oude testpunt;              { nodig voor het bepalen van dekoorde}
       fo : funktiewaarde oude testpunt;{ fo=f(xo);                           }
       fa : richtingscoefficient koorde;{ fa=(f-fo)/(x-xo), mag niet 0 zijn  }
xv : volgendtestpunt;             { uit snijpunt koorde met xas          }
       dx : verandering x;              { dx=xv-x=-f/fa wordt zeer klein      }
dxo: oude verandering in dx;      { dxo=x-xo                             }
       r  : rede van de dx rij;         { r=dx/dxo, in buurt nulpunt <<1      }
       nul: nauwkeurigheid in f;
      fout: fout in benadering;         { schatting fout=sign(r)r^1,6dx        }

  FUNCTION  f=f(x);
```




```
 BEGIN
   {Begin_x}
    read(x);f:=f(x);                { extra punt om koorde te bepalen     }
    xo:=x;fo:=f;                    { extra punt wordt bewaard als (xo,fo)}
    read(xv);
    r:=1; dx:=10;                   { willekeurig gekozen zodat fout=10   }
    n:=1; Volgende_punt(xv;x,f);
    WHILE NOT  abs(f)<=nul DO
      BEGIN
       {Volgende_x}
       fa:=(f-fo)/(x-xo);
       xv:=x-f/fa;                  { snijpunt koorde met x-as            }
       dxo:=dx; dx:=xv-x; r:=dx/dxo; { nodig voor foutbepaling            }
       xo:=x; fo:=f;                { extra nodig om koorde te bepalen    }
       n:=n+1; Volgende_punt(xv;x,f);
      END;
    {Foutschatting}
    fout:=r^1,6dx; write(fout);     { fout is bijna kwadratisch in r      }
 END.
```

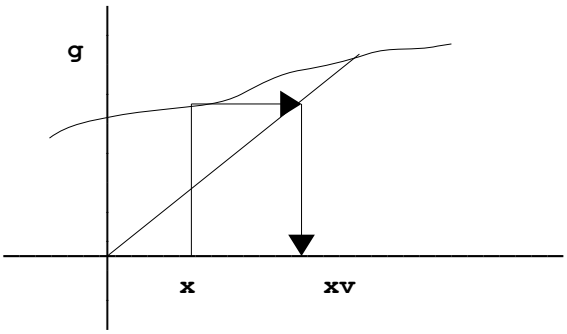
Vertaling van het programma van de koordenmethode.

```
PROCEDURE nulpuntKoorde                    |
                                           |
BEGIN                                      |
  read(x); f:= f(x)                        |  300:INPUT "XO=";X:GOSUB 900
xo:=x;fo:=f;                               |  305:XO=X:FO=F
  read(xv);                                |  310:INPUT "XV=";XV
  r:=1; dx:=10;                            |  315:R=1:DX=10: SO=SGN(R)
  n:=1; Volgende_x(x,f;xv);                |  320:N=1:GOSUB 500
  WHILE NOT abs(f)<=nul DO                 |  330:IF ABS(F)<=NUL THEN GOTO 380
    BEGIN                                  |
    fa:=(f-fo)/(x-xo);                     |  335:FA=(F-FO)/(X-XO)
    xv:=x-f/fa;                            |  340:XV=X-F/FA
    dxo:=dx; dx:=xv-x; so:=sign(r)         |  345:DO=DX:DX=XV-X:SO=SGN(R)
    xo:=x; fo:=f;                          |  350:XO=X:FO=F:R=DX/DO
    n:=n+1; Volgende_x(x,f;xv);            |  360:N=N+1:GOSUB 500
    END;                                   |  370:GOTO 330
  fout:=so*sign(r)*r^1,6*dx;               |  380:FT=SO*SGN(R)*ABS(R)^(1.6)*DX
END;                                       |  390:RETURN
```

Gebruik de fout voor een **allerbeste** benadering. Kontroleer of het teken van de fout korrekt is, door de achtereenvolgende punten te tekenen.

4.4 Substitutiemethode

```
PROCEDURE nulpuntSubstitutie (f(x),g(x),nul;x,fout);

  {  Geeft de oplossing x van de vgl f(x)=0 met de nauwkeurigheid nul in  }
  {  f(x) met behulp van de substitutiefunktie g(x): f(x)=0 als g(x)=x.   }
  {  In de omgeving van de oplossing x zal voor goede convergentie moeten }
  {  gelden |g'|<1; zo niet, dan de inverse funktie g^-1(x) nemen.        }

