Vikingen FutureLook

by

Delphi Finansanalys AB

Översikt

Futurelook är ett unikt och mycket kraftfult verktyg för finansanalytiker och kapitalplacerare. Med FutureLook är det möjligt att på ett enkelt sätt göra beräkningar och testa hypoteser. FutureLook är integrerat med Vikingen och sitter ihop med Vinstgeneratorn vilket gör det enkelt att vinsttesta modeller och hypoteser.

FutureLook är uppbyggt som ett enkelt programmeringsspråk. Det är anpassat och förenklat för att lösa de problem man ställs inför då man vill analysera finansiella tidsserier.

Fördelar med futureLook:

+

+

+

Vad är FutureLook

Till det yttre består FutureLook av några filer som lagras i Vikingen. De är Modeller respektive Presentationer, eller Presentationsmodeller man även säger. När du i Vikingen öppnar "analytikern" och väljer Modeller och anger att köra modellen Kursdiagram, så pekar du faktiskt på en presentationsmodell.

Du kan välja RSI-diagram, Bollingerband eller Multimodell eller någon av alla de andra modellerna beroende på vad du vill göra. Analysera, hitta köpkandidater eller säljkandidater. Beräkna signallistor eller nyckeltal. Allt detta görs genom att peka på rätt Presentationsmodell. Presentationsmodellens namn anger för användaren tex vad modellen gör eller vem som har skrivit modellen. Presentationsmodellen kan därför sägas vara användarens handtag mellan sig själv och FutureLook.

Utöver sin funktion som "interface" innehåller Presentationsmodellen även uppgifter om hur diagrammet eller tabellen ser ut samt vilka inställningar användaren kan göra som tex ändra tidsintervall, färger eller byta till logaritmisk skala.

En Presentationsmodell "hämtar sin data" från en modell. Denna modell hanterar hämtning av data och beräkning.

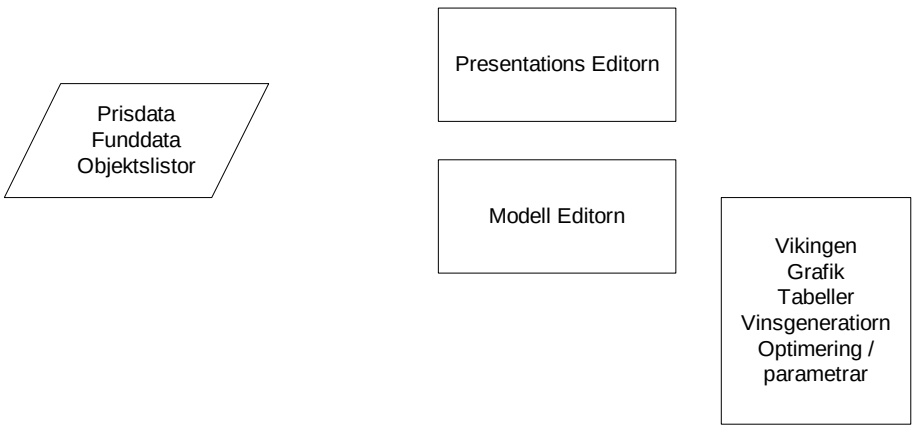
Ett exempel på en enkel modell kan vara att modellen hämtar ett värdepappers sista kurs för alla dagar och lämnar dessa som utdata.

Ofta har man samma namn på modellen och presentationen. Men om man vill kan man ha flera presentationer som hämtar sin data från en och samma modell. Detta är bra om man vill ha speciella utseenden på sina diagram eller tabeller.

Man kan tänka sig en modell som har 10 olika utdata. Man bygger 5 olika presentationsmodeller och i varje av dessa visas endast två utdata per modell. På detta sätt har användaren erhållit 5 unika modeller.

Man kan även tänka sig att en modell har 2 utdata. Man bygger 5 olika presentationsmodeller med olika färg, graftyper, tidsintervall etc. Varje modell belyser en analytisk aspekt av den beräknade utdatan.

Modellen kan vara mycket enkel eller mycket komplex. En enkel modell hämtar tex ett värdepappers börskurs för alla dagar, beräknar två medelvärden och lämnar kursen och två medelvärden som utdata. En komplex modell kan utföra lopar och villkor som baseras på kurs och fundamentaldata för flera aktier och index. Resultatet kan manipulera diagrammet och rita linjer med olika färg och sätta in olika texter beroende på vad resultatet av beräkningarna blir.



Dokumentation och manualer

Det finns en omfattande dokumentation att tillgå för att komma igång eller hitta hjälp om FutureLook. Under betatestfasen finns ingen onlinehjälp, men den kommer sedan. Den dokumentation som vi i detta skede publicerar finns samlade i denna fil. Nedan ser du förteckning över de olika delarna.

Vi är mycket tacksamma för alla typer av synpunkter och förslag till förbättringar.

Dokumentnamn	wordfil	version	Kommentar
FutureLook översikt	1 FutureLook.doc	beta990514	Detta dokument. Översikt över FutureLook och all dokumentation
Starthjälp	2 FL Starthjälp.doc	beta990514	Översiktligt dokument som beskriver hur man konstruerar modeller och använder dessa,
Användarmanual Presentationseditorn	4 FL Presentationskotroll.DOC	beta990514	Programmanual för presentationseditorn. Hur man öppnar, sparar och kör presentationer.
Användarmanual Funktionseditorn	3 FL Funktionskontroll.DOC	beta990514	Programmanual för funktionseditorn. Hur man öppnar, sparar och kör funktioner.
FutureLook Referensmanual	5 FL Referensmanual.doc	beta990514	Beskriver funktionerna som finns i std lib. Tolkningshjälp och en del modellexempel.
Standardfunktioner	6 FL Standardfunktioner.doc	beta990514	Beskriver ingående hur en modell är uppbyggd, vektorer, syntax mm.
Fundamental library, FD.			Skall beskriva specifika funktioner och saker att tänka på då man bygger fundamentalmodeller.
			? Bilaga Fundamental shortname
			? Bilaga Countrycodes
			? Bilaga Sectorcodes

EHModspråk 22,20,18,16

Version 981209

Version 990412

Modeller och hur de skapas med hjälp av FUTURE LOOK.

1 Vad skiljer en modellpresentation från en modell?

Vi börjar med att göra klart skillnaden mellan modell och modellpresentation.

Skillnaden mellan en modell och modellpresentation kan liknas vid skillnaden mellan en datorns hårddisk och dess bildskärm. För att du skall kunna få fram något på skärmen så måste det finnas ett program. Utan skärm så ser du ingenting av vad programmet gör.

Vikingens program består schematiskt av två delar, den ena svarar mot hårddisken, den andra mot bildskärmen.

Hårddiskdelen gör beräkningarna, bildskärmsdelen gör att något kommer fram på skärmen. Vi som gör Vikingen kallar beräkningsdelen för modellen (funktionen) och den andra delen av programmet för modellpresentationen. Modellen (funktionen) kan bestå av en eller flera delmodeller (funktioner) varav några är ”svarta” lådor, färdiga byggelement skapade av våra systemmän i t.ex. C++, andra sådana som du själv konstruerar i Future Look.

Modellen gör, när den körs, beräkningarna och modellpresentationen skapar vad du ser på bildskärmen, diagram, tabeller och text.

Genom att vi har skapat en boskillnad mellan modell och modellpresentation har vi öppnat möjligheterna att ha flera olika modellpresentationer till en och samma modell.

1 Allmänt om modeller

När vi i fortsättningen talar om modeller så använder vi modeller som beteckning på de funktioner, som används i ekonomiska sammanhang.

Modellerna är utgångspunkten vid skapandet av modellpresentationerna.

Varje modell, funktion, har ett unikt namn och är placerade i ett unikt bibliotek. Vill du anropa en viss funktion, t.ex. funktionen ”MAVN” som ligger i biblioteket ”Std” och ge den värdet ”Medelvärde” så skriver du ”Medelvärde := std.MAVN(main.close, medelvärdeslängd);”

Symbolen ”:=” betyder tilldelning. Funktionen Medelvärde ovan betyder t.ex. skapa en tidsserie som innehåller ett medelvärdeslängd långt medelvärde för slutkursen.

Du måste då veta att MAVN kräver två argument, det ena main.close måste vara en tidsserie, t.ex. av slutkurser, och den andra ett heltal, längden på medelvärdet.

Om du skulle vilja köra funktionen Medelvärde, så kommer den om du inte begär annat att köras på det aktuella objektet eller den aktuella listan, om du inte väljer annat.

Vi kommer i fortsättningen att använda modell som beteckning även på modellpresentation när risken för förväxling är låg.

Vi delar in modellpresentationerna i analys - och signalmodeller med avseende på vad de används till.

Analysmodeller är modeller avsedda att resultera i signalmodeller.

Genom att använda analysmodeller kan vi få en uppfattning om lämpligheten att använda modellen som en signalmodell.

För att avgöra om en analysmodell skall kunna användas som signalmodell så måste den innehålla möjligheter att avgöra om de avkastningskrav mätt som resultat och risk, som du fordrar, uppfyllts av modellen ifråga, när den testas på historiska tidsperioder.

När du gör testningarna är det viktigt att testningarna görs på i förhållande till eventuella parameteroptimeringar senare perioder än den för optimeringen.

Först av allt, kom ihåg att allt som datorn skall utföra, det måste den också få klara instruktioner om hur det skall göras. De flesta instruktioner, som behövs för att en modellpresentation skall kunna köras är redan inbyggda i Vikingen programmen.

De specifika instruktioner som måste finnas för att det skall vara möjligt köra en modellpresentation är;

de indata, parametrar, och de övriga variabler, som skall gå i modellen samt de beräkningar som skall utföras på dessa parametrar, d.v.s. själva modellen

de delar av modellen, utdata, som skall kunna ingå i modellpresentationen

och slutligen de objekt, den tidsperiod och den periodlängd som presentationen skall köras på.

De instruktioner som räknats upp under a) och b) ovan ingår i modellen och för instruktionerna under c) finns i Vikingen av Delphi satta standardvärden som sätts in när du kör en modellpresentation, om du inte lämnar andra instruktioner i samband med körningen. Det är viktigt att du ändrar dessa värden till rimliga värden i samband med att du göra presentationerna för modellen.

De ”generella” Indata, som modellen behöver, t.ex. kurser och volymer, och de ”generella” Utdata, som modellen skall generera, anger du, när du skapar modellen.

I modellpresentationen väljer du vilka av de tillgängliga Utdata, som skall ingå i presentationen av modellen, d.v.s. i graferna och i tabellen.

Ändringar av tidsperiod, som du vill köra modellpresentationen, och ändringar av t.ex. längden på medelvärden etc kan du göra i samband med körningen i modellinställningar.

2 Hur konstrueras modeller?

Vi skall här gå igenom hur du konstruerar en modell.

Ett första råd till dig är att du innan du skriver helt egna modeller först kopierar in en redan gjord modell och genom att ändra i den genom en trial and error process successivt lär dig hur modellspråket fungerar.

När du har tänkt till och har en verbal formulering av hur din modell skall se ut, så kan det vara lämpligt att börja med att dels ange vilka indata, som skall ingå i en modell, dels ange, (deklarera, ge datorn möjlighet identifiera) de storheter, som du skall använda dig av inne i modellen. Storheter som kan vara integers, t.ex. längden på ett medelvärde, eller vektorer, tidsserier, t.ex. en vektor (tidsserie) med medelvärden.

Du måste ange vad de består av och hur de mäts. Vi talar om datatyper. I Vikingen har vi följande datatyper; instrument (kurser, volymer), time, set (listor), sträng (text), heltal, reella tal och logiska (boolska) värden (sann och falsk).

Instrument är detsamma som objekt och innehåller för aktier följande värden högsta, lägsta, första och sista kurs under perioden samt volym och efteranmäld volym när periodlängderna är dag eller mer.

Sträng, eng. string, är en följd av tecken, bokstäver, siffror och symboler. Strängar använder du, när du vill lägga in text i diagrammen.

Set är ett antal objekt, som är representerade av vikingkoder. Set använder du, när du vill göra modeller där det läggs villkor på flera objekt.

För tid, heltal, reella tal och logiska värden måste du ange datatypen, time, integer, integervector, real, realvector, boolean och booleanvector.

Som regel ingår kursdata som Indata. Indata kan vara på fyra olika format, intradaydata, dagsdata, veckodata eller månadsdata.

Du skall strax få ett exempel på en modell, men först några allmänna regler.

När du har skapat en modell och skall kompilera, spara undan den, så struntar kompilatorn i mellanslag, tabbar och radavslut så länge som de inte finns inne i reserverade ord eller identifierare (funktionsnamn).

Stora och små bokstäver betraktas som synonymer.

Identifierare får inte börja med en siffra.

Main.close[10] betyder det tionde elementet i vektorn main.close.

Du får här ett exempel på hur en färdig modell kan se ut.

```
(*Detta är en RSI modell som ger köpsignaler då RSI värdet dels just passerat en botten, dels
har ett värde som inte överstiger " ÖvreGränsFörKöp". Säljsignaler ges då RSI värdet juste
passerat en topp på en nivå inte under " NedreGränsFörSälj.*)

par (main : instrument;
PerioderRSI : integer;
ÖvreGränsFörKöp, NedreGränsFörSälj : real;
out RSI : realvector;
out Köpsignal, Säljsignal, Signal : integervector) : realvektor;
out FilledClose, R1: realvector;
Buy, Sell : booleanvector;

begin
        FilledClose := Fill(main.close);
        RSI := RSI(FilledClose,PerioderRSI);
        R1 := SHIFT(RSI,1);
        BUY := (BOTD(RSI,1) < 1.5) AND (R1 <= ÖvreGränsFörKöp);
        SELL := (TOPD(RSI,1) < 1.5) AND (R1 >= NedreGränsFörSälj);
        Köpsignal := ONE(BUY);
        Säljsignal := ONE(SELL);
        Signal :=       Köpsignal - Säljsignal;
        return Signal;
end;
```

Allra först kommer ett textstycke inneslutet mellan (* och *).

Första raden, som börjar med ”par” kallas funktionshuvud.

I ”par” finns inom parentes angivet, deklarerat, de parametrar, main m.fl., som ingår som indata och som används i modellen och dessutom de datatyper, instrument m.fl. som par ametrarna består av.

Datatyperna är, som redan nämnts, instrument, time, set, string, integer, real och boolean samt vektorna timevector, integervector, realvector och booleanvector.

Anger du datatypen till integer innebär det att parametern är ett heltal.

Anger du i stället realvector, integervector eller booleanvector innebär det att parametern är en vektor av tal respektive av True, False.

I Vikingen är som regel integervector, realvector och booleanvector tidsvektorer med en ordningsföljd grundad på tidsperioder som vecka, dag eller intraday perioder.

I modellen i exemplet ovan finns det efter : angivet den datatyp som de införda parametrarna har. Det är en nödvändig instruktion för att datorn skall kunna hantera parametrarna ifråga.

Du måste således vara noga med att i modellerna deklarera samtliga byggstenars parametrar.

Som du ser i exemplet ovan så föregås några tidsserier (realvector och booleanvector ovan) av ”out”.

Genom att i ”par” börja raderna med ”out” deklarerar du att du vill ha möjlighet att när du gör modellpresentationerna ange att de skall vara med i presentationen av modellen. De blir Utdata. Observera att den här möjligheten föreligger bara i ”par” och inte för de i ”var” deklarerade storheterna.

Genom att deklarera datatyperna integer, real, boolean och time i ”par” så får du möjlighet

att i modellen gå in och ge storheten ifråga ett konstant värde

i samband med att du skapar en modellpresentation till modellen ge storheten ifråga ett konstant värde.

i fallet b kan du välja att ta med storheten i modellinställningar och på så sätt ge användaren möjlighet att ändra värdet.

Om en storhet av typen integer eller real skall kunna optimeras måste den tas med under ”par” .

I modellpresentationerna av modellen kommer du sedan att ha möjlighet att välja ut de utparametrar, som du vill ha med i presentationerna av modellen.

För de parametrar under ”var” ovan som är deklarerade som real respektive integer kommer, som redan påpekats, det i samband med körningar vara möjligt att genom att gå in i modellinställnigar ändra värdena på dessa parametrar.

Biblioteket std innehåller ett antal förprogrammerade, hårdlödda, matematiska, statistiska och logiska funktioner samt ekonomiska funktioner (modeller).

Du kan använda samma namn , när du deklarerar namnet på tidsserier som skall användas i modellen, som de funktionsnamn (modellnamn), som ingår i biblioteket, std, om du iakttar nedanstående.

Om du skapat en egen funktion, t.ex. MAVN, och längre ner i den modell, där du skapat den, vill använda standardbibliotekets MAVN funktion, så måste du skriva std.MAVN.

Likaså, om du i ett biblitotek, t.ex. ”Mitt bibliotek”, skapat en funktion med samma namn som en funktion i standardbiblioteket, std, och därefter växlar till ett annat bibliotek och där skapar en modell där du vill använda din ”egen” funktion, så måste ange detta genom att före funktionsnamnet ange dess bibliotek.

Skulle du inte ange något bibliteksnamn före funktionen ifråga, så kommer programmet att använda sig av standardbibliotekets funktion.

Varje instruktion skall avslutas med ett semikolon, ”;”.

Ingenting hindrar att du skriver fler än en instruktion på samma rad, så länge som du avslutar varje instruktion med semikolon.

3 Hur konstrueras modeller? Ett exempel.

En modells struktur är följande.

Först en textdel som omgärdas av (* *). T.ex.

```
(*Modell RSI

version 981207

Detta är en RSI modell, som ger köpsignaler när en stigande  RSI kurva har ett RSI värde som
understiger UndreRSIgräns och säljsignaler när en fallande  RSI kurva har ett värde som
överstiger ÖvreRSIgräns.*)
```

En deklaration av indata och parametrar. T.ex.

```
par (Main : instrument;
RSIlängd ; integer;
ÖvreRSIgräns : real;
UndreRSIgräns : real;
out FilledClose, RSIvärde : realvector;
out BUY, SELL : booleanvector,
out Signal : integervector): integervector;
```

En deklaration av parametrar som inte bedöms ska ingå i någon modellpresentation av modellen. T.ex.

```
varRSIvärdeIGår : realvector;
b1, b2 : booleanvector;
```

Slutligen kärnan, ”själva” modellen. Den skall inledas med Begin och avslutas med end. Kom ihåg att varje instruktion skall avslutas med semikolon. Här ett exempel.

```
Begin

FilledClose : b1 :=          FILL(Main.Close); (*sätter in föregående  senastbetald kurs de
gånger som betalkurs saknas.*)
RSIvärde :=  RSI1(FilledClose, RSIlängd); (*i modellpresentationen ersätter du det dummy
värde som är angivet där med t.ex. 14.*)
RSIvärdeIGår :=          SHIFT(RSIvärde, 1); (*RSIvärdeIGår kommer att ha gårdagens
RSIvärden som sina.*)
b1 :=        RSIvärde < RSIvärdeIGår;
b2 :=        RSIvärde > RSIvärdeIGår;
b3 :=        RSIvärde < UndreRSIgräns;
b4 :=        RSIgärns > ÖvreRSIgräns;
BUY :=       FILTERBUY(b1 AND b3, b2 AND b4);
SELL :=      FILTERSELL(b1 AND b3, b2 AND b4);
Signal :=    ONE(BUY) - ONE(SELL);
return Signal; (*Om modellen döpts till  RSImodell så kommer RSIsignal ha värdet Signal.*)

end;
```

Än en gång, de parametrar som kan användas i presentationseditorn är de som du under ”par” har deklarerat som ”out” storheter.

Tidsserier och konstanter som du avser enbart använda för mellanberäkningar kan du deklarera under ”var”.

Under ”var” deklarerade tidsserier kan du inte utan vidare få med i en modellpresentations diagram eller tabell och under ”var” deklarerade konstanter kan du inte påverka i en modellpresentation .

Om du studerar exemplet ovan så ser du, att vi deklarerat t.ex. FilledClose under ”var”.

För att få med t.ex. FilledClose, här en temporär (lokal) storhet, som returparametrar (parametrar som kan ingå i en modellpresentation) måste du använda dig av instruktionen ”return”.

4 Vad händer när en modell körs?

För ordningens skull, det är en modell som körs men det är en modellpresentation, som du väljer från någon av kategorierna i Vikingen.

De indata, som ingår i modellen laddas in. Tidsserierna är indexerade från 0 och framåt i tiden.

Den första instruktionen körs. Avser den första instruktionen att skapa en vektor, så kommer den vektorn att skapas.

Därefter körs instruktion för instruktion.

Resultaten av instruktionerna kan användas som byggstenar i senare givna instruktioner.

När alla instruktioner är utförda presenteras resultaten av körningen i enlighet med de instruktioner, som modellpresentationen innehåller.

5 Beräkningsordningen för ett uttryck

När du skriver en modell, som innehåller operatorer (not, *, +, > m.fl.) så måste du veta i vilken ordning som operationerna utförs.

Så här är beräkningsordningen i Vikingens modellspråk.

Not

*, /, and

+, -, or

=, <, >, <=, >=

Risken för fel undviker du genom flitigt bruk av paranteser.

Operatorer med samma företräde (precedens) utförs i ett uttryck från vänster till höger. T.ex. 5 + 4 - 6 + 3 beräknas först 9 - 6 + 3 och därefter 3 + 3 och slutligen 6.

Skulle det förekomma något odefinierat tal (operand) i ett uttryck, så blir resultatet odefinierat.

Självklart ger försök att dividera med 0 att resultatet blir odefinierat. Du kan testa det genom att skriva undef(resultat).

Om minst en av operanderna är en vektor, så kommer operationerna att utföras element för element i vektorn (vektorerna). Antag t.ex. att vektorn 3, 4, 8 skall multipliceras med 4 så blir resultatet en ny vektor 12, 16, 32.

6 Ändra i en modell

Du kan ändra i de modeller, som ligger i biblioteken DelphiStandard och DelphiSignaler samt självklart i de modeller, som du själv skapat.

De specialmodeller, som ligger i övriga Delphi bibliotek innehåller information, som vi inte lämnar ut.

När du vill ändra i en av de ”tillåtna” modellerna, så gör du på följande sätt.

Du klickar först på ”visa”, därefter på ”funktionskontroll”, och väljer det bibliotek, som modellen ligger i.

Därefter markerar du den modell du vill ändra och därefter klickar du på ”Editera” efter att ha kontrollerat att rutan efter ”Editera”, (den med ”F/P”) är markerad. Alternativt kan du dubbelklicka på den modell, som du vill ändra på.

När du gjort dina ändringar i den gamla modellen så sparar du undan den ändrade modellen under ett nytt namn.

När du sedan gör en modellpresentation, så måste du tänka på att om du ändrat i modellens ”par” deklaration, så kan du som regel inte använda de gamla modellpresentationen, om det finns en sådan, utan du är tvungen att skapa en ny sådan.

Ett bra sätt att komma igång med modellskapandet är att ta en redan befintlig modell som utgångspunkt.

7 Skapa en modell

7.2 *Skapa bibliotek*

Du har möjligt att dela upp dina modeller genom att lägga dem i olika bibliotek med vissa undantag. De specialmodeller, som Delphi producerat läggs i ”fasta” bibliotek, som Delphi skapat.

Innan du börjar med att skapa en ny modell så skall du ha skapat det bibliotek, t.ex. biblioteket ”Mina egna”, där du vill ha den nya modellen lagrad.

Skapar ett nytt bilbliotek gör du genom att i funktionseditorn först klicka på ”ny” efter att först ha sett till att ”F/B” för ”Ny” inte är markerad och därefter ange namnet på det nya biblioteket.

7.3 *Första steget*

När du skall skapa en ny modell börjar du med att markera ”Visa”, väljer ”funktionseditor” och därefter väljer du det bibliotek, som den nya modellen skall ligga i.

Därefter väljer du ”ny modell” och ger modellen ett namn. I nästa steg klickar du på ”Editera” och nu är det färdigt att skapa modellen.

7.4 *Deklarera inparametrar och variabler*

Du börjar med funktionshuvudet. Du anger då först under par de parametrar och de variabler, som du vet skall ingå i modellen, och som inte har karaktären av . Du kan namnge dem med namn, som du själv väljer på lämpligt sätt. Du kan, om du vill, använda å, ä och ö i namnen.

Alla parametrar och variabler, som skall ingå i modellen måste vara deklarerade under ”par” eller ”var”.

Det objekt, som modellen skall grundas på, anges som obj : instrument.

Upptäcker du under skapandet av modellen att du behöver ytterligare parametrar eller variabler så måste du ange dem i ”par” respektive ”var” ovan.

De parametrar och variabler med undantag av de instrument, som skall ingår i modellen, skall om du vill ha tillgång till dem vid modellpresentationerna deklareras (anges) under ”par” och de måste dessutom föregås av ”var”. Du kommer strax här nedan få exempel på hur du skall göra.

Temporära (lokala) variabler, som du räknar med att inte redovisa i modellpresentationerna deklarerar du under ”var”.

Om du vill kunna dels ange ett värde t.ex. ett gränsvärde i Stochastics som en parameter och också kunna rita gränslinjen i diagram så gör du så här. Skapa under ”par” en real, t.ex. gräns, och en realvector, t.ex. gränslinje, och tilldela i modellkärnan realvektorn gränslinje värdet gräns, som i det här fallet är ett konstant värde.

Du skriver i modellkärnan: gränslinje := gräns; Du kan då i modellpresentationerna ha gräns i modellinställningarna och där välja värdet på gräns och i ett pane ha gränslinje som en rät linje med det valda värdet på gräns.

7.5 Modelluppbyggnaden

Mellan ”begin” och ”end” ger du de instruktioner som behövs för att i modellspråket operationalisera din ”verbala” modell, dina hypoteser.

Modellen innehåller i stort följande;

```
//text innehållande namn på modellen, modellbeskrivning etc.
par( );
var
begin

ett antal instruktioner där bl.a. ett antal vektorer tilldelas värden genom att till höger om ":="
ange hur dessa värden skall beräknas

end;
```

7.6 Funktionsbiblioteket

För att underlätta modellskapandet kan du använda dig av det funktionsbibliotek med en mängd färdiga funktioner, som du hittar i funktionseditorn under biblioteket Delphi. Utöver funktionsbiblioteket med Delphifunktioner så kan du skapa ett eller flera egna bibliotek där du kan lägga in funktioner och modeller, som du själv skapar.

Exempel på for loop

En ”for loop”, som skall addera ett tal, hp1, till en invektor i form av en tidsserie från och med invektorn nr 10 till och med invektorn nr 20 ser ut så här;

Vi börjar med att kontrollera att vi ligger i rätt bibliotek, t.ex. ett som vi döpt till funktioner. Sedan fortsätter vi på samma sätt som vid skapandet av en modell.

```
par (invektor: realvector, hp1: integer) : realvector;

var i : integer;

begin

        for i := 10 to 20 do
                invektor[i] = invektor[i] + hp1;
end;

        return invektor;
end;
```

För att kunna köra funktionen måste det finnas en invektor med minst 20 värden, som kan laddas in som indata. Vidare förutsätts det att hp1 har ett värde, ett heltal. Värdet på alla indata av typen real och integer är som standard satta till 6.

I samband med skapandet av en modellpresentation kan du ändra det värdet och om du i exemplet ovan hade deklarerat hp1 som en variabel så hade du kunnat få med hp1 i modellinställningar. Då hade funktionshuvudet haft följande utseende.

```
par (invektor: realvector, var hp1 : integer) : realvector;
```

7.7 *Skapa en egen funktion*

Låt oss fortsätta med ett exempel där vi skriver en funktion som skapar ett medelvärde. Vi kallar funktionen för MAV. MAV = Moving aveage.

```
Par (invektor : realvector, mavlength : integer ) : realvector;

var i,j : integer; summa : real; utvektor : realvector;

begin
        for i := mavlength - 1 to len(invektor) - 1 do
                summa := 0;
                for j := i - mavlength + 1 to i do
                        summa := summa + invektor[j];
        end;
                utvektor[i] := summa/mavlength;
        end;
        return utvektor;
end;
```

Vad händer, när du kör funktionen?

Antag att vi i en modellpresentation givit mavlength värdet 10 och att invektorn består av 20 element indexerade från 0 till 19.

Det första värdet för styrvariabeln i, blir 9. Summa sätts till noll.

Därefter sker följande.

Med början på värdet 0 för styrvariablen j, 10 - 9 + 1, sätts summa till värdet av invector[0], därefter sätts för styrvariabelvärdet 1 summa till värdet av invector[0] och invector[1] o.s.v. fram t.o.m. styrvariabelvärdet 9 då summavärdet är summan av invektorns första 10 element.

Därefter beräknas värdet på utvektor[9] som summavärdet av invektorns första 10 element, Summa dividerat med 10, mavlength.

Nu sätts styrvariablen i till värdet 10, varefter summa sätts till noll.

Med början på värdet 1 för styrvariablen j, 11 - 9 + 1, sätts summa till värdet av invector[1], därefter sätts för styrvariabelvärdet 2 summa till värdet av invector[1] och invector[2] o.s.v. fram t.o.m. styrvariabelvärdet 10 då summavärdet är summan av invektorns element 1 t.o.m 11.

Utvektor[10]:s värde blir nu summavärdet av invektorns element 1 t.o.m 11 dividerat med 10.

Sedan börjas en ny omgång med styrvariabeln i satt till 11. Detta kommer att fortsätta t.o.m att i fått värdet 19. (Vi antog att det fanns 20 element i invektorn.)

Om du fått klart för dig hur loopar fungerar, så kan du hoppas över följande liknelse och gå vidare till nästa avsnitt.

På en ”gammeldags” klocka finns ofta både en timvisare, en minutvisare och en sekundvisare.

En enkel loop kan då uppfattas som en instruktion att för varje timme från 0 till 12 gå ett steg framåt.

En dubbel loop, som i exemplet ovan om medelvärde, kan då uppfattas som en instruktion till klockan att för varje minut, från noll till 60 gå ett steg framåt och att göra det för varje timme från 0 till 12.

Om du nu tar en titt på mav modellen ovan, så har du där en instruktion ”summa := 0”. Motsvarigheten i vårt klockexempel blir att för varje ny timme sätta antalet minuter till 0.

7.8 Anropa en egen funktion

Du kan nu använda MAV funktionen ovan för att t.ex. skapa ett MAV på ett MAV.

```
Par (obj : instrument) : realvector;

var

begin

        return MAV(MAV(obj.close, 5),7);
end;
```

Om det skulle finnas en Delphi funktion med namnet MAVN så kommer den funktionen, om du skapar en ny modell i samma bibliotek och i den anropar MAVN inte att anropas utan i stället den av dig skapade.

Vill du att Delphifunktion MAVN skall anropas, så skriver du std.MAVN, eftersom Delphis standardfunktioner ligger i biblioteket std.

7.8.1 Åtkomst av invektorer för ett objekt och för en ordning

Kurser och volymer för aktierna (instrumenten) i din databas kommer du åt genom att efter objekt ange kurs eller volym, t.ex. objekt.close, objekt.high, objekt.vol etc.

Du kan också använda dig av kurser och volymer för en ordning av instrument. Antag att du vill summera slutkurserna för en ordning, topp16. Då blir modellen för detta följande;

```
par (topp16 : set) : realvector;

var r0 : realvector;

begin
r0 := 0;
        for i := 0 to len(top16) -1 do
r0 := r0 + GetClose(topp16[i], "close");
end;
        return r0;
end;
```

Den första instruktionen här, ”r0 := 0;” gör att vektorn r0 i ”begynnelsen” är rensad från eventuellt skräp.

7.9 Åtkomst av tidsskalan

Hur du sätter en tidsvariabel till ett visst datum eller till ett visst datum och klockslag.

Det gör du med funktionen time.

Med funktionen TimeOfIndex kan du ta fram tidpunkten för ett visst index.

Med funktionen IndexOfTime kan du ta fram index för en viss tidpunkt.

Med funktion TimeUnit kan du bestämma vilken tidsenhet, som skall gälla.

Om du vill låsa ett villkor till senaste tidsperiod, dag, vecka etc, så använder du dig av Now. Skriver du t.ex. dagenskurs := main.close[Now]; så kommer dagenskurs att vara det sista värdet i vektorn av slutkurser för dagsdata.

Du kan också lägga till eller dra ifrån ett antal tidsperioder från en given tidpunkt. Vill du t.ex. gå tillbaka 1 kvartal från idag så skriver du

```
tidpunkt := TimeOfIndex(timevec, now);
tidpunkt := NextQuarter(tidpunkt, -1);
```

I referensmanulen finns mer information om vilka möjligheter du har att ta reda på tidpunkter och index samt ändra dessa.

8 Vad kan du göra ytterligare i modellspråket?

I referensmanualen kan du hitta en mängd uppgifter om vad du kan göra och också information om hur du skall göra det.

Du rekommenderas därför att studera referensmanualen noga. Börja med att skumma igenom den för att vid förnyad läsning fördjupa dig i delar av den till dess du har en klar uppfattning om vad som kan göras.

Skulle du då finna, att modellspråket inte gör det möjligt för dig att operationalisera de hypoteser om ekonomiska förhållande, som du vill testa, ta då kontakt med oss.

Slutligen ger vi dig nedan ytterligare ett exempel på en modell.

9 Ett exempel på en modell

Antag att du har den uppfattningen att RSI och Parabolic tillsammans kan ge en bra prognos av den kommande kursutvecklingen. Du tror att en bra köpsignal fås av endera RSI eller Parabolic och en bra säljsignal först när både RSI och Parabolic signalerar för sälj.

```
// MODELL EXEMPEL                                © Delphi Economics AB/EH
//
// Köp görs om endera RSI eller Parabolic ger köpsignal. Sälj görs då
// både RSI och Parabolic ger säljsignaler.
//
par(Main : Instrument; PerioderRSI, PerioderMAVförRSI : Integer; MaxförParabolic,
TillväxtförParabolic, ÖvreGränsförRSIköp, NedreGränsFöreSälj: real; var( RSI, MAVförRSI,
Parabolic: realvector; Buy, Sell, Short, Cover : booleanvector);

var FilledClose,
begin
        FilledClose := FILL(obj.close);
        RSI := RSI(FilledClose, perioderRSI);
        MAVförRSI := MAV(RSI, PerioderMAVförRSI;
        PAR:=PARAB(obj.close,MaxförParabolic,TillväxtförParbolic);
        Buy := ((BOTD(RSI,1) < 1.5) AND ( RSI < ÖvreGränsförKöp) ) OR (PAR <
FilledClose);
        Sell := (TOPD(RSI,1) < 1.5) AND (PAR >  FilledClose);
        Short := Sell;
        Cover := Buy;
        return FilledClose;
end;
```

Kom ihåg följande;

”//” används för att lägga in kommentarer på en enskild rad i modellen och om du vill lägga in en kommentar som sträcker sig över flera rader så använder du (*och *) för att påbörja och avslutaen kommentar över flera rader

i funktionshuvudet efter par anger du de indata och utdata som skall ingå i modellen

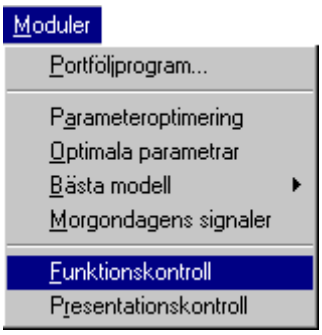
samtliga storheter i modellen som inte är kända av programspråket måste vara definierade i ”par” respektive ”var”

de parametrar och variabler deklarerade under ”par” som skall kunna användas i modellpresentationer måste föregås av ” out”

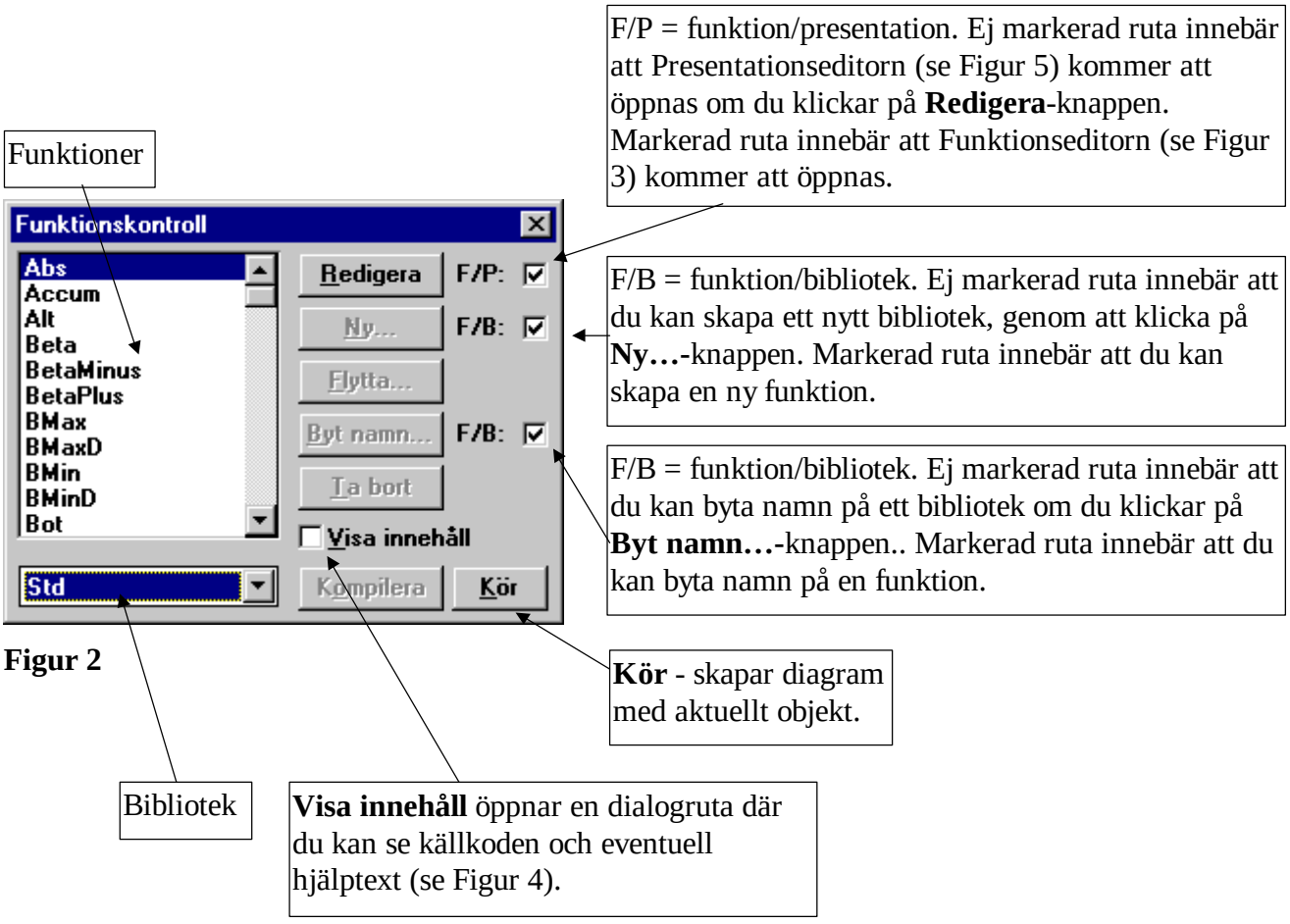
de variabler, tidsserier, som enbart används för mellanberäkningar deklareras under ”var”, i vårt exempel ovan ”FilledClose”.

Funktionskontroll:

För att skapa en funktion välj **Moduler** | **Funktionskontroll** (se Figur 1).



Figur 1



Figur 2

Först öppnas Funktionskontrollsfönstret. Markera den funktion eller de funktioner du vill ändra. Du kan markera flera funktioner genom att hålla ctrl-tangenten nedtryckt samtidigt som du klickar på de funktioner du vill välja.

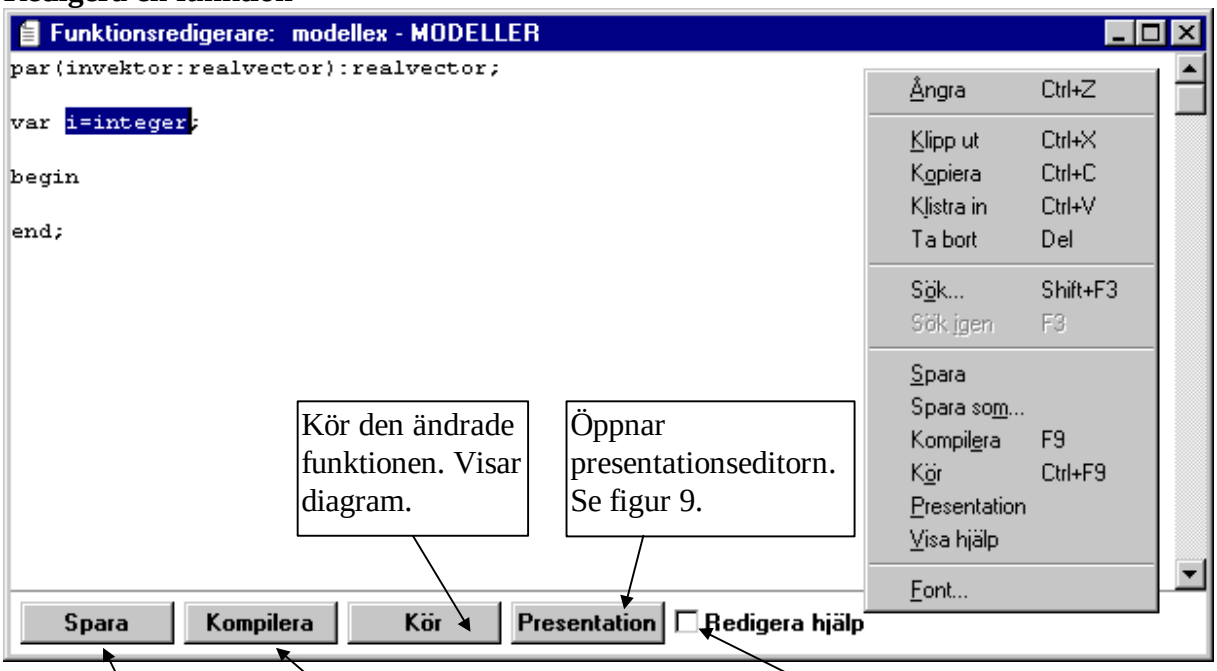
Om du vill kompilera en funktion, så klickar du på **Kompilera.** Om funktionen du valt innehåller fel, får du upp en dialogruta med kompileringsfel. Varje fel står på en egen rad. Du kan välja att dubbelklicka på en av raderna och får då upp *Funktionseditorn* där du kan rätta till felet. Det specifika fel du valde att klicka på kommer att vara markerat.