  VAR x  : testpunt x-waarde;
      f  : testpunt f-waarde;          { getest op f(x) voldoende nul       }
      n  : nummerindex testpunten;     { n=0,1,2,...                        }
      xv : volgende testwaarde;        { xv=g(x)=gvolgens substitutiemethode}
      nul: nauwkeurigheid in f-x;
      dx : verandering in x;           { dx=xv-x=g(x)-x=g(x)-g(xo)          }
      dxo: oude dx;                    { dxo=x-xo                           }
      r  : rede van de dx rij;         { r=dx/dxo=(g(x)-g(xo))/(x-xo)=g'    }
     fout: nauwkeurigheid in x;        { schatting fout=rdx/(1-r) als |r|<1 }

  FUNCTION  f=f(x);                  { funktie waarvan nulpunt wordt bepaald }
  FUNCTION  g=g(x);                  { substitutiefunktie om x te vinden     }
```




```
BEGIN
  {Begin_x}
  read(xv);
  r:=1; dx:=10;                    { willekeurig zodat fout oneindig is  }
  n:=1; Volgende_punt(xv;x,f);
  WHILE NOT  abs(f)<=nul DO
  BEGIN
    {Volgende_x}
    xv:=g(x);
    dxo:=dx; dx:=xv-x; r:=dx/dx0;
    IF abs(r)>1 THEN write('divergent');
    n:=n+1; Volgende_punt(xv;x,f)
  END;
  {Foutschatting}
  fout:=rdx/(1-r)                  { foutschatting bijna lineair in r   }
END;
```

De vertaling van de substitutiemethode naar BASIC.
De substitutiefunktie g(x) zetten we weer vlak bij de funktie f(x) en de afgeleide funktie f'(x)) op regelnummer 940:

```
PROCEDURE nulpuntSubstitutie
FUNCTION g(x);                          |  940:G=g(X)
                                        |  950:RETURN
BEGIN                                   |
 read(xv);                              |  400:INPUT "XV=";XV
 r:=1; dx:=10;                          |  410:R=1:DX=10
 n:=1; Volgende_x(x,f;xv);              |  420:N=1:GOSUB 500
 WHILE NOT abs(f)<=nul                  |  430:IF ABS(F)<=NUL THEN GOTO 480
 BEGIN                                  |
   xv:=g(x);                            |  440:GOSUB 940:XV=G
   dxo:=dx; dx:=xv-x; r:=dx/dx0;        |  445:DO=DX:DX=XV-X:R=DX/DO
   IF abs(r)>=1 THEN write('div.')      |  450:IF ABS(R)>=1 THEN PRINT "DIVERGENT"
n:=n+1; Volgende_x(x,f;xv);             |  460:N=N+1:GOSUB 500
 END;                                   |  470:GOTO 430
 fout:=rdx/(1-r)                        |  480:FT=R*DX/(1-R)
 END;                                   |  490:RETURN
```

Gebruik de fout voor een **allerbeste** benadering.

Merk op, dat na een waarschuwing "divergent' het programma gewoon doorgaat. Er is immers altijd wel een of ander nulpunt te vinden, ook al is dat ergens anders dan oorspronkelijk was vermoed.

4.6. Foutschatting

4.6.1 Fout bij afbreken rij

Bij alle nulpuntsmethoden verkrijgen we een rij x-waarden die naar het nulpunt convergeren, als de methode tenminste werkt. We zullen de fout, die we maken als we de rij onderbreken, hier nader bezien. Eerst zullen we de achtereenvolgende **benaderingen** nummeren:

$x_1, x_2, x_3, x_4,.., x_n, x_{n+1}, x_{n+2}, \ldots x$

We nemen aan dat het **nulpunt** x is. Als we afbreken bij een x_n, dan maken we een **fout** E_n, in de bepaling van het nulpunt, ter groote:

$E_n = x - x_n$

De waarde x is niet bekend, dus moeten we proberen met behulp van de laatste x-waarden deze fout af te schatten. Daarvoor gebruiken we de **toename** dx_n van de x-waarden:

$dx_n = x_{n+1} - x_n$

Merk op, dat we die toename kunnen uitdrukken in de (onbekende) fouten:

$dx_n = E_n - E_{n+1}$

Nemen we aan, dat de fouten steeds kleiner worden (ze moeten immers naar nul), dan kunnen we de grootste faktor E_n eruit halen:

$dx_n = E_n (1- E_{n+1}/E_n)$

Kennelijk is voor de `snelheid' waarmee de fout kleiner wordt de verhouding tussen de fout en de vorige fout, de **foutverhouding** q, van belang:

$q_{n+1} = E_{n+1}/E_n$

$dx_n = E_n (1- q_{n+1})$

Als we de rij stoppen met term n, dan willen we de fout E_n uitdrukken in de `bekende' getallen dx_{n-1}, dx_{n-2}, en lager. Met $E_n = q_n E_{n-1}$, wordt dan

$$E_n = q_n dx_{n-1}/(1- q_n)$$

Welke getallen kunnen we verwachten voor de verhoudingen? Het ligt voor de hand dat er een nauw verband zal zijn tussen de verhouding van de achtereenvolgende fouten q, en de verhouding tussen de toename en de vorige toename, de **toenameverhouding** r:

$r_n = dx_n/dx_{n-1}$

$= E_n (1- q_{n+1})/E_{n-1} (1- q_n)$, dus

$r_n = q_n (1- q_{n+1})/(1- q_n)$

Helaas zijn r-waarden slechts bekend tot r_{n-1}, zodat we nog moeten weten hoe de voor de fout nodige q_n is om te zetten in de voorafgaande waarde q_{n-1}.

Hoe de verhoudingen q zich precies gedragen hangt af van de methode die we gebruiken. We onderkennen twee gevallen:

1. lineair als q_n ---> q , mits abs(q) < 1
2. snel als q_n ---> 0

In het `lineaire' geval is de fout snel te vinden:

1. lineair $E_n = q\, dx_{n-1}/(1-q)$ met $q = r_{n-1}$

Dat q niet te groot kan zijn is duidelijk: de fouten moeten kleiner worden.

De methode heet lineair, omdat de foutverhouding een eerste macht van de vorige is:

1. lineair $q_{n+1} = q_n^1$

In het `snelle' geval moeten we er rekening mee houden, dat vanwege de snelle convergentie de q-waarden zeker niet gelijk zijn. Wel is de volgende q verwaarloosbaar ten opzichte van de vorige (of tov 1):

2. snel $E_n = q_n \, dx_{n-1}$ met $q_n = r_n$

We zullen laten zien, dat de raaklijn- en koorde- methode de foutverhouding een macht groter dan 1 van de vorige is (dichtbij het nulpunt):

2r. raaklijn $q_{n+1} = q_n^2$

2k. koorde $q_{n+1} = q_n^{1,6}$

Bij de koorde methode moeten we nog apart rekening houden met de tekens van de toenamen, in tegenstelling tot de raaklijn methode waarbij de x-waarden uiteindelijk van een kant komen, dus gelijk teken hebben.

4.6.2. Lineaire methode (substitutie)

Bij de substitutie methode wordt de volgende x-waarde bepaald door de vorige met behulp van de **substitutiefunktie g(x)**:

$x_{n+1} = g(x_n)$

Kennelijk convergeert de rij naar x, zodat

$x = g(x)$

De fout die we maken bij afbreken bij de term n+1 is:

$E_{n+1} = x - x_{n+1} = g(x) - g(x_n)$

Door **lineaire interpolatie** rond x blijkt de $_{fout}$ te worden bepaald door de afgeleide funktie in een punt t tussen x en x_n:

$E_{n+1} = (x - x_n)\, g'(t)$

$E_{n+1} = E_n \, g'(t)$

Daaruit volgt direkt voor de foutverhouding een `konstante' waarde:

$q_{n+1} = g'(t)$

Natuurlijk moeten we voldoende dicht bij x zijn om t=x te kunnen nemen.
Omdat de convergentie pas bij abs(q)<1 is, moet de substitutiefunktie g in de buurt van het nulpunt niet te steil lopen: abs(g'(x)) < 1. Numeriek is g' te bepalen met q, dus r:

$g'(t) = r_n$

4.6.3.a Raaklijn methode

Bij de raaklijn methode van Newton wordt het volgende punt gevonden door de raaklijn te tekenen en het snijpunt met de x-as te nemen als volgende benadering voor het nulpunt. De vergelijking van de raaklijn, in een tussenpunt t uitgedrukt ipv de gebruikelijke variabele x, is:

$p(t) = f(x_n) + (t - x_n)\, f'(x_n)$

Dus geldt voor x_{n+1}:

$0 = f(x_n) + (x_{n+1} - x_n)\, f'(x_n)$

$x_{n+1} = x_n - f(x_n)/f'(x_n)$

Daarmee is de substitutiefunktie g gevonden:

$g(_t) = _t - f(_t)/f'(_t)$

In dit geval is echter niet alleen g(x) = x, maar ook g'(x) = 0, zoals door differentieren en invullen is te zien:

$g'(t) = 1 - f'(t)/f'(t) + f(t)f''(t)/(f'(t))^2 = f(t)f''(t)/(f'(t))^2$

$g'(x) = 0$

$g''(t) = f'(t)f''(t)/(f'(t))^2 + f(t)(...)$

$g''(x) = f''(x)/f'(x)$

Omdat g'(x)=0 is, moeten we de funktie $g(x_n)$, bij het bepalen van de fout, **kwadratisch interpoleren** rond x:

$E_{n+1} = g(x) - g(x_n)$

$E_{n+1} = -(x_n - x)^2 g''(t)/2$

$E_{n+1} = -E_n^2 g''(t)/2$

Deze vergelijking voor de fouten kan via twee overwegingen worden omgewerkt naar een vergelijking voor de verhoudingen q. Deel enerzijds de vgl door E_n, en er volgt:

$q_{n+1} = -E_n g''(t)/2$

Vermenigvuldig anderzijds de vgl met de faktor -g''(t)/2:

$-E_{n+1} g''(t)/2 = (-E_n g''(t)/2)^2$

Gekombineerd vinden we de kwadratische foutafname:

$q_{n+1} = q_n^2$

Inderdaad blijkt de raaklijn methode tot een kwadratische afname van de verhoudingen q te leiden, en daardoor tot zeer snelle convergentie van de reeks x-waarden naar het nulpunt x. De faktor -g''/2 is te vinden uit de verhouding q_{n+1}/E_n, welke numeriek kan worden omgezet naar r en dx:

$-g''(t)/2 = r_n/dx_{n-1} = dx_n/dx_{n-1}^2$

4.6.3.b Koorde methode

Feitelijk is de koorde methode niet veel verschillend van de raaklijn methode, omdat we de raaklijn vervangen door een koorde van punten die dichtbij liggen. We zullen zien dat dat toch konsekwenties heeft voor de snelheid van de convergentie. Daar staat echter tegenover, dat we geen afgeleide funktie nodig hebben, en dus niet hoeven te berekenen (we benutten steeds reeds berekende punten van de funktie voor de koorde). Alles bijeen genomen weegt het verminderde rekenwerk op tegen het snelheidsverlies van de convergentie: de koorde methode is nummer 1.

De vergelijking van de koorde door de punten x_n en x_{n+1}, uitgedrukt in een willekeurig tussenpunt t is:

$p(t) = f(x_{n+1}) + (t - x_{n+1}) f'(t_n)$

met $f'(t_n) = (f(x_{n+1}) - f(x_n))/(x_{n+1} - x_n)$

Het snijpunt met de x-as geeft x_{n+2}:

$0 = f(x_{n+1}) + (x_{n+2} - x_{n+1}) f'(t_n)$

Omdat twee punten een rol spelen gaan we nu iets anders te werk dan bij de raaklijn. We interpoleren de funktie f(x) door de punten $x_{n+1 \text{ en } x}n$, met de variabele x. Deze kwadratische interpolatie geeft voor f(x)=0:

$0 = f(x_{n+1}) + (x - x_{n+1}) f'(t_n) + (x - x_{n+1})(x - x_n) f''(t)/2$

De fout vinden we door de vergelijking voor de koorde hier af te trekken:

$0 = (x - x_{n+2}) f'(t_n) + (x - x_{n+1})(x - x_n) f''(t)/2$

$_0 = E_{n+2} f'(tn) + E_{n+1} E_n f''(t)/2$

$E_{n+2} = -E_{n+1} E_n f''(t)/2 f'(t_n)$

Als we aannemen dat de punten dicht bij x liggen, dan kunnen we, als tevoren, redelijkerwijs de substitutiefunktie g gebruiken:

$E_{n+2} = -E_{n+1} E_n g''(t)/2$

$g''(t) = f''(t)/f'(t_n)$

Ook deze vergelijking kan eenvoudig worden omgewerkt naar een vergelijking voor de verhoudingen q. Eerst delen we door de n+1-ste fout:

$q_{n+2} = -E_n g''(t)/2$

Dan vermenigvuldigen we met de faktor -g''(t)/2:

$-E_{n+2} g''(t)/2 = (-E_{n+1} g''(t)/2)(-E_n g''(t)/2)$

$q_{n+4} = q_{n+3} q_{n+2}$

Evengoed geldt dan natuurlijk voor de foutverhoudingen:

$$q_{n+2} = q_{n+1} q_n$$

Helaas is deze vergelijking iets ingewikkelder dan bij de raaklijn methode: er zijn nu twee verhoudingen in het rechterlid. Daaruit volgt dat de foutverhouding een macht μ is van de vorige:

$$q_{n+1} = q_n^{\mu}$$

Door invullen blijkt wat μ moet zijn; allereerst is

$q_{n+2} = q_{n+1}^{\mu} = (q_n^{\mu})^{\mu} = q_n^{\mu.\mu}$

en $q_{n+1} q_n = q_n^{\mu} q_n = q_n^{\mu+1}$

$\mu^2 = \mu + 1$

met de oplossing voor de macht:

$$\mu = (1+\sqrt{5})/2 = 1{,}6$$

Tenslotte merken we nog op, dat de tekenwaarde van de fout hier niet kon worden meegenomen; die wordt bepaald door de oorspronkelijke vergelijking die uitdrukt dat de foutverhouding het produkt is van de vorige twee. Daaruit konkluderen we, dat of alle tekens + blijven, of, als er tekenwisseling optreedt, deze om de drie is: . . -+- -+- -+ . ..

C.M.Fortuin, fortuinc@xs4all.nl