Om funktionen du valt inte innehåller fel kommer inget meddelande att visas.

Vill du flytta en funktion till ett nytt bibliotek, så markerar du funktionen och trycker på **Flytta**-knappen. Du kommer att få ange destinations bibliotek i en dialogruta innan funktionen flyttas.

Ta bort-knappen fungerar så att den markerade funktionen tas bort. När alla funktioner i ett bibliotek tagits bort kan du välja **Ta bort** ytterligare en gång och då tas biblioteket bort.
Du kommer att få bekräfta att du vill ta bort funktionen/biblioteket.

Redigera en funktion



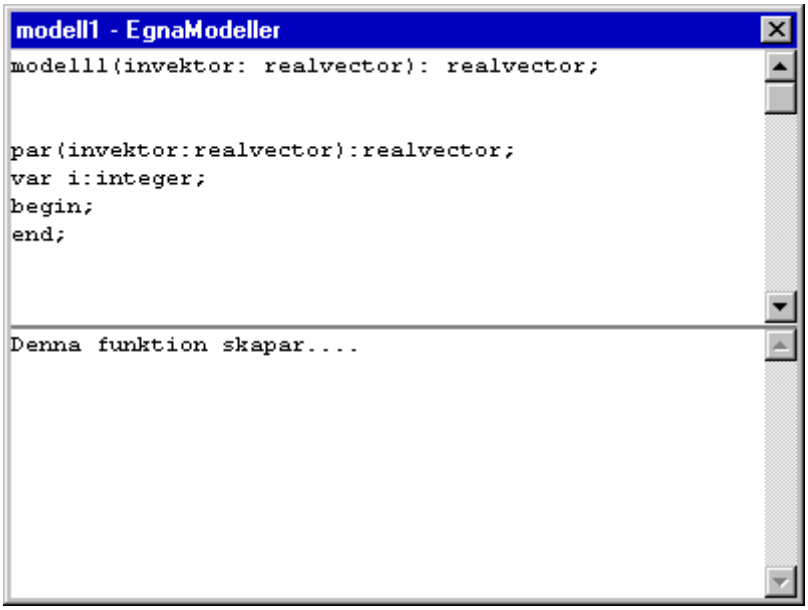
Figur 3

Du kan välja att redigera en funktion genom att klicka på **Redigera**-knappen när rutan F/B är markerad.
De standardfunktioner som ingår i Std-biblioteket går inte att redigera. De är skrivskyddade.
Egna modeller går att redigera i dialogrutan ovan.
Om du valt att skapa en ny funktion, så kommer det att finnas en funktionsmall inlagd. Du fyller sedan på med dina specifika variabler, vektorer mm.
Förutom med de funktions-knappar som syns längst ner i fönstret kan du redigera funktionen med hjälp av de funktioner som förekommer på snabbmenyn. Högerklicka när pekaren är inom funktionsfönstret. Du kan här välja att ändra **Font**, **Kopiera**, **Sparar som…** mm. De alternativ som finns på **Redigera**-menyn i huvudmenyn kan också användas.

För att köra funktionen kan du förutom att klicka på knappen **Kör**, även välja **Analysera** | **Kör** på huvudmenyn.

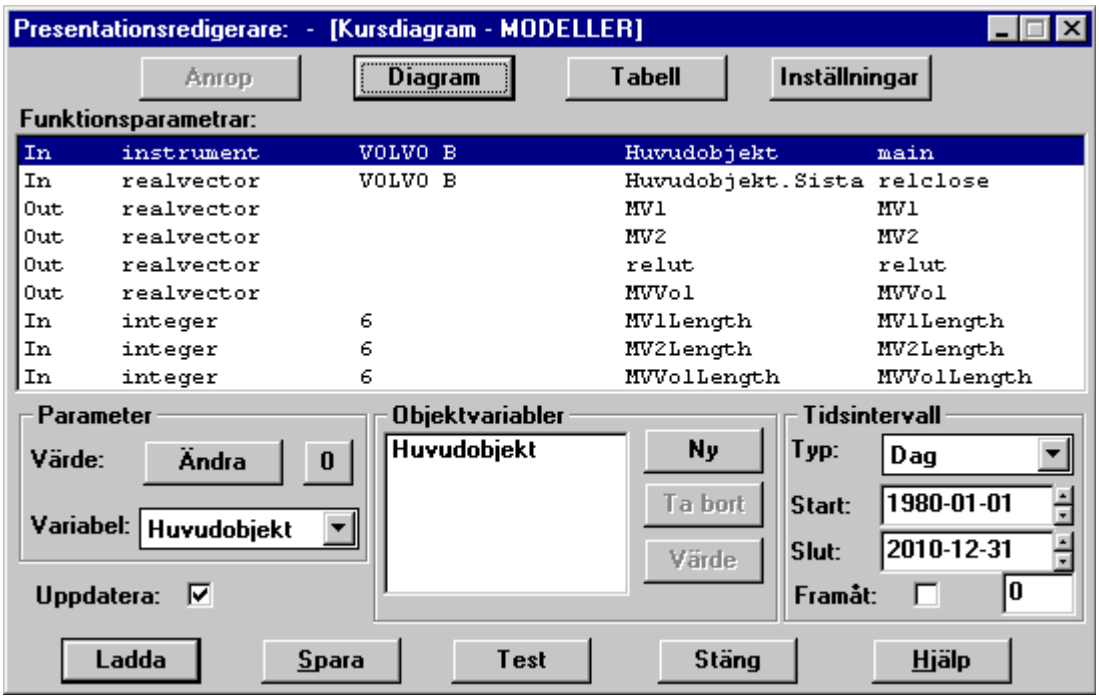
Visa innehåll i en funktion

Om du väljer att markera **Visa innehåll** i *Funktionskontrollfönstret* så kommer en dialogruta, med källkoden i den övre rutan och hjälpfilen i den undre rutan, att visas.



Figur 4

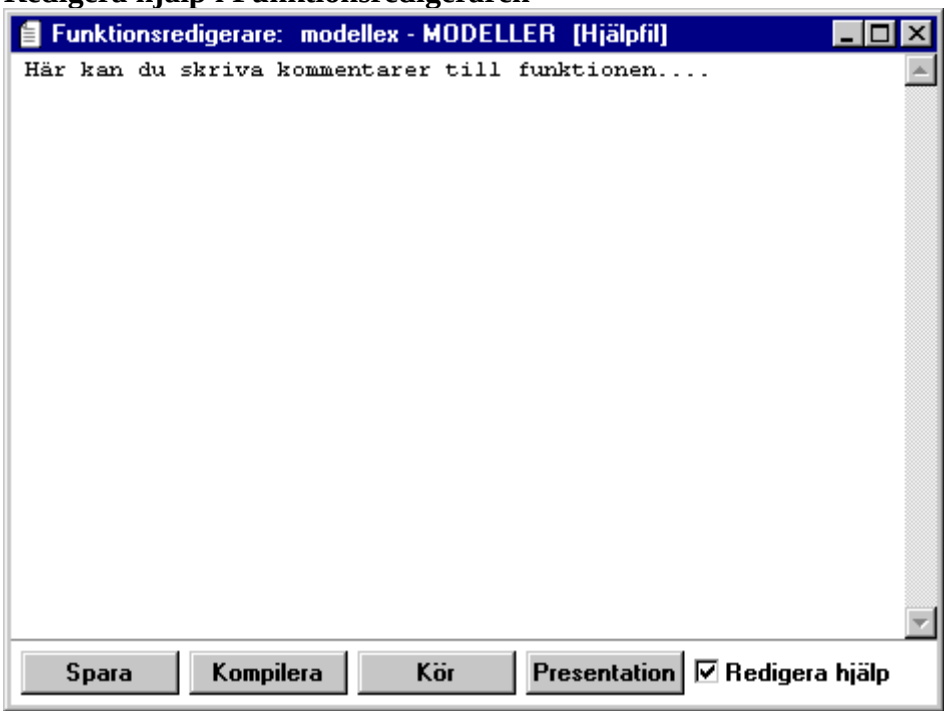
Här kan du se källkoden i den övre rutan och hjälpfilstexten i den undre rutan.
Det här fönstret visas konstant om man markerat rutan i *Funktionskontroll*. Om du byter funktion i *Funktionskontroll*, så kommer fönstret att istället visa den nyvalda funktionen.
I funktioner du själv skapat visas överst ett funktionshuvud (uppräkning av parametrarna).



Figur 5

För mer information om presentationseditorn, se särskild funktionsspec.

Redigera hjälp i Funktionsredigeraren



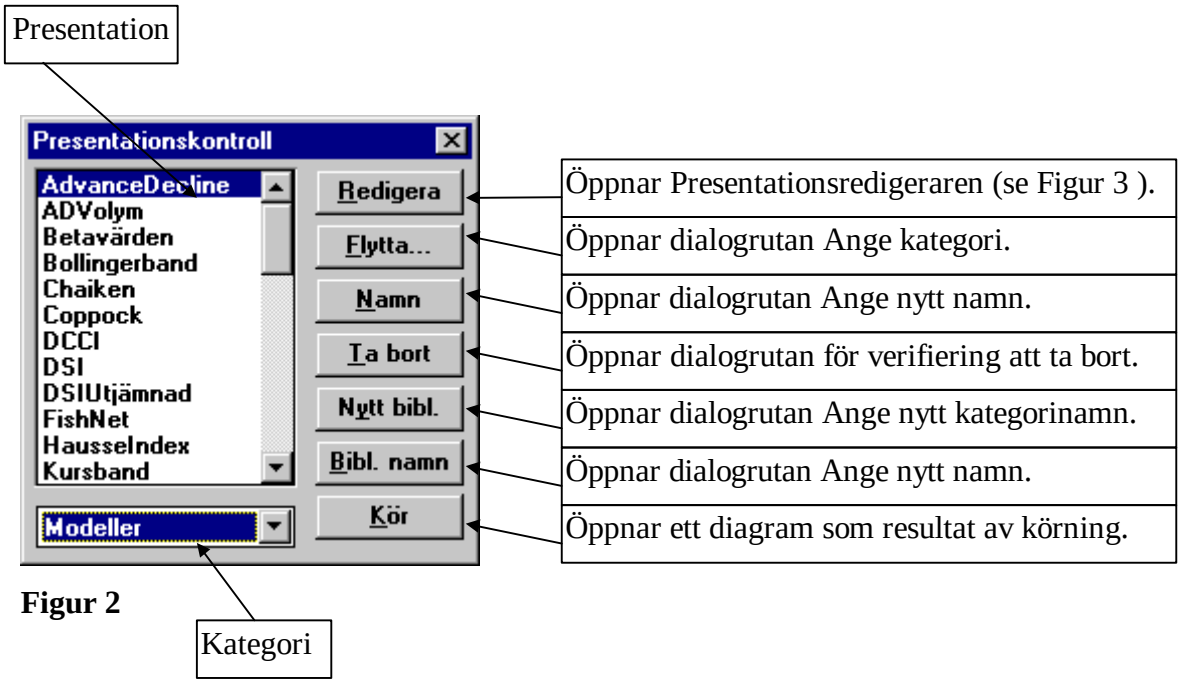
Figur 6

Funktionsredigeraren övergår från att visa källkoden till att visa funktionens hjälpfil. Du kan här skriva kommentarer till dina funktioner. Hälpfiler kan endast skrivas för de funktioner du själv skapat. Standardfunktionerna är skrivskyddade. Funktionens hjälpfil kommer att visas om du väljer **Hjälp** genom att trycka **F1** när diagrammet eller tabellen som är resultatet av körningen är aktivt. Om du i *Presentationseditorn* (se Figur 5) skapar en presentation av din funktion och i presentationen skapar en hjälpfil, så kommer den att visas i första hand.

Presentationskontroll:

För att ändra presentationen av en modell, väljer du **Moduler** | **Presentationskontroll** (Figur 1).

Figur 1



Figur 2

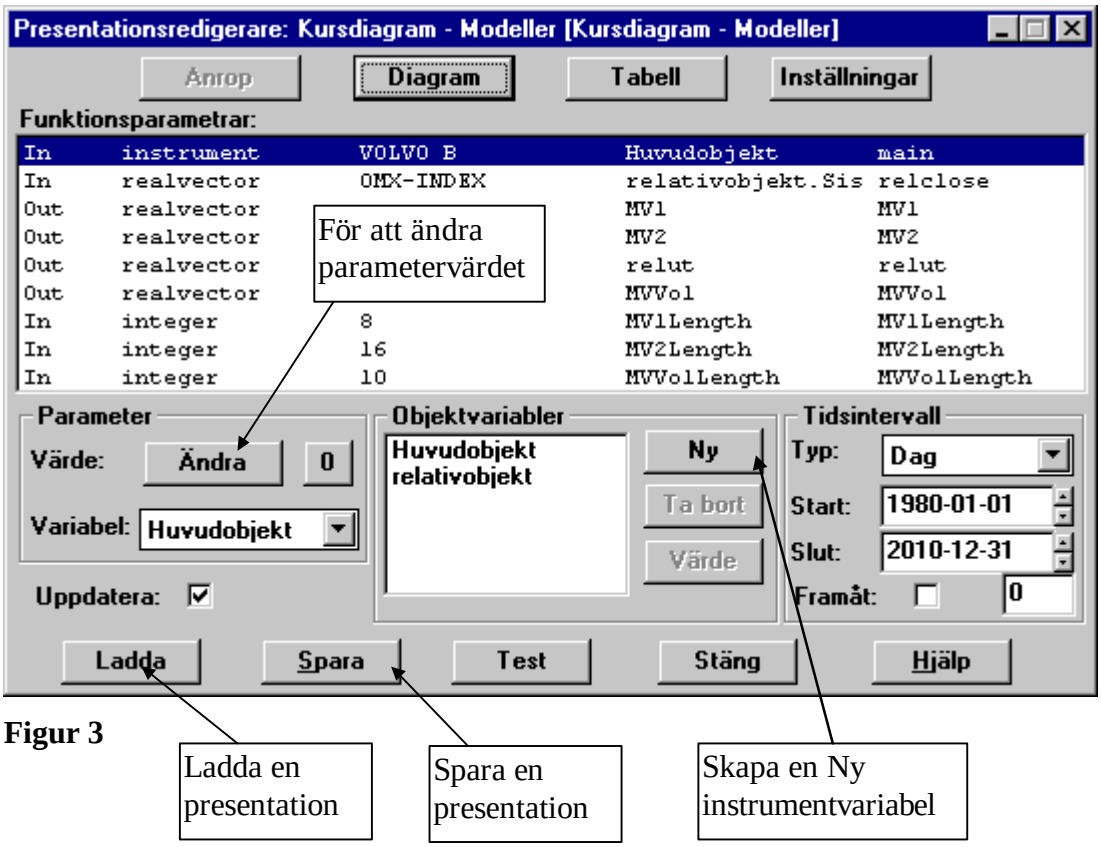
I dialogrutan *Presentationskontroll* kan du utföra ett flertal olika ändringar på dina presentationer. I den stora rutan ser du namnen på de presentationer du skapat. Du skapar en presentation genom att skapa eller öppna en funktion under **Moduler** | **Funktionskontroll** och när du redigerar funktionen, klicka på knappen **Presentation** (se Figur 1).

För att utforma presentationen klickar du på **Redigera**, då får du upp *Presentationsredigeraren* (se Figur 3).
I listrutan i botten på dialogrutan ser du de olika kategorier som finns att välja på. Du kan skapa nya kategorier genom att klicka på **Ny Kat.** och sedan ange namnet i den dialogruta du får upp. Om du vill byta namn på kategorin, så klicka på **Kat. namn** och ange namnet i dialogrutan *Ange nytt namn*.

Du kan flytta en presentation till en ny kategori genom att markera presentationen och därefter klicka på **Flytta**. Ange namnet på den kategori du vill flytta presentationen till.

Byt namn på en presentation genom att markera den och klicka på **Namn**. I dialogrutan *Ange nytt namn* skriver du in namnet.

Ta bort-knappen fungerar så att den först tar bort den presentation som är markerad och när du tagit bort alla presentationer i en kategori, så tar den bort kategorin.



Figur 3

Högst upp i fönstret ser du titelraden. Den talar om vilken presentation, från vilken kategori som visas. Därefter följer inom parantes vilken funktion, från vilket bibliotek som ligger till grund för presentationen.

Under titelraden finns fyra knappar, där kan du välja vilken del i presentationen du vill förändra:

Anrop - för hur funktionen ska anropas.
Diagram - för hur diagrammen ska utformas.
Tabell - för hur tabellerna ska utformas.
Inställningar - för hur modellinställningarna ska utformas.

I rutan med *Funktionsparametrar* finns en uppräkning på de parametrar, som ingår i modellen, t ex längderna på medelvärdena. Kolumnen längst till vänster visar vilken *parameterkategori* parametern tillhör. Nästa kolumn visar vilken *typ*. Tredje kolumnen visar parameterns *värde*. Fjärde kolumnen visar *variabeln* som ger parametern dess värde. Sista kolumnen visar

parameternamnet. Om du vill ändra på de angivna parametervärdena så markerar du först den aktuella parametern, t ex i och klickar därefter på **Ändra** i rutan *Parameter*. Alternativt, om parametern är instrument eller realvector, kan du välja den önskade parametern genom att rulla fram den i listrutan - *Variabel*. När du klickat på **Ändra** får du upp en dialogruta, där du anger vilket heltalsvärde parametern ska ha. Genom att klicka på **0**-knappen, så nollställer du värdet för den markerade parametern.

Vill du ändra något objekt, t ex om du har något relativobjekt, så markerar du det i rutan med *Objektvariabler*. Klicka på **Ändra** och du får upp objektlistan, där du kan välja ett nytt objekt. Objektvariabeln HuvObj, som är huvudobjekt, byter alltid värde automatiskt till det aktuella objektet när man kör presentationen. Det är således ingen mening med att byta värde i *Presentationseditorn* på den variabeln. Volvo B är här ett defaultvärde.
Du kan inom den här rutan också skapa nya objektvariabler genom att välja **Ny** och ange ett nytt variabelnamn eller ta bort en variabel genom att välja **Ta bort**.

I *Tidsintervall*-rutan kan du välja periodlängd, start- och slutperiod, samt antal perioder framåt i tiden som skall beräknas. Väljer du t ex Vecka och markerar Framåt och sätter antal till 5, så kommer programmet att beräkna fem veckor framåt. Om du väljer Intraday som Typ, så får du ange Intervall i minutrar. Värdet 0 innebär varje tick.

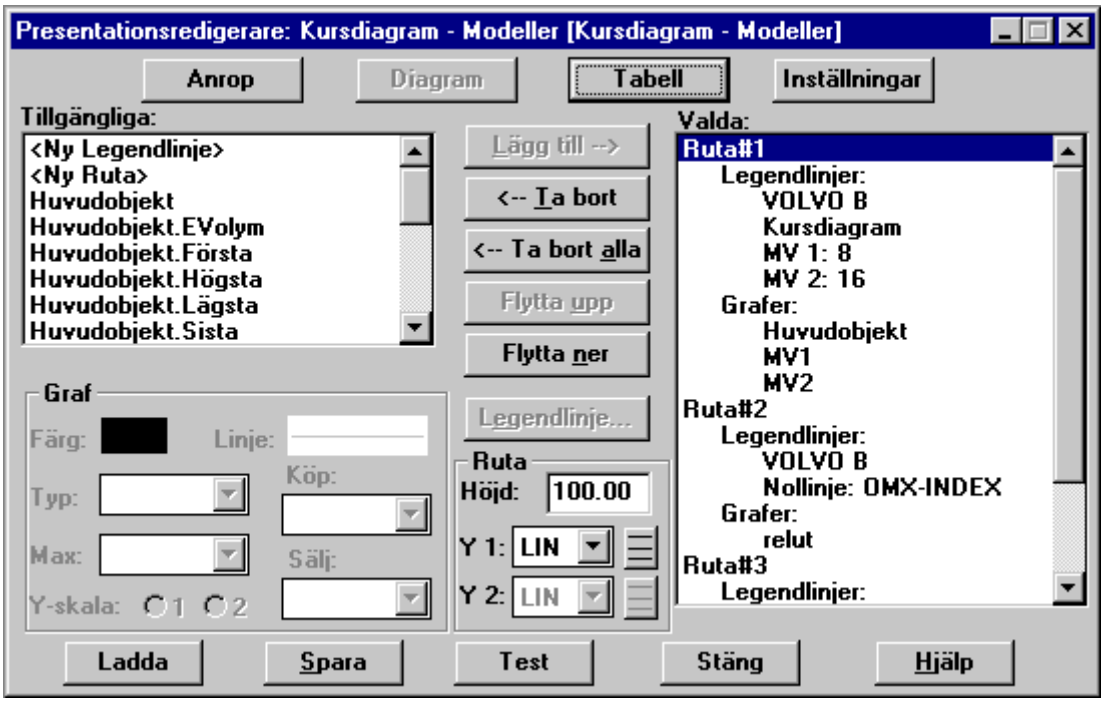
Längst ner i fönstret finns fem funktionsknappar:

Ladda - öppnar en befintlig presentation. Ange kategori och namn på presentationen.

Spara - sparar presentationen i den kategori och under det namn som du anger. Vill du skapa en ny kategori, så kan du göra det genom att skriva in namnet på den nya kategorin.

Test - skapar ett fönster med en kurskurva i den övre rutan och ett volymdiagram i den undre rutan.

Stäng - Om du inte gjort några ändringar i presentationen, så stängs *Presentationsredigeraren*. Om du har gjort ändringar av inställningarna, så kommer du att få välja om du vill spara dem.



Figur 4

Klicka på **Diagram** i *Presentationsredigeraren* för att ändra utseendet på diagrammen. I rutan *Tillgängliga* finns de grafer som du kan ha med i diagrammen, samt två fält <Ny legendlinje> och <Ny ruta>. Om din underliggande funktion innehåller två booleanska utvektorer, så kommer även fältet <Vinsttest> att synas.

I rutan under *Valda* finns de olika diagram, med valda legendlinjer och grafer, som kommer att visas vid körning av presentationen. Det fönster som genereras vid en körning kan innehålla en eller flera rutor. Under *Valda* markeras varje sådan ruta med Ruta#1 osv. För varje ruta väljer du vilka legendlinjer och grafer som ska presenteras.
När du skapar en ny presentation är *Valda* fyllt med standardvärdena för ett Kursdiagram. Du kan välja att använda dem eller byta ut dem.

Under *Graf* kan du välja olika inställningar för de olika graferna i diagrammen. Genom att markera en variabel under rubriken *Grafer* i *Valda* och sedan klicka på färgrutan i rutan *Graf*, kan du ändra färg på grafen. Bland färgvalen finns en ”färgpalett”, som genom att du trycker på den ger dig möjlighet att kombinera en ”egen” färgnyans.

Klicka på linjen vid rubriken *Linje* och du kan välja utseende på grafen. Du gör ditt val genom att markera den önskade linjetypen

Vid *Typ* kan du välja mellan alternativen Linje, Stapel eller Candlestick. Om du valt linje vid *Typ*, men vill ha möjlighet att byta typ under *Modellinställningar* när du kör presentationen, så ska du vid *Max* välja Stapel eller Candlestick. Väljer du Candlestick, så får du möjlighet att välja alla tre alternativen. Om du väljer Stapel, så kommer du att kunna välja mellan Linje och Stapel i *Modellinställningarna*. Har du valt Linje så kommer du inte att kunna byta graftyp i diagrammet.

Vid *Y-skala* kan du välja om du vill ha en, bara höger, eller två, både höger och vänster, Y-skalor. Inställningarna för skalorna gör du under diagrammets snabbmeny när du kör presentationen.

Om din underliggande funktion innehåller två booleanska utvektorer kommer du vid *Köp* och *Sälj* att kunna välja vilken som ska användas som köp- respektive säljvektor för grafen.

Under *Ruta* kan du sätta höjden på de olika rutorna. Du kan också välja om du vill ha linjär eller logaritmisk Y-skala. Dessutom kan du skapa fasta, vågräta nivålinjer för den ena eller för båda av de lodräta skalorna genom att klicka på knappen med linjer till höger om skaltypsrutorna. När du trycker på någon av de båda knapparna, så får du fram uppgifter om de nivålinjer, som finns att välja mellan. Du kan här också, om du vill, ta bort redan inlagda nivålinjer.

Lägga till grafer i en redan skapad ruta
Skulle du vilja lägga till en graf i en redan skapad ruta, så markerar du först rutan, t ex Ruta#1, och därefter namnet på den önskade grafen under Tillgängliga och klickar sedan på **Lägg till**.

Ta bort en redan skapad ruta
Vill du ta bort en ruta, så markerar du rutan, t ex Ruta#1, under Valda och klickar på **Ta bort**.

Ändra namn på en kurva
För att ändra namnet på en kurva trycker du på **Legendlinje.**

Lägga till en ruta
Vill du lägga till ytterligare en ruta så klickar du på <Ny ruta>. Då läggs det in en ny, tom ruta, under rubriken *Valda*.
Du kan flytta den tomma rutan till önskad plats genom att klicka på **Flytta upp** respektive **Flytta ner**.

Lägga in en graf i en ruta
Du börjar med att markera den ruta, under rubriken *Valda*, i vilken du vill lägga in en graf. Under *Tillgängliga* finns de grafer, som du kan välja mellan. Du markerar där den graf, som du vill lägga in, och klickar på **Lägg till**. Då läggs grafen in, vilket du kan se genom att under den aktuella rutan, under *Valda*, kommer namnet på grafen fram under *Grafer*. Texten <tom> läggs till under *Legendlinjer*.

För att ge grafen ett namn, så klickar du på **Legendlinjer…**.
Då får du upp dialogrutan *Redigera Legendlinje*.
Här kan du antingen ange ett av de tillgängliga namnen eller så väljer du att ange ett nytt namn. I det senare fallet markerar du **Ny sträng** och skriver in det nya namnet i rutan bredvid *Sträng*. Därefter klickar du på **OK** och kommer då tillbaka till *Presentationsredigeraren*.

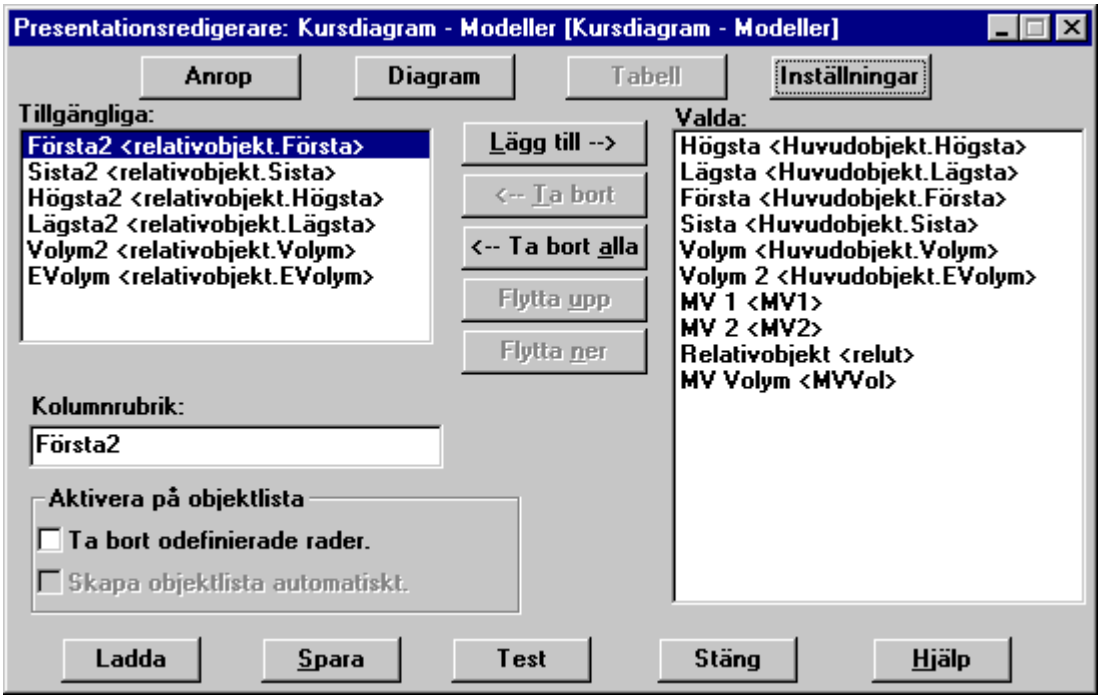
Ta bort en graf från en ruta
Om du vill ta bort en graf från en ruta, så markerar du grafen under *Valda* och klickar på **Ta bort**.

Ändra graftyp och färg

När du klickar på någon av de grafer, som du valt att ha med i din presentation, så kommer du att kunna se den gällande graftypen, linje, stapel eller candlestick, och den gällande färgen. Du har då möjlighet att ändra presentationen av grafen, om du skulle föredra en annan utformning. Kontrollera att du valt rätt ruta och att markeringen gjorts under rubriken *Grafer* och INTE under rubriken *Legendlinjer*.

Du behöver inte spara efter varje grafs inställning, men när du gjort alla inställningar för din presentation måste du spara genom att trycka på **Spara**.

Välja storlek på ruta
Du kan genom att skriva in ett nytt tal, standardvärdet är 100, ändra storleken på en ruta du markerat under *Valda*.

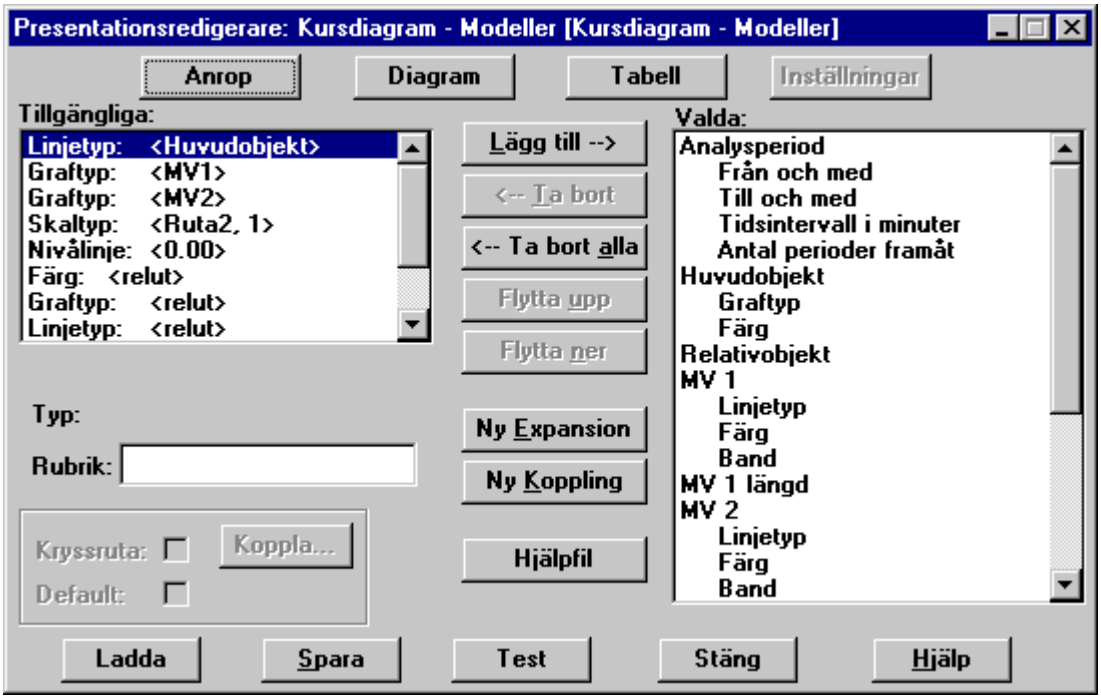


Figur 5

Utforma tabeller
För att utforma tabeller trycker du på **Tabell**-knappen. Du får då fram dialogrutan ovan (se Figur 5).
Här kan du lägga till kolumner, som skall ingå i tabellen, genom att först markera den önskade uppgiften under *Tillgängliga*, och därefter trycka på **Lägg till**.

På samma sätt kan du ta bort kolumner, som ingår i *Valda*, genom att markera uppgiften och trycka på **Ta bort**.

Vill du ändra på en kolumnrubrik, så gör du det genom att under *Valda* markera den rubriken. Den kommer då att synas i rutan under *Kolumnrubrik* och där kan du skriva in den önskade rubriken. Du har möjlighet att få rubriken skriven på två eller flera rader genom att använda \n som radbrytningstecken. Om du tex vill ha rubriken momentumvärde på två rader, så kan du skriva momentum\nvärde.



Figur 6

När du trycker på knappen **Inställningar**, så kommer du att se fönstret ovan. Här bestämmer du hur dialogrutan för *Modellinställningar* ska utformas.
Under *Valda* ser du standardinställningen. Du kan förändra utseendet på *Modellinställningar* genom att lägga till och ta bort de olika alternativen.
Om du markerar en rubrik under *Valda*, så kommer namnet att synas i rutan bredvid *Rubrik*. Du kan skriva in ett nytt namn på rubriken.
De rubriker som under *Valda* står ut i vänsterkant och har en indenterad rubrik under, är expansioner, dvs i *Modellinställningar* kommer de att ha en knapp bredvid sig. När du i sin tur klickar på den öppnas en dialogruta, i vilken du kan välja mellan flera alternativ.
För att skapa en expansion klickar du på **Ny Expansion**, sedan skriver du in rubriken i rutan bredvid *Rubrik*.

För att enkelt kunna slå på och av grafer och kolumner så kan man koppla dem. Genom att koppla grafer och kolumner, så grupperar du dem. Om du t ex inte vill visa volymen i ett diagram och en tabell och volymen är kopplad, så går du in under *Modellinställningar* och avmarkerar rutan för volym. Volymen kommer då varken att visas i digrammet eller i tabellen.
För att skapa en ny koppling, klicka på **Ny Koppling**, skriv in en rubrik. Klicka därefter på **Koppla…**, så får du upp en dialogruta där du ska välja vilka grafer och kolumner som ska kopplas. *Kryssruta* måste vara markerad för att du ska kunna koppla. Du kan välja vilken default du vill ha för kryssrutan - markerad eller omarkerad. Det markerar du i rutan vid *Default*.

Flytta rubriker upp och ner gör du genom att markera den rubrik du vill flytta och sedan klicka på **Flytta Upp** eller **Flytta Ner**. Om du vill flytta upp en rubrik, så ska du tänka på att om rubriken står under en expansion, så kommer den att flyttas in i expansionen. Klickar du

ytterligare en gång på **Flytta Upp** så kommer den att flyttas ur expansionen och hamna ovanför.

När du har sparat din presentation under ett specifikt namn, så kan du välja **Hjälpfil**-knappen och skriva en hjälpfil för presentationen. Observera att om du inte skriver en hjälpfil för presentationen, så kommer funktionens hjälpfil att visas om du väljer Hjälp, F1, när presentationen körs.

Spara en presentation
Du sparar en presentation genom att klicka på **Spara** och välja dels den kategori, som presentationen skall sparas under, dels ge ett namn till presentationen.

Vikingen FutureLook

by

DELPHI Economics

Delphi Finansanalys AB

All text i rött måste kontrolleras eller justeras.

Till ordlistan: kompilator, sträng, kod, uttryck, returvärde

Innehållsförteckning

Referensmanual modellspråket

Inledning

Ett modell skrivs i modellspråket som en *funktion*. En funktion består av ett textdokument som beskriver hur funktionen beräknas och vilka in- och utdata den har. Varje funktion ges ett unikt namn så att den kan identifieras från Vikingen men även från andra funktioner.

Varje funktion tillhör ett *bibliotek*, som helt enkelt är en samling funktioner. För att anropa funktionen *Analys* i biblioteket *MittNyaBibliotek* skriv så här:

```
MittNyaBibliotek.Analys(p1, p2, p3);
```

Om funktionen *Analys* ligger i samma bibliotek som anropande funktion behöver man inte ange biblioteksnamnet.

Med modellspråket följer ett standardbibliotek vid namn ”Std” med en mängd användbara funktioner. Biblioteksnamnet behövs inte anges vid anrop av Std-funktioner. Vilka funktioner som ingår hittar du i <referens till std-kapitlet>.

Här följer ett exempel på hur en komplett funktion ser ut. Denna funktion adderar parametern term till varje element i close-vektorn för ett finansiellt instrument (exv en aktie) och returnerar den därigenom framräknade vektorn.

Symbolen ”:=” betyder *tilldelning*. I nedanstående funktion skulle man kunna omformulera tilldelningssatsen som följer: ”Låt värdet av utvektor[i] vara lika med resultatet av beräkningen objekt.close[i] + term”. Variabeln till vänster *tilldelas* värdet av uttrycket till höger.

Värdet på parametrarna objekt och term sätts vid körning av funktionen till värden som bestäms av användaren, precis som vid körning av vilken modell som helst.

```
par(objekt : instrument; term : integer) : realvector;
var         i : integer;
                        utvektor : realvector;
begin
            for i := 1 to len(objekt.close) - 1 do
                        utvektor[i] := objekt.close[i] + term;
            end;
            return utvektor;
end;
```

Tidsenheter

En funktion tar vanligtvis kursdata som indata. Dessa kursdata kan vara på fyra olika format, *intradaydata*, *dagsdata*, *veckodata* eller *månadsdata*. I Vikingen bestäms tidsenheten från Presentationseditorn eller från verktygsraden.

Via funktionen Std.TimeUnit kan man ta reda på vilken tidsenhet som funktionen körs på. Inom parentes nedan anges vilka värden Std.TimeUnit returnerar för respektive tidsenhet.

Intradaydata (1)

Vilka vektorer finns med alla tick, vilka vektorer finns med x-minuters intervall? (Erland)

Dagsdata (2)

Vilka vektorer finns och vad innehåller dom? (Erland)

Veckodata (3)

Vilka vektorer finns och vad innehåller dom? (Erland)

Månadsdata (4)

Vilka vektorer finns och vad innehåller dom? (Erland)

Kommentarer och mellanslag

Mellanslag, tabbar och radavslut ignoreras av kompilatorn så länge dom inte finns inuti reserverade ord eller identifierare. Följande två exempel är ekvivalenta.

```
utvektor := abs(invektor);
```

```
utvektor :=
   abs(
      invektor
   );
```

Kommentarer i koden kan göras på två sätt.

`a := 12; // Kommentar`	All text efter // fram till slutet av raden är en kommentar, och ignoreras av kompilatorn.

(* Detta är en flerradig kommentar. Detta är en flerradig kommentar. Detta är en flerradig kommentar. ***)**	All text mellan (* och *) är en kommentar och ignoreras av kompilatorn. Kommentaren kan sträcka sig över flera rader.

Identifierare

Identifierare är ett annat ord för variabelnamn eller funktionsnamn. Följande exempel visar hur en identifierare får se ut (och inte får se ut).

Längd	Å, Ä och Ö är tillåtna i variabel- och funktionsnamn.
LÄNGD	Modellspråket skiljer inte mellan stora och små bokstäver. LÄNGD refererar till samma variabel som längd.
längd10	Siffror kan användas så länge identifieraren inte börjar med en siffra.
10längd	Denna sträng är inte godkänd eftersom den börjar med en siffra.
DettaÄrEttLångtNamn	Identifierare kan maximalt vara 149 tecken långa.
Return	Denna sträng är inte godkänd eftersom den är ett *reserverat ord*. Se 0 Reserverade ord (sida 49).

Identifierare kan kombineras med andra identifierare och symboler enligt följande exempel.

Utvektor := **Std.Abs**(invektor)	Anrop av av funktionen Abs i biblioteket Std. Observera att just i fallet med funktioner i Std så är det inte nödvändigt med biblioteksnamnet, utan

	funktionsnamnet räcker.
objekt.close	Variabeln objekt av typen instrument innehåller medlemsvariabler, bland annat en som heter close.
invektor[12]	Skriv så här för att komma åt det 12:fte elementet i vektorn invektor.
objekt.open[i]	Variabeln i är en heltalsvariabel. Om i har värdet 10 kommer man åt element 10 i medlemsvariabeln open i variabeln objekt.
Objektlista[0]	Skriv så här för att komma åt den första vikingkoden i objektlista.

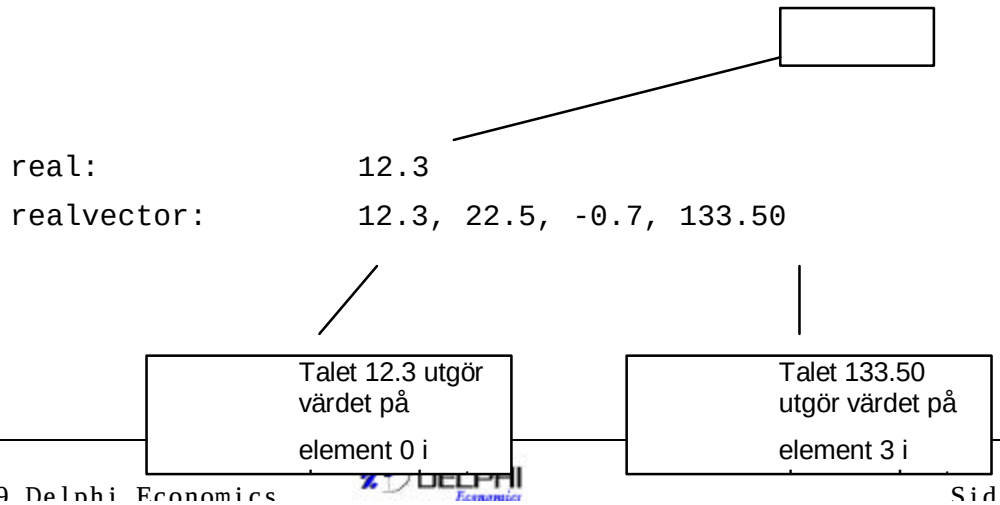
Datatyper

Varje parameter och variabel är av en viss *datatyp*, eller kortare: *typ*. Funktioner som returnerar ett värde är också av en viss typ. Typen anges så här.

```
var index : integer;
```

Variabelnamnet är ”index”. Typen är ”integer”. En variabel kan beskrivas som en lagringsplats för ett värde. Lagringsplatsen har ett namn, variabelnamnet, för att man skall kunna referera till dess värde.

För typerna *integer*, *real*, *boolean* och *time* finns dom motsvarande vektortyperna *integervector*, *realvector*, *booleanvector* och *timevector*. En vektor är i modellspråket en följd av tal (eller andra värden) av samma typ. Innehållet i (tal-)följden kallas *element* och åtkoms via [ ]-parenteser: ”vektor[12]” motsvarar det 12:fte elementet i vektorn.



Vid en given körning av en funktion är alla vektorer lika långa. Längden beror av hur mycket kursdata som laddats upp för huvudobjektet i Vikingen. Användaren bestämmer detta tidsintervall i modellinställningar. Antalet element kan inte ändras inifrån en funktion.

Dom typer som finns är följande.

boolean, booleanvector

En variabel av typen boolean (en ”boolsk” variabel) kan ha ett av två värden, true eller false. En variabel av typen booleanvector, är en vektor med enbart boolska värden. Det första elementet i en vektor har index 0.

`b := true;` `b := false;`	En boolsk variabel kan sättas till antingen true eller false.
`b := 12 > 10;`	Resultatet av en jämförelse är alltid ett boolskt värde. I detta fall kommer b att sättas till true.

`par(rv : realvector) : booleanvector;`
`var köp : booleanvector;`
`begin`
`  köp := rv > mavn(rv, 10);`
`  return köp;`
`end;`
	Vektorn köp sätts till resultatet av en elementvis jämförelse mellan två reella vektorer. Varje element har antingen värdet true eller false.

`if b1 and b2 then` `  // Både b1 och b2 har värdet true.`	Se nedan under rubriken 0

end;	Operatorer (sida 44) för mer information.

if b1 or b2 then
// Antingen b1 eller b2 har värdet
// true.
end;
	Se nedan under rubriken 0 Operatorer (sida 44) för mer information.

par(objekt : instrument; längd : integer);
begin
if **objekt.close > 100** then
 	
end;
end;
 	
Detta uttryck är av typen booleanvector, eftersom **objekt.close** är av typen
	If-satser tar alltid en boolean som parameter.
	
	Denna if-sats ger ett kompileringsfel eftersom uttrycket returnerar en booleanvector, där varje element kan vara true eller false.
	
	Jämförelser mellan booleanvectorer genomförs lämpligtvis i en loop där varje element kontrolleras. Se 0 For, sida 27.

not, and, or, =, <>	Dessa operatorer kan användas med boolean och booleanvector. Se nedan under

	rubriken Operatorer för mer information.

instrument

Typen instrument motsvarar ett värdepappersobjekt – financial instrument. En variabel av denna typ har 7 *medlemsvariabler* när man kör på tidsenheterna *dag*, *vecka* och *månad*.

`rv := objekt.open;`	Sätt rv till objektets open-vektor (öppningskurs).
`rv := objekt.close;`	Sätt rv till objektets close-vektor (slutkurs).
`rv := objekt.high;`	Sätt rv till objektets high-vektor (högstakurs).
`rv := objekt.low;`	Sätt rv till objektets low-vektor (lägstakurs).
`rv:= objekt.vol;`	Sätt rv till objektets vol-vektor (volym).
`rv:= objekt.evol;`	Sätt rv till objektets evol-vektor (efteranmäld volym).
`str1 := objekt.code;`	Sätt sträng-variabeln str1 till objektets vikingkod. Se 0 string, sida 14.

`if objekt1 = objekt2 then`
`  // ...`
`end;`
	Inga operatorer (exv =, <>, +, -) på instrument stöds i den nuvarande versionen av modellspråket. Däremot stöds flera operatorer på ett instruments medlemsvektorer, se 0 real, realvector, sida12.

När en modells körs på *intradaydata* har den 5 medlemsvariabler.

`rv := objekt.last;`	Sätt rv till objektets

	last-vektor (senastekurs).
rv := objekt.**ask**;	Sätt rv till objektets ask-vektor (säljkurs).
rv := objekt.**bid**;	Sätt rv till objektets bid-vektor (köpkurs).
rv:= objekt.**vol**;	Sätt rv till objektets vol-vektor (volym).
str1 := objekt.**code**;	Sätt sträng-variabeln str1 till objektets vikingkod.

integer, integervector

Integer betyder heltal. Heltal används till exempel som index i vektorer. En heltalsvektor - integervector - innehåller enbart heltal. Det första elementet i en vektor har index 0.

par(värde : integer) : integervector;
var heltalsvektor : integervector;
begin
heltalsvektor[20] := värde;
return heltalsvektor;
end;
	I denna funktion sätts element 20 i heltalsvektor till värde. Övriga element i vektorn förblir odefinierade.

+, -, *, /, =, <>, <, <=, >, >=	Dessa operatorer kan användas med integer och integervector. Se nedan under rubriken Operatorer för mer information.

index := 2147483646;	Detta är det högsta heltalet som kan användas.
index := 2147483647;	Detta ger kompilerings fel. Värdet är för högt.

if undef(i) then
// i är odefinierad, och bör inte
// användas i fortsatta beräkningar.
Else
// i är ok, och kan användas i
// fortsatta beräkningar.
end;
	En integer-variabel kan ha ett odefinierat värde. Detta testas med funktionen undef.

real, realvector

Aktiekursen för en viss dag är av typen real (decimaltal). Aktiekurserna för flera dagar i en följd är av typen realvector. Vektorn close i en variabel av typen instrument är alltså av typen realvector. Det första elementet i en vektor har index 0.

utvektor := objekt.close;	Kopiera varje element i objekt.close till utvektor.
utvektor[0] := 12.23;	Sätt första element i utvektor till 12.23.
r := 2000.0	Sätt r till 2000.0.
r := 2.0E3;	Ett annat sätt att sätta r till 2000.0 ($2.0 * 10^3$).
r := 0.002;	Sätt r till 0.002.
r := 2.0E-3;	Ett annat sätt att sätta r till 0.002 ($2.0 * 10^{-3}$).

+, -, *, /, =, <>, <, <=, >, >=	Dessa operatorer kan användas med real och realvector. Se nedan under rubriken Operatorer för mer information.

if undef(r) then
  // r är odefinierad
else
  // r är ok.
end;
	En real-variabel kan ha ett odefinierat värde. Detta testas med funktionen undef.

set

Vikingens objektlistor kan hanteras i modellspråket. Dess datatyp heter set. Ett set är i modellspråket en vektor av vikingkodsträngar. Det vanligaste sättet att använda ett set är att anropa GetData för att hämta in data för en viss vikingkod. Det första elementet i ett set har index 0.

En vikingkod är en sträng av bokstäver och siffror som är högst 11 tecken lång. Till exempel är ”190072” vikingkoden för OMX-INDEX.

par(var objektlista : set);
var rv : realvector;
begin
  for i := 0 to len(objektlista) - 1 do
  GetData(
  Rv,
  "PriceData", // Databas
  objektlista[i], // Vikingkod
  "close", // Medlemsvariabel
  "", "", 0, 0, 0, 0, 0, 0
  );
  end;
  // Gör nåt här med inladdade data
end;
	Hämta in close-vektorn för varje objekt i objektlista.

RemoveFromSet(objektlista, 0);	Ta bort det första elementet ur objektlista.
AddToSet(objektlista, "190072");	Lägg till vikingkoden ” 190072” till objektlista.

if objekt1 = objekt2 then
  // ...
end;
	Inga operatorer på set stöds i den nuvarande versionen av modellspråket.

string

String, eller sträng på svenska, är en följd av tecken. Ett tecken är en bokstav, en siffra eller en annan symbol på tangentbordet. Textsträngar används för olika ändamål i modellspråket, se exempel nedan.

par(objekt : instrument) : realvector;
var rv : realvector;
begin
	Strängar används i anrop till GetData för

GetData(rv, "PriceData", // Databas "100115", // Vikingkod "close", // Medlemsvariabel "", "", 0, 0, 0, 0, 0, 0); return rv; end;	att ange vilken databas, vilket objekt och vilken medlemsvariabel som skall laddas in.
Str1 := 'Hejsan'; str2 := "Hejsan";	En sträng avgränsas med ' eller ".
str1 := "Vem är "Alan Greenspan" ?"; str2 := 'Vem är 'Alan Greenspan'.';	Dessa fungerar inte.
str1 := "Alan är en 'finansnörd' ?"; str2 := 'Alan är en "finansnörd".';	Dessa fungerar utmärkt. Om " skall finnas med i strängen, använd ' som avgränsare, och vice versa.

if str1 = str2 then
// ...
end;
	Inga operatorer på strängar stöds i den nuvarande versionen av modellspråket.

time, timevector

Datatypen time representerar en tidpunkt. Se <std library-kapitlet> för mer information om användningen av de tidsrelaterade funktioner som förekommer i nedanstående exempel.

Datatypen timevector används endast i form av konstanten timevec, som behövs som parameter till vissa funktioner (se nedan). Timevector kan inte användas som typ för en parameter eller variabel.

```
par(var tidpunkt1 : time) : time;
var tidpunkt2 : time;
begin
   tidpunkt1 := time("97-02-19");
   tidpunkt2 := time("1992-09-01 12.20.30");
   return tidpunkt2;
        end;
```

Datatypen heter time. För att sätta en tidsvariabel till ett visst datum och klockslag så används *funktionen* time som tar ett sträng med en tidsuttryck som parameter.

Tidpunkt2 := **TimeOfIndex**(timevec, 0);	Funktionen TimeOfIndex används för att ta fram tidpunkten för ett visst index, i detta fall 0.
Index := **IndexOfTime**(timevec, time("930202"));	Funktionen IndexOfTime gör tvärtom, den returnerar index för en tidpunkt.
DistansKalenderdagar := DistanceInDays(time("1890-01-01"), time("1990-01-01"));	Funktionen DistanceInDays returnerar antalet kalenderdagar mellan två tidpunkter.

DistansBörsdagar := **IndexOfTime**(timevec, time("1997-01-01")) - IndexOfTime(timevec, time("1996-01-01"));	Beräkna antal börsdagar mellan två datum genom att ta skillnaden i index mellan dessa datum. Kursvektorer innehåller endast börsdagar.

if not undef(tidpunkt) then
// tidpunkt är ok
end;
	En tidpunkt kan vara odefinierad. Använd undef för att testa detta.

if TimeUnit(timevec) = 1 then
// Intraday
end;
	Använd funktionen TimeUnit för att ta reda på vilken tidsenhet som man kör på. TimeUnit returnerar:

	1 - Intraday 2 - Dag 3 - Vecka 4 - Månad
DagensClose := main.close[Now];	Now är ett index som alltid pekar på dagens värde i kursvektorerna (eller annan tidsenhets aktuella tidpunkt).

if TimeUnit(timevec) = 1 and Minutes = 0 then
// Alla tick
end;
	Minutes innehåller tidsintervallet vid körning på intraday. Om Minutes är noll, så körs modellen på samtliga tick.

=, <>, >, >=, <, <=	Dessa operatorer kan använda

	s med tidpunkter. Se nedan under rubriken Operatorer för mer information.

tidpunkt := time("2000-01-01");
 	
// Lägg till två kvartal
tidpunkt := NextQuarter(tidpunkt, 2);
// Nu är tidpunkt lika med 2000-07-01
 	
// Dra ifrån 3 dagar
tidpunkt := NextDay(tidpunkt, -3);
// Nu är tidpunkt lika med 2000-06-28
	Använd funktionerna NextSecond, NextMinute, NextHour, NextDay, NextWeek, NextQuarter och NextYear för att lägga till en eller flera tidsenheter till en tidpunkt.
	
	Observera att funktionerna också klarar negativ

	a tal.

Korttids- och långtidsformat

Tidpunkter finns i två varianter, korttidsformat och långtidsformat.

Korttidsformat innehåller uppgifter om både datum och tidpunkt. Långtidsformatet innehåller endast datum. För intraday används alltså korttidsformatet.

Nackdelen med korttidsformatet är att det bara kan representera tidpunkter mellan "1990-01-01 00.00.00" och "2053-12-31 23.59.59". Om man försöker använda en tidpunkt utanför detta intervall så returneras undef.

Korttidsformatet innehåller endast jämna sekunder, om man försöker använda 12.02.23 så konverteras detta automatiskt till 12.02.22.

En tidpunkt lagras på långtids- respektive korttidsformat beroende på om klockslag anges eller inte.

Här följer exempel på tidpunkter i korttids- och långtidsformat.

```
// Ange en tidpunkt på korttidsformat
tidpunkt1 := time("960612 17.34.34");

// Detta blir undef eftersom datum inte anges
tidpunkt1 := time("17.34.34");

// Tidpunkten ligger utanför det godkända
// intervallet för korttid, tidpunkt1 blir
// undef.
tidpunkt1 := time("1989-06-12 17.34.34");

// Detta fungerar eftersom långtidsformat
// används när inte klockslag anges.
```

```
tidpunkt1 := time("1989-06-12");


if time("1997-01-01 12.02.23") =
   time("1997-01-01 12.02.22") then
   // Detta är sant, eftersom endast jämna
   // sekunder lagras.
end;
```

Funktionshuvud

Funktionens första rad som alltid inleds med ”par” (= ”parameter list”) kallas funktionens huvud. ”Par” följs alltid av ”(<lista av parametrar>)”. Funktionens namn står alltså inte med i koden.

Funktionshuvudet kan se ut på följande sätt.

`par(objekt : instrument);`	Funktionen tar en parameter av typen instrument, som fått namnet objekt.
`par(objekt1, objekt2, objekt3 : instrument);`	Funktionen tar tre parametrar av typen instrument, som fått namnen objekt1, objekt2 och objekt3.
`Par(objekt : instrument; längd : integer);`	Funktionen tar två parametrar, den första av typen instrument och den andra av typen integer. Den andra parametern har fått namnet längd.
`par(invektor : realvector; var utvektor : realvector);`	Funktionen tar två parametrar, båda av typen realvector. Den andra parametern vid namn utvektor är *var-deklarerad*. Var står för ”variabel” vilket betyder att när funktionen anropas måste denna parameter vara en variabel. Ändringar som görs till denna variabel inuti funktionen görs

	direkt på originalet, och inte på en kopia vilket är normalfallet. Att var-deklarera en parameter innebär att den kan användas för att bära resultatet av en beräkning.
par(invektor : realvector; var utvektor1, utvektor2 : realvector);	Både utvektor1 och utvektor2 är var-deklarerade.
par(objekt : instrument) : realvector;	Efter avslutande parentes följer ett kolon samt funktionens *returtyp*. Denna funktion kan därmed anropas (från en annan funktion) som ett uttryck eller en del i ett uttryck. Om funktionen heter mav skulle man kunna anropa funktionen så här: ”rv := mav(obj)”. Detta är det andra sättet en funktion kan leverera sina utdata (det första är via var-parametrar enligt exempel ovan).

Variabeldeklarationer

Om funktionen använder sig av variabler som inte är parametrar, följs funktionshuvudet av variabeldeklarationer.

Startvärden

Ett startvärde är det värde en variabel har precis efter ”begin”, innan någon kod har hunnit köras.

Variabler av typen integer, integervector, real, realvector och time sätts till odefinierat som startvärden. Använd funktionen undef för att testa om ett värde är odefinierat.

Bool och booleanvector sätts till false som initialvärde. En string sätts till "" (tom sträng). Ett set är tomt från början (len(setvariabel) returnerar 0).

Exempel

Variabeldeklarationer kan se ut på följande sätt.

`Var`	Funktionen använder inga variabler. Här kan man helt utsluta var och forsätta direkt med begin.
`var i : integer;`	Deklaration av variabeln i av typen integer.
`var i, j, k : integer;`	Deklaration av variablerna i, j och k av typen integer.

`var i : integer;`
`j : integer;`
`r : real;`
`var b1, b2 : booleanvector;`
	Deklaration av fem variabler av varierande typ. Som synes behövs var endast en gång, men kan användas vid följande deklarationer om så önskas.

Kod

Efter variabeldeklarationerna kommer själva *koden* som består av en eller flera rader med *satser* inneslutna mellan begin och end;. Ordet *sats* kan sägas vara synonymt med *kommando*.

`Begin`
`// Här finns en massa`
`// satser`
`BeräknaSumman(a, b);`
`...`
`end;`
	begin följs av ett godtyckligt antal rader satser, följt av end;.
	
	Glöm inte att alltid avsluta med semikolon.

Satser, funktioner och konstanter

Kodens huvudbeståndsdelar är satser och *uttryck*. Uttryck kan inte stå ensamma utan måste finnas i en sats.

`i := k + m * x;`	Detta är ett tilldelningssats. Den innehåller uttrycket (k +

	m * x).

Här följer en beskrivning av alla satser, specialfunktioner (len och undef) och konstanter (now, minutes, timevec) som ingår i modellspråket. Listan är i bokstavsordning.

AddGraphic

Med AddGraphic kan man lägga till grafiska element – trendlinjer och textsträngar – i diagrammen.

AddGraphic(

Grafvariabel,

Denna parameter är valfri, och kan utelämnas.

Om grafvariabeln finns med kommer det grafiska elementet att kopplas till den graf som variabeln motsvarar. Detta innebär att elementet alltid placeras i samma pane som grafvariabeln.

Om variabeln utelämnas kopplas elementet alltid till det översta panet.

Variabeln måste vara en var-deklarararerad parameter av typen realvector.

Typ, — I nuläget stöds typerna "Line" och "Text".

Text, — Texten som skrivs ut i diagrammet, om typen är "Text". Om typen är "Line" så ignoreras denna sträng.

x1, — Startposition i längdled. Ett heltal som anger på vilket vektorindex texten eller trendlinjen skall starta. Använd IndexOfTime för att hitta rätt index för en viss tidpunkt.

y1, — Startposition i höjdled. Ett reellt tal som lämpligvis sätts till ett värde nära rv[x1], om rv är den vektor som det grafiska elementet är kopplat till.

x2, — Slutposition i längdled. Ett heltal som anger på vilket vektorindex trendlinjen skall sluta. Detta värde ignoreras för Texter. Använd IndexOfTime för att hitta

	rätt index för en viss tidpunkt.
y2	Slutposition i höjdled. Ett reellt tal som lämpligvis sätts till ett värde nära rv[x2], om rv är den vektor som det grafiska elementet är kopplat till. Detta värde ignoreras för Texter.
);	

Här följer några exempel på använding av AddGraphic.

par(var rv : realvector) : realvector;
var x1, x2 : integer;
utvektor : realvector;
begin
x2 := len(rv) - 1;
x1 := x2 - 50;
AddGraphic(rv, "Line", "",
x1, rv[x1],
x2, rv[x2]);
end;
	Placera en trendlinje från och med 50 dagar från sista dag i vektorn till och med sista dag.
	
	Start- och slutposition i höjdled ligger på respektive dags vektorvärde.

AddGraphic(rv, "Text", "Köp NU!", x1, rv[x1], 0, 0.0);	Samma sak som ovan fast texten "Köp NU!" placeras ut.
AddGraphic("Text", "Köp NU!", x1, rv[x1], 0, 0.0);	Samma som ovan fast texten följer inte med en speciell variabel utan hamnar

	alltid i översta pane.

AddToSet och RemoveFromSet

AddToSet och RemoveFromSet lägger till och tar bort vikingkodsträngar från en objektlista.

par(var objektlista : set);
begin
AddToSet(objektlista, "100100");
end;
	Vikingkoden 100100 läggs till på slutet av objektlista.

par(var objektlista : set);
begin
RemoveFromSet(objektlista, 78);
end;
	Den 78:de vikingkoden tas bort från objektlista.
	
	Om inte element 78 finns genereras ett körningsfel.

For

For-satsen, eller *for-loopen* används när man vill utföra en kodsekvens för varje element i en vektor. For-loopen tar tre parametrar, den första kallas loopvariabel och är av typen integer. Variabelns värde ökas med ett för varje varv i loopen. Dom andra två parametrar är start- och slutvärde för loopvariabeln.

Om start- och slutvärdena är lika kommer repetitionen att göras en gång. Om startvärdet är större än slutvärdet kommer repetitionen inte att göras alls.

- Värdet på loopvariabeln får inte ändras i kodsekvensen.
- Observera att slutvärdet endast beräknas en gång.
- For-satsen måste avslutas med end;.

for loopvariabel := start to slut do	Så här ser en

// Gör nåt här **end;**	for-loop ut.

for i := 0 to len(objekt.close) - 1 do
utvektor[i] := objekt.close[i];
end;
	Kopiera hela objekt.close till utvektor.

Funktionsanrop

Funktioner som inte har ett returvärde kan anropas som en sats. Funktioner som har ett returvärde måste användas som ett uttryck eller del av ett uttryck.

par(objekt : instrument);
begin
MyFunction(objekt.close);
end;
	Anropa den egna funktionen MyFunction med parametern objekt.close.

par(objekt : instrument) : realvector;
var rv : realvector;
begin
// Det första anropet
Abs(objekt.close);
 	
// Det andra anropet
rv := Abs(objekt.close) + 5;
 	
return rv;
end;
	Det första anropet ger kompileringsfel eftersom Abs returnerar en realvector.
	
	Det andra anropet fungerar eftersom anropet används som ett uttryck.

// Anropa MyFunction i MyLibrary-
// biblioteket.
MyLibrary.MyFunction(x);
	För anrop av funktioner i andra bibliotek måste biblioteksnamnet och en punkt placeras framför funktionsnamnet.

// Anropa Abs i Std-biblioteket rv := Abs(objekt.close);	Funktioner i Std-biblioteket

	kan anropas utan att biblioteksnamnet ges.

// Anropa Abs i samma bibliotek
rv := Abs(objekt.close);
 	
// Anropa Abs i Std-biblioteket
rv := Std.Abs(objekt.close);
	Om funktionen Abs även finns i aktuellt bibliotek så anropas den funktionen istället för Abs i Std-biblioteket.
	
	Om Std.Abs skall anropas så måste biblioteksnamnet placeras framför funktionsnamnet.

GetData

GetData är en generell funktion som hämtar in data från olika databaser. I nuläget stöds inhämtning av kursdata och fundamentaldata.

Vid varje körning av en funktion utgår alla beräkningar från ett aktuellt objekt (instrument) och en aktuell tidsvektor. För tidsenheterna dag, vecka och månad innehåller tidsvektorn alla börsdagar/veckor/månader, även om aktuellt objekt saknar data för en viss dag/vecka/månad. Om kursdata saknas innehåller kursvektorerna ett odefinierat värde.

Om ett objekt som laddats med GetData innehåller data på dagar som inte finns i det aktuella objektets tidsvektor, kommer dessa data att filtreras bort. Detta bör dock inte ske speciellt ofta.

För tidsenheten intraday innehåller tidsvektorn bara tidpunkter där data finns för aktuellt objekt. Eftersom flödet av kursdata ofta är sporadiskt kommer olika objekt i databasen att innehålla data i olika tidsintervall. Vid användning av GetData för uppladdning av intradaydata för ett objekt annat än det aktuella kommer därför kursvektorerna att anpassas enligt det aktuella objektets tidsvektor. Nedanstående exempel visar hur denna anpassning utförs.

Objekt1 är det aktuella objektet, som nedan innehåller kursdata för sex tidpunkter. Objekt2 är laddas upp med GetData. Det innehåller kursdata för sju tidpunkter i databasen.

Tidpunkter	Index	Objekt1.last som den ser ut i funktionen och databasen	Objekt2.last som den ser ut i databasen	Objekt2.last som den ser ut i funktionen (efter uppladdning med GetData)
11.33.40			328	
11.33.44			330	
12.20.00	0	122		330
12.20.24	1	122.50		odefinierat värde
12.35.44	2	122	329.50	329.50

12.36.00			330.50	
13.22.10	3	121.50		330.50
13.22.14	4	121	331	331
12.22.22	5	122	331.50	331.50
12.22.24			333	

Data flyttas alltså i tiden för att inte försvinna helt och hållet. Denna metod används också vid laddning av fundamentaldata. Fundamentaldata är ofta placerat på sista dagen i varje månad, vilket bara ibland är en börsdag. För att dessa data inte skall försvinna i uppladdningen placeras dom på närmast efterföljande tidpunkt i det aktuella objektets tidsvektor.

GetData finns i två varianter, en som tar 12 parametrar och en som tar 13. I den trettonde parametern lagras en eventuell felkod. Om den trettonde parametern saknas omvandlas ett eventuellt fel till ett körningsfel. Det kan vara bra att använda 12-parametersvarianten vid utveckling, för att växla till 13-parametersvarianten för den färdiga funktionen. Genom att ta hand om felkoden i funktionen undviker man att användaren får konstiga felmeddelanden vid körningen av en modell.

Parametrarnas innebörd beror av vilken databas som data laddas ifrån. Nedan används kursdatabas och fundamentaldatabas som exempel.

GetData(	
Resultat,	Här hamnar resultatet av uppladdningen. Denna variabel kan vara av typen instrument, realvector eller integer, beroende på vilka data som skall laddas.
Databas,	En sträng som talar om vilken databas som skall användas. I nuläget stöds följande tre: ”PriceData” för kursdata, ”FundData” för fundamentaldata samt ”FundInfo” för speciella data som används för fundamentalanalys.
Vikingkod,	Vikingkod för det objekt som data skall laddas för.
Vektornamn,	Namn på medlemsvektor i ett instrument (”close”) eller kortnamn på en fundamentalvektor (”AXF”).
String1,	En i nuläget oanvänd strängparameter.
String2,	En i nuläget oanvänd strängparameter.
Fundindex,	Index för en fundamentalvektor. Om kursdata laddas ignoreras denna.
Rapporttyp,	Används vid fundamentalladdning: 1 - Årsrapport laddas, 2 - Kommuniké laddas, 3 - Prognos laddas.
Fundtidsenhet,	Används vid fundamentalladdning: 1 - Månad, 2 - Kvartal, 3 - År.
Adjust,	Används vid fundamentalladdning: 1 - Adjustfaktorn laddas, 2 - fundamentalvektor laddas (enligt Vektornamn eller Fundindex).
Integer1,	En i nuläget oanvänd heltalsparameter.
Integer2,	En i nuläget oanvänd heltalsparameter.
Felvariabel	Denna parameter är valfri, och kan utelämnas. Om den saknas kommer alla fel som GetData ger upphov till att förvandlas till körningsfel.

Om den finns, däremot, kommer den få ett felvärde enligt följande lista.

0 Inget fel

1 Typfel på Resultat-variabeln.

2 Databasen finns inte.

3 Ett objekt med given vikingkod finns inte.

4 En vektor med givet namn finns inte.

5 Dataladdning misslyckades. Ospecificerat fel.

6 GetData-laddning inte tillgänglig från denna applikation.

7 Aktuell tidsenhet (inte samma som fundtidsenhet) stöds inte vid denna typ av dataladdning. Exempelvis stöds inte laddning av fundamentaldata när man kör på intradaydata.

8 Varken Vektornamn eller FundIndex är givna vid fundamentalladdning.

9 Både Vektornamn och FundIndex är givna vid fundamentalladdning. Endast en i taget får anges.

10 Felaktig RapportTyp given. Den måste vara 1, 2 eller 3.

11 Felaktig Fundtidsenhet given. Den måste vara 1, 2 eller 3.

12 Felaktig Adjust given. Den måste vara 1 eller 2.

13 Givet Vektornamn saknas för Vikingkod vid fundamentalladdning.

14 Givet Fundindex saknas för Vikingkod vid fundamentalladdning.

);

Här följer några exempel på användning av GetData.

```
par(objektlista : set) : realvector;
var i : integer;
    objekt : instrument;
    utvektor : realvector;
begin
```

Ladda data för objekten i en objektlista. Returnera

for i := 0 to len(objektlista) - 1 do
GetData(**objekt**, "PriceData",
objektlista[i], "",
"", "",
0, 0, 0, 0, 0, 0);
utvektor[i] :=
objekt.close[len(objekt.close)-1];
end;
return utvektor;
end;
	en vektor med senaste close-värdet för varje objekt.
	
	Vektornamnet behövs inte anges eftersom data för ett helt objekt laddas.

var rv : realvector;
begin
GetData(**rv**, "PriceData",
objektlista[i], "**close**",
"", "",
0, 0, 0, 0, 0, 0);
end;
	Här är resultatvariabeln en realvector, och då måste vektornamnet ges : "close".

par(objektlista : set) : realvector;
var i : integer;
rv : realvector;
utvektor : realvector;
begin
for i := 0 to len(objektlista) - 1 do
GetData(rv, "**FundData**",
objektlista[i], "**ANST**",
"", "",
0, **1**, **3**, **2**, 0, 0);
rv := fill(rv);
utvektor[i] := rv[len(rv)-1];
end;
return utvektor;
end;
	Ladda fundamentalvektorn med *antal anställda* ("ANST") för objekten i en objektlista. Returnera en vektor med senaste värdet för varje objekt.
	
	Vektornamnet skall alltid skrivas med stora

	bokstäver vid fundamentalladdning. OBS. Olika mycket data laddas upp beroende på kombinationen mellan Rapporttyp och Fundtidsenhet. Man måste vara noggrann när man skriver en funddatamodell och kontrollera att man får upp önskade data.
GetData(rv, "FundData", objektlista[i], "", "", "", **16**, 1, 3, 2, 0, 0);	Ladda fundamentalvektorn med *antal anställda* (fundamentalindex 16). Detta är ett alternativt sätt att ladda en specifik vektor. Se dokumentation för fundamenta

	lanalys för mer informatio n.

par(vikingKod : string) : integer;
var accountType : integer;
begin
GetData(accountType, "**FundInfo**",
vikingKod, "**AccountType**",
"", "",
0, 0, 0, 0, 0 ,0);
return accountType;
end;
	Databasen "FundInfo" används för att ladda upp "AccountTy pe" (Land) eller "Sector" (Bransch).
	
	Resultatet är ett heltal.
	
	Se dokumentat ion för fundamenta lanalys för mer informatio n.

If

If-sats används för villkorlig körning av en följd av satser (kallas även *kodsekvens*). If-satsen måste alltid avslutas med end;. Den kan innehålla en else-del.

if a > b then
c := d;
end;
	Om a är större är b så kommer c att sättas till d.

if undef(objekt.close[0]) then
// Första elementet i close-
// vektorn är odefinierat
else
// Första element i close-
// vektorn är ok.
end;
	Else-delen som körs om villkoret är false.

par(a, b : real) : real;
var villkor : boolean;
begin
  villkor := a > b;
  if villkor then
  return a;
  else
  return b;
  end;
end;
	Variabeln villkor är av typen boolean. En if-sats tar alltid en boolean som parameter.
	
	Denna funktion returnerar det största värdet av a och b.

if a > b then
  if x = y then
  // Gör nåt här
  else
  // Gär nåt annat här
  end;
else
  if x = z then
  // Gör nåt här
  else
  // Gör nåt annat här
  end;
end;
	Ibland behöver man använda en if-sats inuti en annan if-sats.
	
	Det kallas att *nästla* if-satser.

Len

for i := 0 to **len(**objekt.close**)** - 1 do
  // Gör nåt här
end;
 	
for i := 0 to **len(**objektlista**)** - 1 do
  // Gör nåt här
end;
	Använd funktionen len för att ta reda på längden av en vector eller en objektlista. Detta behövs om man skall loopa genom en vektor eller objektlist

	a (= repetera något för alla element).

Loop och exit

Loop-satsen används för att upprepa en kodsekvens. En loop-sats innehåller inget villkor som while- och repeat-satserna. Repetitionen måste avbrytas med ett anrop till Exit-satsen.

Loop-satsen måste avslutas med end;.

i := 50;
loop
rvLoop[i] := 25.3;
i := i + 2;
if i >= 100 then
exit; // Avbryt repetitionen
end;
end;
	Utför samma sak som i exemplet med while-sats ovan:
	
	Vartannat element i rvLoop från och med 50 till och med 98 sätts till 25.3.

loop
while villkor1 do
repeat
exit;
until villkor2;
// Mer kod här
end;
// Mer kod här
end;
	Exit avslutar endast närmaste loop-, while- eller repeat-sats.
	
	I detta fall är det repeat-satsen som omedelbart avslutas.

Minutes

If TimeUnit(timevec) = 0 then
if minutes = 0 then
// Alla tick
	Minutes innehåller tidsintervalle

else
// Tick samlas ihop i intervall om
// minutes minuter.
End;
end;
	t vid körning på intraday. Om Minutes är noll, så körs modellen på samtliga tick.

Now

Konstanten now används för att hitta dagens värde i vektorer. Beroende på tidseneheten innehåller now index för aktuell *tidpunkt*, *dag*, *vecka* eller *månad*.

Now är av typen integer.

dagensClose := objekt.close[now];	Sätt variabeln dagensClose till dagens close-värde i objekt.

RemoveFromSet

Se 0 AddToSet och RemoveFromSet (sida 27).

Repeat

Repeat-satsen används för att upprepa en kodsekvens. Repetitionen kommer att fortsätta *till dess att villkoret är* sant. Den andra skillnaden mot while-satsen är att villkoret för repetitionen ligger efter kodsekvensen. Detta innebär att satserna alltid kommer att utföras minst en gång.

Repeat-satsen måste avslutas med end;.

i := 50; **repeat**	Utför samma sak som i

rvRepeat[i] := 25.3;
i := i + 2;
until i >= 100;
	exemplet med while-sats ovan:
	
	Vartannat element i rvRepeat från och med 50 till och med 98 sätts till 25.3.

Return

Return-satsen avslutar körningen av en funktion. Den finns i två varianter, en med parameter och en utan.

par(objekt : instrument);
begin
return;
// Följande kod körs aldrig
objekt.close := 2;
end;
	Ingen kod efter return kommer att köras.

par(objekt : instrument) : realvector;
begin
return objekt.vol;
end;
	Denna funktion returnerar vol-vektorn i objekt.

par(objekt : instrument);
begin
return objekt.vol;
end;
	Detta ger kompileringsfel eftersom funktionshuvudet inte innehåller en typdeklaration för funktionen.

par(objekt : instrument) : realvector;
begin
return;
end;
	Detta ger kompileringsfel eftersom funktionen enligt funktionshuvudet måste returnera en realvector.

Tilldelning

Tilldelning kopierar värdet av uttrycket till höger till variabeln till vänster. Tilldelningen ”i := x + y” kan utläsas ”Sätt variabeln i till summan av x och y”.

Vid tilldelning mellan vektortyper sker tilldelningen element för element. Satsen

```
vektor1 := vektor2;
```

omvandlas internt till

```
vektor1[0] := vektor2[0];
vektor1[1] := vektor2[1];
vektor1[2] := vektor2[2];
...
vektor1[len(vektor1) - 1] := vektor2[len(vektor1) - 1];
```

Tilldelning kan göras mellan följande typer.

`integer := integer;`	-
`integervector := integervector;`	
`integervector := integer;`	Alla element i vektorn får samma värde.
`real := real;`	-
`real := integer;`	-
`realvector := realvector;`	-
`realvector := integervector;`	
`realvector := real;`	Alla element i vektorn får samma värde.
`realvector := integer;`	Alla element i vektorn får samma värde.
`boolean := boolean;`	-
`booleanvector := booleanvector;`	-

booleanvector := boolean;	Alla element i vektorn får samma värde.
string := string;	-
instrument := instrument;	-
time := time;	-
timevector := timevector;	Fel. Denna typ kan inte användas för parametrar eller variabler. Se 0 time, timevector på sida 3. Justera rubrik och sidnummer.
set := set;	-

Exempel på tilldelningar.

index := 0;	index sätts till 0.
objekt.close := objekt.open;	Innehållet i open-vektorn kopieras till close-vektorn.
invektor[12] := 12.21;	Element 12 i invektor sätts till 12.21.
objekt.close[index] := 0.0;	Givet index i close-vektorn sätt till 0.0.
objektlista[0] := objekt.code;	Det första vikingkoden i objektlista sätts till objektets vikingkod.

Timevec

Timevec är en vektor vars element är av typen Time. Den innehåller tidpunkterna för motsvarande index i övriga vektorer. I den nuvarande versionen av modellspråket kan dock inte Timevec[n] användas för att hitta tidpunkten för det n:te elementet. Istället måste TimeOfIndex-funktionen användas.

Timevec används som parameter i några funktioner, exempelvis IndexOfTime, TimeOfIndex och TimeUnit. Se <std library kapitlet> för dessa funktioner för mer information.

Undef

r := 0/0;
if undef(r) then
// Detta är sant eftersom division
// med noll ger odefinierat resultat.
end;
 	
tid := time("10:00:00");
if undef(tid) then
// Detta är sant eftersom datum
// måste anges för en tidpunkt.
end;
	Använd funktionen undef för att ta reda på om en integer, real eller time är odefinierad.

While

While-satsen används för att upprepa en kodsekvens. Repetitionen kommer att fortsätta *så länge villkoret är sant*. Om villkoret för repetition är falskt första gången kommer kodsekvensen mellan while och end aldrig att köras.

While-satsen måste avslutas med end;.

i := 50;
while i < 100 do
rvWhile[i] := 25.3;
i := i + 2;
end;
	Vartannat element i rvWhile från och med 50 till och med 98 sätts till 25.3.

while 1 < 2 do
// Gör nåt, oändligt många gånger
end;
	Varning! Detta ger en oändlig repetition, som inte kan avbrytas. 1 kommer alltid att vara mindre än 2 så repetitionen kommer aldrig att upphöra.

Operatorer

Precedens och parenteser

Plus och minus är två exempel på operatorer. En operator är en funktion där funktionsnamnet står mellan parametrarna istället för före parametrarna. Man skriver ”12 + x” och inte ”+(12, x)”.

Problemet med detta skrivsätt är att tvetydigheter uppstår. Skall additionen eller multiplikationen utföras först i uttrycket ”x + 5 * 3”? Svaret är att multiplikationen skall utföras först. När ett uttryck beräknas är beräkningsordningen följande:

1. not
2. * / and
3. + - or
4. = <> < <= > >=

Not är högst upp på listan och sägs därför ha högst *precedens*. Operatorer med samma precedens beräknas från vänster till höger.

Observera att uttrycket i satsen

```
if x = 12 and b < 100 then
   ...
end;
```

är syntaktiskt *inkorrekt*. Eftersom and har högst precedens skulle beräkningen börja med ”12 and b” vilket är felaktigt eftersom 12 och b inte är av typen boolean. Man löser problemet med parenteser:

```
if (x = 12) and (b < 100) then
   ...
end;
```

I det första exemplet ”x + 5 * 3” kan man ändra beräkningsordningen så här

```
(x + 5) * 3
```

Nu kommer additionen att utföras först.

Aritmetiska operatorer

För dom aritmetiska operatorerna gäller att om en eller båda av *operanderna* (parametrarna) är odefinierad så blir resultatet odefinierat.

När en eller båda av parametrarna till en operator är vektorer utförs operationen element för element i vektorerna.

Resultatet av division med 0 är odefinierat (kan testas med undef(resultat)).

Unärt minus (minustecken som används för att negera ett uttryck) kan bara förekomma först i ett uttryck eller direkt efter en parentes. Följande är ok ”-2 * 3” men inte ”-2 * -3”. Använd istället ”-2 * (-3)”.

Följande typer och operatorer går att kombinera.

* + -	Resultattyp
integer * + - integer	integer
integer * + - real	real
integer * + - integervector	integervector
integer * + - realvector	realvector
real * + - integer	real
real * + - real	real
real * + - integervector	realvector
real * + - realvector	realvector
integervector * + - integer	integervector
integervector * + - real	realvector
integervector * + - integervector	integervector
integervector * + - realvector	realvector
realvector * + - integer	realvector
realvector * + - real	realvector
realvector * + - integervector	realvector
realvector * + - realvector	realvector

/	Resultattyp
integer / integer	real
integer / real	real
integer / integervector	realvector
integer / realvector	realvector
real / integer	real
real / real	real
real / integervector	realvector
real / realvector	realvector
integervector / integer	realvector
integervector / real	realvector
integervector / integervector	realvector
integervector / realvector	realvector
realvector / integer	realvector
realvector / real	realvector
realvector / integervector	realvector
realvector / realvector	realvector

Boolska operatorer

När en eller båda av parametrarna till en operator är vektorer utförs operationen element för element i vektorerna.

Observera att båda *operanderna* (parametrarna) alltid beräknas, även om resultatet av beräkningen inte påverkas av den andra operandens värde.

Dom boolska operatorer fungerar så här. Givet värdena till vänster på A och B blir resultaten av dom olika operationerna enligt värdena till höger.

A	A and B	A or B	

true		true	true	false
true		false	true	false
false		false	true	true
false		false	false	true

Följande typer och operatorer går att kombinera.

and or	Resultattyp
boolean and or boolean	boolean
boolean and or booleanvector	booleanvector
booleanvector and or boolean	booleanvector
booleanvector and or booleanvector	booleanvector

Relationella operatorer

För dom relationella operatorerna gäller att om en eller båda av *operanderna* (parametrarna) är odefinierad så blir resultatet odefinierat.

När en eller båda av parametrarna till en operator är vektorer utförs operationen element för element i vektorerna.

Följande typer och operatorer går att kombinera.

= <>	Resultattyp
integer = <> integer	boolean

integer = <> real	boolean
integer = <> integervector	booleanvector
integer = <> realvector	booleanvector
real = <> integer	boolean
real = <> real	boolean
real = <> integervector	booleanvector
real = <> realvector	booleanvector
boolean = <> boolean	boolean
boolean = <> booleanvector	booleanvector
integervector = <> integer	booleanvector
integervector = <> real	booleanvector
integervector = <> integervector	booleanvector
integervector = <> realvector	booleanvector
realvector = <> integer	booleanvector
realvector = <> real	booleanvector
realvector = <> integervector	booleanvector
realvector = <> realvector	booleanvector
booleanvector = <> boolean	booleanvector
booleanvector = <> booleanvector	booleanvector
time = <> time	boolean

< <= > >=	**Resultattyp**
integer < <= > >= integer	boolean
integer < <= > >= real	boolean
integer < <= > >= integervector	booleanvector
integer < <= > >= realvector	booleanvector
real < <= > >= integer	boolean
real < <= > >= real	boolean
real < <= > >= integervector	booleanvector
real < <= > >= realvector	booleanvector
integervector < <= > >= integer	booleanvector
integervector < <= > >= real	booleanvector
integervector < <= > >= integervector	booleanvector
integervector < <= > >= realvector	booleanvector
realvector < <= > >= integer	booleanvector
realvector < <= > >= real	booleanvector

realvector < <= > >= integervector	booleanvector
realvector < <= > >= realvector	booleanvector
time < <= > >= time	boolean

Reserverade ord

Följande ord är reserverade för speciella ändamål i modellspråket kan inte användas som variabel- eller funktionsnamn.

addgraphic, addtoset, and, begin, do, else, end, exit, false, for, getdata, if, len, loop, minutes, not, now, or, par, removefromset, repeat, return, then, timevec, to, true, undef, until, var, while

Funktioner till Future Look. Modellspråket som skapar nya möjligheter.

1 Inledning

Här kommer en genomgång av de funktioner som ingår i standardversionen av modellspråket Future Look. Funktionerna används som byggstenar vid skapandet av de modeller, som du vill konstruera.

Först något om strukturen i en modell. Så här ser ”skelettet” till en modell ut.

```
(*text,text,text*)
Par (main : instrument; ... );

Var ... ; // text,text,text

Begin

end;
```

Låt oss nu ta de olika delarna i tur och ordning.

(* *) betyder att programmet skall betrakta det som står inuti som text. Du kan t.ex. att ge dig själv och andra information om en RSI modell genom att skriva (*Det här är en RSI-modell, som ger signaler….. *).

Ett annat sätt att infoga text , givet att texten ryms före radslutet, är att ange två slashar, //, som anger att allt till höger om // fram till slutet av raden är text.

Efter Par anges alla de tidsserier och de parametrar som skall ingå i modellen, t.ex. längden på medelvärden, och dessutom alla de tidsserier, som du vill kunna visa på skärmen. Det gäller både de som skall hämtas från databasen och de som du skapar i modellen.

Sedan har vi var, som anger att de uppräkningar som följer efter denna bara skall användas för mellanberäkningar och aldrig blir aktuella att redovisa i en presentation.

När ovanstående är gjort, så har vi angivit de storheter, som skall ingå i modellen.

Mellan Begin och end; skapar du din modell.

Vi ger nu ett exempel på hur du gör en modell, som skapar ett medelvärde på slutkursen på aktier. Då börjar du med att skriva så här:

```
(*Den här modellen skapar medelvärden. *)
 Par( main : instrument;
MAVlängd : integer;
out Medelvärde : realvector);
```

När man kör en modell så behöver man data till modellen. Det vanliga är att man använder sig av ett objekt, som finns i Vikingens databas, aktie, index, ränta eller råvara. För att använda ett objekt i Vikingens databas så börjar du modellen med main : instrument. Du kan uppfatta det så att du genom att ange main : instrument får en option att använda det aktuella objektets öppnings, högsta, lägsta och slutkurser samt volym i din modell. Med det aktuella objektet avses det objekt, som laddats upp, när du startade Vikingen.

Här kommer en finess i Future Look. Du kan namnge de storheter du använder i modellen i klartext. I det exempel, som vi håller på med så använder vi MAVlängd och Medelvärde.

Datatyperna för storheterna i modellen måste anges. Det gör vi på följande sätt.

```
MAVlängd : integer;
```

Genom att skriva integer anger du att MAVlängd är ett heltal.

Genom att göra så här, så kan du när du kör modellen i modellinställningar ändra längden på medelvärdet.

```
Medelvärde : realvector;
```

Medelvärde : realvector talar om att när du i din modell skriver Medelvärde så avser du en tidsserie, som består av reella tal.

Out i Medelvärde : realvector anger att du vill ha möjligheten att presentera tidsserien ifråga i diagram och/eller tabell.

När man gör en modell, så finns ett stort antal funktioner skapade, som du kan använda dig av. Dessa presenteras senare.

I vårt exempel, så kommer vi att använda oss av MAVN, som är en funktion som ger en tidsserie av icke centrerade medelvärden.

I MAVN funktion anger vi dels et objekt, som det skall skapas ett medelvärde för, dels längden på medelvärdet. I vårt fall ser det ut så här. MAVN(main.close, MAVlängd).

För mer information om MAVN se nedan under MAVN.

Nu skapar vi själva modellen.

```
Begin

Medelvärde := MAVN(main.close, MAVlängd);

end;
```

När du gör modeller använder du dig av tilldelningstecknet :=. Det anger att det som står till vänster om det skall få det värde som finns till höger om det.

I vårt exempel så har vi givit Medelvärde värdet av de beräkningar som ¨MAVN utfört på objektets slutkurser. Hade vi stället för main.close skrivit main.low, så hade Medelvärde beräknats på objektets lägsta kurser.

Här kommer nu den fullständiga modellen för medelvärdesberäkningen .

```
(*Den här modellen skapar medelvärden. *)

Par( main : instrument;
MAVlängd : integer;
out Medelvärde : realvector);

Begin

Medelvärde := MAVN(main.close, MAVlängd);

end;
```

Nu ger vi dig ett ytterligare exempel på en modell där du använder dig av MAVN.

Antag att du vill skapa en oscillatormodell. D.v.s. en modell där du skapar två medelvärden för att sedan ta skillnaden mellan dessa. Oscillatorn är tidsserien av skillnaderna. Det skulle kunna se ut så här

```
(*Oscillator.                                    © Delphi International Company

Version 981203.

Detta är en modell som skapar en oscillator, d.v.s. skillnaden mellan två medelvärden av
längderna MAVkort och MAVlång.*)

Par(main : instrument;
MAVkort, MAVlångt: integer;
out Oscillatorvärde : realvector);

var KortMedelvärde, LångtMedelvärde : realvector;

Begin

KortMedelvärde :=          MAVN(Main.Close, MAVkort);
LångtMedelvärde :=         MAVN(Main.Close, MAVlångt);
Oscillatorvärde :=         KortMedelvärde -LångtMedelvärde;

end;
```

Nu har du redan förstått hur Begin och end används. Begin talar om att nu börjar instruktionerna för beräkningarna, end att nu är det slut.

När vi i Par angivet main : instrumnet så ger vi Vikingen instruktioner att när vi väljer att köra modellen på aktier, ladda upp högsta, lägsta, öppnings och slutkurser samt volymer.

Det innebär inte att du måste använda alla dessa tidsserier i din modell, bara att du har en option att använda dem.

Avslutningsvis mer om parametrar. De vektorer som vi använder är som regel tidsserier och beräkningsmässigt börjar de med index 0. Vektorerna är av tre slag, integervector , realvector och booleanvector. En integervector består av en serie av heltal. En realvector består av en serie av reella tal. En booleanvector består av en serie av Sann och Falsk.

I vissa lägen som t.ex. vid längden på medelvärden och vid jämförelser bakåt i tiden måste du ange datatypen som heltal.Använder du heltal, så har du den fördelen att beräkningarna går snabbare.

Exempel på parametrar är längden på medelvärde, signalnivåerna i RSI och antalet perioder i beräkningen av momentum. Parametrar, som inte är tidsserier, skall definieras som integer, real respektive boolean.

2 En kort genomgång

A *timeseries* X is defined as a limited, ordered sequence of *n* elements $\{x_i\}_{i=1}^{n} = x_1, x_2, \ldots, x_n$

Exempel på tidsserier är slutkurser, högsta och lägsta kurser och volymer. Andra exempel är tidsserier, som du skapar genom att använda modellspråket och dess funktioner, t.ex. ett medelvärde på slutkurserna. Ytterligare exempel är tidsserier, som inte innehåller tal, utan består av elementen Sann (True) och Falsk (False).

A, B, C, D, E, F	are used to denote timeseries with boolean elements. (True och False)
X, Y, Z, U, V, W	are used to denote timeseries with real elements. (T.ex. slutkurser)
K, L, M, N	are used to denote timeseries with integer elements. (T.ex. längden på ett medelvärde. Anledningen att använda Integervectors (tidsserier med heltal) är att beräkningarna går snabbare än om du använder Realvectors (tidsserier med reella tal, flyttal))
X[i],	x_i are used to denote the i:th element in the timeseries X.
lower case letters	are used to denote integer and float values
t1, t2	are used to denote times
#	are used to denote undefined elements in a timeseries in the examples
T	are used to denote a TRUE value in a boolean timeseries in the examples
F	are used to denote a FALSE value in a boolean timeseries in the examples
R0, R1, R2, R3, R4	are used to denote timeseries with float elements (flyttal) in the examples
I0, I1, I2, I3, I4	are used to denote timeseries with integer (heltal) elements in the examples
B0, B1, B2, B3, B4	are used to denote timeseries with boolean elements in the examples lower case letters are used to denote integer and float values in the examples
t1, t2	are used to denote times in the examples

3 Modellspråksfunktioner

Här kommer nu en uppräkning av de olika funktionerna i modellspråket. Funktioner som kan användas som byggelement i skapandet av modeller.

ABS

Syntax

```
X := ABS(Y);
```

Description

The function calculates the absolute value for each element in Y and returns them in X.

ABS använder du t.ex. när du vill beräkna slutkursens volatilitet genom att jämföra dagens kurs med gårdagens kurs. Du vill då ha nedgångarna med positiva tecken och det får du genom att använda ABS.

$$\mathbf{X} = \{|y_i|\}_{i=1}^{n}$$

Example

```
R0                10.0        -10.0        #          -8.5        0.0
R1 := ABS(R0);
R1                10.0        10.0         #          8.5         0.0
```

Modellexempel

```
(*Volatilitetsberäkning

Version 981203

Detta är en modell för beräkning av slutkursens  volatilitet.
Differns är lika med skillnaden mellan dagens slutkurs och gårdagens.
SummaDifferens är lika med  slutkursensens förändring under  AntalPerioder dagar.
Volatilitet är absolutbeloppet av slutkursens förändring mellan idag och igår.
SummaVolatilitet är summan av  volatiliteterna under  AntalPerioder dagar.*)

Par (main : instrument;
AntalPerioder : integer;
out SummaDifferens , Volatilitet, SummaVolatilitet : relavector): ;

var Differens: realvector;



Begin

Differens :=  Main.Close - SHIFT(Main.Close, 1);
SummaDifferens :=              SUM(Differens, AntalPerioder);
Volatilitet :=  ABS(Differens);
SummaVolatilitet :=            SUM(Volatilitet, AntalPerioder);

end;
```

ACCUM

Syntax

```
Z := ACCUM(A, X, Y);
```

Description

The function calculates the accumulated sum of the elements from X and Y. If A[i] is true the value of X[i] is added to the sum, if A[i] is false the value of Y[i] is added. Any undefined values in X or Y is treated as 0.

$$\mathbf{Z} = \left\{ \sum_{k=1}^{i} f(a_k, x_k, y_k) \right\}_{i=1}^{n}, \text{ where } f(a,x,y) = \begin{cases} x, & \text{if } a \text{ is true} \\ y, & \text{if } a \text{ is false} \end{cases}$$

ACCUM kan användas t.ex. för att ta fram förändringen mellan startdag och slutdag för en kurs.

Example

R0	-1.0	-2.0	-3.0	-4.0	-5.0
R1	1.0	2.0	#	4.0	5.0
B0	F	T	T	F	T

R2 := ACCUM(B0, R0, R1);

R2	1.00	-1.00	-4.00	0.00	-5.00

Modellexempel

```
(*Kursförändring                                    © Delphi Internatioanl Company

Version 981203

Detta är en modell som gör det möjligt att beräkna a) kursförändringen
mellan idag och igår b) få en serie som visar vilka dagar kursen gått upp
respektive ner och c) NerUppfrekvensen uttryckt som ett procental för kvoten
mellan nettoantalet uppdagar och summa dagar.*)

par(Main : instrument;
out Kursförändring, UppNerfrekvens, NerUpp : realvector);

var PositivKursförändring, NegativKursförändring : booleanvector;
Uppdag, Nerdag : realvector;

begin

Kursförändring :=          Main.Close - SHIFT(Main.Close, 1);
PositivKursförändring :=   Kursförändring >= 0;
Uppdag :=    ONE(PositivKursförändring);
NegativKursförändring :=   Kursförändring <= 0;
Nerdag :=    ONE(NegativKursförändring);
NerUpp :=    Uppdag - Nerdag;
UppNerfrekvens :=          (100*ACCUM(PositivKursförändring, Uppdag, - Nerdag))/
ACCUM(PositivKursförändring, Uppdag, Nerdag);

end;
```

ALT

Syntax

```
Z := ALT(A, X, Y);
```

Description

The function copies elements from X and Y to Z. From which timeseries the element is copied is decided by the boolean values in A. If A[i] is true X[i] is copied to Z[i], if A[i] is false Y[i] is copied.

$$\mathbf{Z} = \left\{ f\left(a_i, x_i, y_i\right) \right\}_{i=1}^{n}, \text{ where } f\left(a,x,y\right) = \begin{cases} x, & \text{if } a \text{ is true} \\ y, & \text{if } a \text{ is false} \end{cases}$$

ALT använder du t.ex. om du vill skapa en tidsserie som endast innehåller kursdifferenser för de dagar då kursen har varit stigande.

Example

```
R2 := ALT(B0, R0, R1);
```

B0	F	T	F	F	T
R0	-1.0	-2.0	-3.0	-4.0	-5.0
R1	1.0	2.0	#	4.0	5.0
R2	1.0	-2.0	#	4.0	-5.0

Observera att om det ena eller båda av x och y är odefinierade, så blir Z (i exemplet ovan R2) odefinierat.

Om du vill undvika att få in odefinierade värden, så kan du använda dig av funktionen FILL. FILL fyller ut odefinierade värden i en tidsserie med närmast föregående värde. Du kan läsa mera om FILL nedan.

Modellexempel

```
(*StigandeKurser                                        © Delphi International
Company

Version 981203.

Detta är en modell som skapar en tidsserie som endast innehåller kursdifferenser
för de dagar då kursen har varit stigande.*)

Par(Main : instrument;
out EndastPositivaKursförändringar :  realvector);

var
PositivDIff : booleanvector;
Kursdifferens : realvector;

Begin

Kursdifferens :=              Main.Close - SHIFT(Main.Close, 1);
PositivDiff :=                Kursdifferens >= 0;
EndastPositivaKursförändringar := ALT( PositivDiff, Kursdifferens,Const(0.0));

end;
```

BETA

Syntax

Z := BETA(Y, X, p);

Description

The function calculates the beta-values β for the regression line y = α + β(x - m) where m is the mean of x over the period *p* and returns them in Z.

Y can for example be an index and X a stock price.

The beta-value for the regression line

$$y = a + b\ (x - \bar{x}), \text{ where } \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

is calculated as follows

$$S_{xy} = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}), \text{ where } \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \text{ and } \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

$$S_{xx} = \sum_{i=1}^{n}(x_i - \bar{x})^2, \text{ where } \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$b = \frac{S_{xy}}{S_{xx}} = \left(n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i\right) \Big/ \left(n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2\right)$$

which gives

$$\mathbf{Z} = \left\{\left(p\sum_{k=i-p+1}^{i} x_k y_k - \sum_{k=i-p+1}^{i} x_k \sum_{k=i-p+1}^{i} y_k\right) \Big/ \left(p\sum_{k=i-p+1}^{i} x_k^2 - \left(\sum_{k=i-p+1}^{i} x_k\right)^2\right)\right\}_{i=p}^{n}$$

Example

```
R0          1.0     2.0     3.0     4.0     #       6.0
  7.0       8.0     9.0     10.0
R1          10.0    20.0    30.0    40.0    50.0    60.0
  65.0      70.0    75.0    80.0

R2 := BETA(R0, R1, 4);

R2                                  10.00    10.00   10.00
  8.57      5.00    5.00    5.00
```

BMAX

Syntax

```
Y := BMAX(X, A);
```

Description

The function finds the local maximum from the signals in A.

Du vill t.ex. ta reda på det högsta värdet från senaste köpdag. I modellspråket finns funktionerna BUY och SELL. De kan användas för att beräkna resultaten av olika strategier med hjälp av den Vinsttest som är inbyggd i Vikingen.

Example

Tid		Y:=BMAX(**X**,**A**)	**X**		**A**
1998-08-10	218.50		218.5000	Sann	
1998-08-11	219		219		Falsk
1998-08-12	222.50		222.5000	Falsk	
1998-08-13	222.50		220		Falsk
1998-08-14	222.50		222.5000	Falsk	
1998-08-17	222.50		221.5000	Falsk	
1998-08-18	229		229		Falsk
1998-08-19	229		224.5000	Falsk	
1998-08-20	229		220		Falsk
1998-08-21	213.50		213.5000	Sann	
1998-08-24	213.50		206.5000	Falsk	
1998-08-25	213.50		207.5000	Falsk	

Modellexempel

```
(*MaxvärdeFrånKöp              © Delphi International Company

Version 981203

Denna modell ger dig högsta värdet på slutkursen från köptillfället.
Som köpvillkor har satts att slutkursen måste överstiga sitt  MAVlängd
långa medelvärde. Som säljvillkor har satts att slutkursen måste
understiga sitt MAVlängd långa medelvärde.*)

Par(main :instrument;
MAVlängd : integer;
out Medelvärde, Maxkurs :  realvector;
out BUY, SELL : booleanvector);

var Köp, Sälj, b1 :  booleanvector;
Undefvec : realvector;

Begin

Medelvärde :=                  MAVN( Main.Close, MAVlängd);
Köp :=                         Main.Close > Medelvärde;
Sälj :=                        Main.Close <= Medelvärde;
(*Nedan används FILTERBUY och FILTERSELL funktionerna som sorterar bort
allt utom "första" köpsignal och "första" säljsignal.*)
BUY :=                         FILTERBUY(Köp, Sälj);
SELL :=                        FILTERSELL(Köp, Sälj);
MaxKurs :=  BMAX(Main.Close, BUY);
(*Det som följer här nedan gör att vi bara får maxkurserna i köplägen
Undefvec är en vektor som är tom och som skall gälla så snart vi inte ligger
i köp. Att vi lägger till NOT  SELL beror på att vi annars skulle fått värden för
MaxKurs de dagar som säljsignalerna givits.*)
b1 :=          TOPD(ONE(BUY), 1) < TOPD(ONE(SELL), 1);
MaxKurs :=  ALT((b1 AND NOT  SELL), MaxKurs, Undefvec);

end;
```

BMAXD

Syntax

```
Y := BMAXD(X, A);
```

Description

The function finds the number of days from the last local maximal value in X with respect to the signals in A.

Här kan du t.ex. få reda på avståndet till senaste lokala maximum efter en köpsignal.

Example

Tid		**Y** := BMAXD(**X**, **A**);	**X**	**A**
1998-08-10	0		218.50	Sann
1998-08-11	0		219	Falsk
1998-08-12	0		222.50	Falsk
1998-08-13	1		220	Falsk
1998-08-14	0		222.50	Falsk
1998-08-17	1		221.50	Falsk
1998-08-18	0		229	Falsk
1998-08-19	1		224.50	Falsk
1998-08-20	2		220	Falsk
1998-08-21	0		213.50	Sann
1998-08-24	1		206.50	Falsk
1998-08-25	2		207.50	Falsk

Modellexempel

```
(*Avstånd                                   © Delphi International Company


Version 981203

I den här modellen beräknas avståendet från det lokala maximivärdet efter senaste köpsignal.
Som villkor för köp krävs i modellen att slutkursen skall vara högre än medelvärdet för de
senaste MAVlängd perioderna. *)


Par(main :instrument;

MAVlängd : integer;

out Avstånd : realvector);


Var

Köp, Sälj : booleanvector;

Köpsignal : integervector;


Begin


Köp :=                  Main.Close > MAVN( Main.Close, MAVlängd);

Sälj :=                 Main.Close <= MAVN(Main.Close, MAVlängd);

Köp :=                  FILTERBUY(Köp, Sälj); (* FILTERBUY rensar bort alla
köpsignaler utom den första efter en säljsignal. Vill du rensa bort alla icke unika säljsignaler, så
använder du dig av FILTERSELL. *)

Köpsignal :=            ONE(Köp); (*ONE funktionen skapar en tidsserie med 1 för
unika köpsignaler och i övrigt innehåller tidsserien nollor.


Avstånd :=              BMAXD(Main.Close, Köpsignal);


end;
```

BMIN

Se BMAX ovan.

BMIND

Syntax

Same as the equivalent 'max'-functions

Description

The two 'min'-functions performes the same calculations as BMAX and BMAXD above but minimum values are returned.

Se BMAXD ovan.

BOT

Syntax

```
Y := BOT(X, p);
```

Description

Y[i] = the *p*:th local minimum extreme value of X before X[i].

X[i] is a local minimum extreme value if X[i-1] > X[i] and X[i] <= X[i+1].

BOT använder du dig av för att få storleken på någon av de senaste bottnarna för t.ex. slutkursen.

Example

```
R0          3.0     2.0     2.0     4.0     1.0     2.0
R1 := BOT(R0, 1);
R1                          #       2.0     2.0     1.0
```

Modellexempel

```
(*Bottenkurs                    © Delphi International Company

Version 981203.

Här får du storleken på en bottenkurs. Du kan välja vilken bottenkurs bakåt i tiden
som du vill ha uppgift om genom att  angevärdet på AntalBottnarTillbaka.*)

PAR(Main : instrument;
AntalBottnarTillbaka :  integer;
out Bottenkurs :  realvector);

Begin
Bottenkurs := BOT( Main.Close, AntalBottnarTillbaka);
end;
```

BOTD

Syntax

```
Y := BOTD(X, p);
```

Description

Y[i] = the distance to the *p*:th local minimum extreme value of X before X[i].

X[i] is a local minimum extreme value if X[i-1] > X[i] and X[i] <= X[i+1].

BOTD ger dig avståndet till någon av de senaste bottnarna för en valfri tidsserie.

Example

```
R0          3.0     2.0     2.0     4.0     1.0     2.0
R1 := BOTD(R0, 1);
R1                          #       1.0     2.0     1.0
```

Modellexempel

```
(*BottenavståndRSI                                    © Delphi International Company

Version 981203

I den här modellen beräknas avståndet till den  RSI botten, som ligger
AntalBottnarTillbaka.*)

Par(main : instrument;
RSIlängd, AntalBottnarTillbaka :  integer;
out AvståndTillRSIbotten, RSIvärde : realvector);

var

Begin

RSIvärde := RSI1( Main.Close, RSIlängd);

AvståndTillRSIbotten :=  BOTD(RSIVärde, AntalBottnarTillbaka);

end;
```

CONST

Syntax

X := CONST(y);

Description

Returns a constant timeseries with float elements which all has the value *y*. *See also FLOATV*.

$$\mathbf{X} = \{y\}_{i=1}^{n}$$

Example

```
R0 := CONST(42.0);
```

R0	42.0	42.0	42.0	42.0	42.0	42.0

COPPOCK

Syntax

```
Y := COPPOCK(X, a, b, c);
```

Description

The function calculates the Coppock indicator for X where *a* is the length of the moving average, *b* is the number of periods to base changes on and *c* is the period to sum the differences for.

I ”Snabba aktievinster” kan du läsa om Coppockmodellen och också om ytterligare ett antal vanligen använda tekniska modeller.

Example

```
R0          100         105         110         105         102         110
  130       135         110         112

R1 := COPPOCK(R0, 2, 2, 2);

R1                                                          0.06        -0.09
  0.13      0.57        0.52        -0.12
```

DiffFromSignal

Syntax

```
X := DiffFromSignal(A,Y);
```

Description

The function calculates how much the Price data in Y has gone up or down (in %) with respect to the signals in the vector A.

Filterbuy och Filtersell används för att filtrera bort alla sanna värden för de logiska tidsserier som svarar mot behåll (hold) respektive icke behåll (stay out) andra än de första sanna värdena efter ett sant värde av motsatt slag.

Example

The vectors SELL and BUY is constructed with FilterBuy and FilterSell.

```
Y is a filled vector with last price

SELL:       T           F           F           F           F           F
  F         T           F
BUY:                    F           F           F           F           T
  F         F           F           F
Y:                      3           4           5           5.5         6
  3         4           7           8

X := DiffFromSignal(BUY or SELL, Y);

X :=                                            33.3        66.6        83.3
  -50       33.3                                14.3
```

EXP

Syntax

```
Y := EXP(X);
```

Description

The function calculates the exponential of each element in X and returns them in Y.

$$\mathbf{Y} = \left\{ e^{x_i} \right\}_{i=1}^{n}$$

Example

```
R0          0.0     1.0     #       0.1     0.01
R1 := EXP(R0);
R1          1.0     2.7     #       1.11    1.01
```

FILL

Syntax

```
Y := FILL(X);
```

Description

Y[i] = the same as X[i], but with undefined values replaced with the previous defined value.

$$\mathbf{Y}[i] = \begin{cases} \mathbf{X}[i] & \text{, if } \mathbf{X}[i] \text{ is defined} \\ \mathbf{Y}[i-1] & \text{, if } \mathbf{X}[i] \text{ is undefined} \end{cases} \text{, for } i = 1,\ldots,n$$

FILL är utmärkt att använda då du arbetar med tidsserier där det saknas data för vissa dagar, t.ex. för kurser för bolag med låg omsättning. FILL fyller då ut tidsserien så att du har en komplett tidsserie. Att använda FILL i en sådan här situation är detsamma som att säga att så länge som ny information inte finns tillgänglig, så antar man status quo gäller.

Example

```
R0          #       1.0     #       #       0.01    #
```

```
R1 := FILL(R0);
R1          #          1.0          1.0          1.0          0.01          0.01
```

FILTERBUY

Syntax

```
C := FILTERBUY(A, B);
```

Description

The function calculates a new buy timeseries A from the buy timeseries B and the sell timeseries C where all unnecessary buy signals in the buy timeseries are removed. All buy signals that occur after a buy signal, but before any sell signal will be removed.

Example

```
B0          F          T          T          F          T          F
B1          F          F          F          T          T          F
B2 := FILTERBUY(B0, B1);
B2          F          T          F          F          T          F
```

FILTERSELL

Syntax

```
C := FILTERSELL(A, B);
```

Description

The function calculates a new sell timeseries C from the buy timeseries A and the sell timeseries B where all unnecessary sell signals in the sell timeseries are removed. All sell signals that occur before any buy signal or after a sell signal, but before any buy signal will be removed.

Example

```
B0          F          T          T          F          T          F
B1          T          F          T          T          T          T
B2 := FILTERSELL(B0, B1);
B2          F          F          T          F          F          T
```

GetBuySellAge

Syntax

```
i := GetBuySellAge(A, B, X, Y);
```

Description

The function calculates the number of tradingdays from a signal and fills the vectors X and Y with the result. The return value is a dummy required only by internally constructions.

Example

```
A:          F          T          F          F          F          F
  F         T          F
B:          F          F          F          F          T          F
  F         F          F

i := GetBuySellAge(BUY, SELL, X, Y);

X:          #          0          1          2          #          #
  #         0          1
Y:          #          #          #          #          0          1
  2         #          #
```

GROWTHANN

Syntax

```
Y := GROWTHANN(X, y, p);
```

Description

The function calculates the annual growth rate in percentage, y is the length of the year in timeunits and p is the period for the change.

$$\mathbf{Y} = \left\{ \sqrt{\frac{t}{p-1} \sum_{k=i-p+1}^{i} \left(\ln\left(\frac{x_k}{x_{k-1}}\right) - \frac{1}{p} \sum_{v=i-p+1}^{i} \ln\left(\frac{x_v}{x_{v-1}}\right) \right)^2} \right\}_{i=p+1}^{n}$$

Example

```
R0          102.00     103.00     102.50     102.50     101.00
  101.50    102.00     102.00     103.00     102.50

R1 := GROWTHANN(R0, 252, 5); (*  one week annual growthrate *)

R1                                                              -38.90
  -38.60    0.00       62.87      63.66
```

GROWTHSM

Syntax

```
Y := GROWTHSM(X, y, p);
```

Description

The function calculates the smoothed annual growth rate in percentage, y is the length of the year in timeunits and p is the period for the change.

$$\mathbf{Y} = \left\{ \sqrt{\frac{t}{p-1} \sum_{k=i-p+1}^{i} \left(\ln\left(\frac{x_k}{x_{k-1}}\right) - \frac{1}{p} \sum_{v=i-p+1}^{i} \ln\left(\frac{x_v}{x_{v-1}}\right) \right)^2} \right\}_{i=p+1}^{n}$$

LOG

Syntax

```
Y := LOG(X);
```

Description

The function calculates the natural logarithm for each element in X and returns them in Y.

$$\mathbf{Y} = \left\{ \log\left(x_i\right) \right\}_{i=1}^{n}$$

Example

R0	2.71	10.0	#	100.0	-5.0
R1 := LOG(R0);					
R1	1.0	2.3	#	4.61	#

MAVC

Syntax

```
Y := MAVC(X, p);
```

Description

The function calculates a centered moving average of length *p* for the timeseries X and returns it in Y.

$$\mathbf{Y} = \left\{ \frac{1}{p} \sum_{k=i-p+1}^{i} x_k \right\}_{i=p}^{n}$$

Detta är en sällan använd funktion i teknisk analys där det i stället regelmässigt används medelvärden som är icke centrerade vilket innebär att medelvärdena lägga på den sista dagen i stället för på den mittersta dagen.

Example

```
R0              1               2               5               #               2               3
  4             1               2               3

R1 := MAVC(R0, 3);

R1              #               2.67            3.50            3.50            2.50            3.00
  2.67          2.33            2.00            #
```

MAVN

Syntax

```
Y := MAVN(X, p);
```

Description

The function calculates a normal moving average of length *p* for the timeseries X and returns it in Y.

$$\mathbf{Y} = \left\{ \frac{1}{p} \sum_{k=i-p+1}^{i} x_k \right\}_{i=p}^{n}$$

Example

```
R0              1               2               5               #               2               3
  4             1               2               3

R1 := MAVN(R0, 3);

R1                                              2.67            3.50            3.50            2.50
  3.00          2.67            2.33            2.00
```

Här beräknas en ”vanligt” medelvärde, d.v.s. ett oviktat medelvärde där varje värde har samma vikt. Medelvärdet placeras på sista dagen för de värden som ingår i beräkningen av medelvärdet.

Modellexempel

```
*Medelvärden

Version 981203

Här beräknas två stycken medelvärden med längderna  KortMAV och LångtMAV.*)

Par(main : instrument;
KortMAV, LångtMAV : integer;
out KortMedelvärde, LångtMedelvärde : realvector);

var

Begin

KortMedelvärde :=          MAVN(Main.Close, KortMAV);
LångtMedelvärde :=         MAVN(Main.Close, LångtMAV);

end;
```

MAVW

Syntax

Y := MAVW(X, p);

Description

The function calculates a weighted moving average of length *p* for X and returns it in Y.

$$\mathbf{Y} = \left\{ \frac{2}{p(p+1)} \sum_{k=i-p+1}^{i} k x_k \right\}_{i=p}^{n}$$

Det här medelvärdet är ett viktat, icke centrerat medelvärde, där de ingående vikternas värde ökar successivt.

Example

```
R0              1               2               5               #               2               3
  4             1               2               3

R1 := MAVW(R0, 3);

R1                                              3.00            4.00            2.00            2.00
  3.00          2.00            2.00            2.00
```

MAVX

Syntax

Y := MAVX(X, p);

Description

The function calculates an exponential moving average of length *p* for X and returns it in Y.

$$y_p = \frac{1}{p}\sum_{i=1}^{p} x_i$$

$$y_k = (1-r)y_{k-1} + rx_k, \text{ where } r = \begin{cases} 1 & \text{if } p = 1 \\ p/2 & \text{if } p > 1 \end{cases}, \text{ for } k = p+1,\ldots,n$$

Här är det icke centrerade medelvärdets vikter exponentiellt vägda.

Example

```
R0              1               2               5               #               2               3
  4             1               2               3

R1 := MAVX(R0, 3);

R1                                              2.67            0.89            1.63            2.54
  3.51          1.84            1.95            2.65
```

MAX

Syntax

Y := MAX(X, a);

Description

The function calculates Y where every element is the maximum of the last *a* elements in X.

Den här funktionen använder du när du vill ta fram ett högsta värde för t.ex. slutkurserna under ett antal dagar.

Example

```
R0              1           2           5           #           -2          -3
  -4            1           2           3

R1 := MAX(R0, 3);

R1                                      5           5           5           -2
  -2            1           2           3
```

Modellexempel

```
(*Maxkurs

Version 981203

Här beräknas den högsta kursen för ett antal,  AntalDagarTillbaka, dagar.*)

Par(Main: instrument;
AntalDagarTillbaka :  integer;
out HögstaKurs : realvector);
var

Begin

HögstaKurs :=                           Max(Main.Close, AntalDagarTillbaka);

end;
```

MAXD

Syntax

Y := MAXD(X, a);

Description

The function calculates Y where every element is the distance to the maximum of the last *a* elements in X.

Här får du avstående till den högsta kursen under de antal dagar som du anger.

Maxavstånd := MAXD(Main.close, 10); ger dig avståndet till maxvärdet för de senaste 10 börsdagarna.

Example

```
R0              1              2              5              #              -2             -3
  -4            1              2              3

R1 := MAXD(R0, 3);

R1                                            0              1              2              1
  2             0              0              0
```

MIN

Syntax

Y := MIN(X, a);

Description

The function calculates Y where every element is the minimum of the last *a* elements in Y.

Fungerar som MAX ovan.

Example

```
R0              1              2              5              #              -2             -3
  -4            1              2              3

R1 := MIN(R0, 3);

R1                                            1              2              .2             -3
  -4            -4             -4             1
```

MIND

Syntax

Y := MIND(X, a);

Description

The function calculates Y where every element is the distance to the minimum of the last *a* elements in X.

Fungerar som MAXD ovan.

Example

```
R0              1              2              5              #              -2             -3
  -4            1              2              3

R1 := MIND(R0, 3);

R1                                            2              2              0              0
  0             1              2              2
```

MOMABS

Syntax

```
Y := MOMABS(X, p);
```

Description

The function calculates the absolute momentum of length *p* for X and returns it in Y.

$$\mathbf{Y} = \left\{ x_i - x_{i-p+1} \right\}_{i=p}^{n}$$

Antag att du vill beräkna skillnaden mellan kursen idag och fem dagar tillbaka för att få en uppfattning om volatiliteten för slutkurserna. Antag vidare att du vill beräkna ett medelvärde för volatiliteten. Då vill du inte att skillnaderna skall kunna vara negativa och det är här som MOMABS kan användas.

Example

```
R0              1              2              5              #              -2             -3
  -4            1              2              3

R1 := MOMABS(R0, 3);

R1                                            4              #              -7             #
  -2            4              6              2
```

MOMREL

Syntax

```
Y := MOMREL(X, p);
```

Description

The function calculates the relative momentum of length *p* for X and returns it in Y.

$$\mathbf{Y} = \left\{ 100 \left(\frac{x_i}{x_{i-p+1}} \right) - 1 \right\}_{i=p}^{n}$$

Ibland är du mer betjänt av att använda den relativa förändringen i en kurs uttryckt i procent än den absoluta. Då använder du MOMREL.

Example

```
R0              1               2               5               #               -2              -3
  -4            1               2               3

R1 := MOMREL(R0, 3);

R1                                              400.00          #               -140.00         #
  100.00        -133.33         -150.00         200.00
```

NORM

Syntax

Z := NORM(X, Y, a);

Description

If *a* is non-zero the function computes a normalization of X to Y, i e Z shows how X would have looked like if it had the same relative development as Y. If *a* is zero the funtion copies Y to Z.

$Z[1] = x_1$

$$\mathbf{Z} = \left\{ z_{i-1} \left(\frac{y_i}{y_{i-1}} \right) \right\}_{i=2}^{n}$$

Example 1

```
R0              100             200             300             200             200             100
  0             100             300             100
R1              10              20              30              20              20              10
  0             10              30              10

R2 := NORM(R0, R1, 1);

R2              100             200             300             200             200             100
  0             0               0               0
```

Example 2

```
R0              100             200             300             200             200             100
  #             100             300             100
R1              10              40              60              40              40              20
  #             5               15              5

R2 := NORM(R0, R1, 1);

R2              100             400             600             400             400             200
  200           200             600             200
```

Observera att jämförelserna görs mellan två på varandra följande data och inte med avseende på serierna första data.

NTREND, see TREND

ONE

Syntax

```
Y := ONE(A);
```

Description

The function converts a boolean timeseries A to a numerical timeseries Y. All true values in A are converted to 1 in Y and all false values to 0.

$$y_i = \begin{cases} 1 & \text{if } a_i \text{ is true} \\ 0 & \text{if } a_i \text{ is false} \end{cases}, \text{ for } i = 1,\ldots,n$$

ONE är en mycket värdefull funktion. Antag t.ex. att du angivit villkoren för BUY och Sell som ju är boolska storheter och att du vill i en presentation visa dessa i staplar innehållande +1, 0 och -1 beroende på om det föreligger köpläge, neutralläge eller säljläge. Då kan du göra som i modellexemplet nedan.

Example

```
B0          T          F          T          T          F          F
  F         T          F

R0 := ONE(R0);

R0          1.0        0.0        1.0        1.0        0.0        0.0
  0.0       1.0        0.0
```

Modellexempel

```
(*Signallista
Version 981203
Här beräknas en tidsserie, Position, innehållande 1 för unika köp, -1 för
unika sälj och i övrigt 0.
Den här modellen kan med fördel köras på en lista av aktier och tas ut i
form av en samlingstabell.*)

Par(Main : instrument;
out Position : realvector;
out BUY, SELL, Köp, Sälj : booleanvector);

var b1, b2 : booleanvector;



Begin

(*Här har lagts in två boolska vektorer för att visa
att du om du vill kan använda kortnamn som b1, b2 etc *)
b1 :=        Main.Close > MAVN(Main.Close, 5);
b2 :=        Main.Close < MAVN(Main.Close, 5);
//Som du ser ovan går det bra att också ge en konstant som längd på medelvärdet.
Köp :=                    FILTERBUY(b1, b2);
Sälj :=                   FILTERSELL(b1, b2);
Position :=   ONE(Köp) - ONE(Sälj);
(*Nedan tilldelas BUY Köp och SEll Sälj. Det har gjorts för att få tillfälle
visa att om du inte använder dig av BUYoch SELL så kan du i samband med presentationerna
inte få gröna och röda markeringar av signallägena. Du kan övertyga dig om det genom
att markera bort de två närmaste raderna här under.*)
BUY :=       Köp;
SELL :=      Sälj;

end;
```

OSC

Syntax

```
Y := OSC(X, l, s);
```

Description

The function calculates an oscillator timeseries, i e the difference of a (longer) moving average for X with length *l* and a (shorter) moving average for X with length *s*.

$$\mathbf{Y} = \left\{ \frac{1}{l} \sum_{k=i-l+1}^{i} x_k - \frac{1}{s} \sum_{k=i-s+1}^{i} x_k \right\}_{i=l}^{n}$$

Example

```
R0                0          1          2          4          3          2
  0               3          6          8          5

R1 := OSC(R0, 6, 3);

R1                                                                      -1.00
  0.33          0.67       0.00      -2.00      -2.33
```

Modellexempel

```
(*Oscillatormodell

Version 981203

Här används OSC för att skapa köp- och säljsignaler. Funktionen OSC skapar
en tidsserie innehållande skillnaden mellan två medelvärden, i det här fallet
på slutkurser. Signalkurvan är här en tidsserie med 1 för unika köp och -1
för unika sälj och i övrigt nollor.*)

Par(Main : instrument;
LångtMAV, KortMAV : integer;
out UnikaKöp, UnikaSälj :booleanvector;
out Signalkurvan : realvector);

var BUY, SELL : booleanvector;

Begin

BUY := OSC(Main.Close, LångtMAV, KortMAV) > 0;
SELL := OSC(Main.Close, LångtMAV, KortMAV) < 0;
UnikaKöp := FILTERBUY(BUY,SELL);
UnikaSälj := FILTERSELL(BUY, SELL);
Signalkurvan := ONE(UnikaKöp) - ONE(UnikaSälj);

end;
```

PARAB3

Syntax

```
Z := PARAB3(L, H, a, b, c, X, Y);
```

Description

The function calculates the limits for going short, X, and going long, Y, according to the parabolic trading model. The parabolic growth rate starts at a and increments with b for each day until it reach its maximum c. Z is the composition of X and Y. See also TPARAB.

Det här är en funktion för parabolic metoden. Du rekommenderas studera uppbyggnaden av en sådan modell, som du finner bland Delphis Standardmodeller.

PTREND, see TREND

REGC

Syntax

```
Y := REGC(X, p);
```

Description

The function calculates the beta-values β for the regression line y = α + β(x - m) where m is the mean of x over the period *p* and returns them in Y.

Här har du möjlighet att skapa dig en uppfattning om den rådande trenden som kan vara lika så god om inte bättre än den du får genom att använda medelvärde, toppar och bottnar, trendlinjer eller Delphis Smooth funktion.

Trendriktningen får du t.ex. ur Trendriktning := REGC(Main.Close, 20); trenden för de senaste 20 börsdagarna.

The beta-value for the regression line

$$y = a + b\ (x - \bar{x}), \text{ where } \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

is calculated as follows

$$S_{xy} = \sum_{i=1}^{n}\left(x_i - \bar{x}\right)\left(y_i - \bar{y}\right), \text{ where } \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \text{ and } \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

$$S_{xx} = \sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2, \text{ where } \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$b = \frac{S_{xy}}{S_{xx}} = \left(n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i\right) \Big/ \left(n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2\right)$$

which gives

$$\mathbf{Y} = \left\{ \left(p \sum_{k=i-p+1}^{i} k x_k - \sum_{k=i-p+1}^{i} k \sum_{k=i-p+1}^{i} x_k \right) \Big/ \left(p \sum_{k=i-p+1}^{i} k^2 - \left(\sum_{k=i-p+1}^{i} k \right)^2 \right) \right\}_{i=p}^{n}$$

Example

```
R0          10.0        20.0        30.0        40.0        #           #
  70.0      80.0        90.0        100.0

R1 := REGC(R0, 4);

R1                                                1.0         1.0         1.0
  1.0       1.0         1.0         1.0
```

REGR

Syntax

```
Z := REGR(X, Y, p, U, V,);
```

Description

The function calculates the linear regression of Y relative X over a period p at every time. It also returns the regressionkoefficient in V and its displacement at x=0 in U.

Example

```
R0          10.0        20.0        30.0        40.0        50.0        60.0
  65.0      70.0        75.0        80.0
R1          1.0         2.0         3.0         4.0         #           6.0
  7.0       8.0         9.0         10.0

R2 := REGR(R0, R1, 4, R3, R4);

R2                                                4.00        5.00        6.00
  7.00      8.00        9.00        10.00
R3                                                0.0         0.0         0.0
  0.0       0.0         0.0         0.0
R4                                                0.1         0.1         0.1
  0.1       0.1         0.1         0.1
```

REL

Syntax

```
Z := REL(X, Y);
```

Description

The function calculates a timeseries with X:s development relative Y:s, i e a timeseries that starts at 1.0 and shows how X develops relative to Y.

$$\mathbf{Z} = \left\{ \frac{y_1}{x_1} \cdot \frac{x_i}{y_i} \right\}_{i=1}^{n}$$

Du kan t.ex. använda REL för att få en uppfattning om hur en kurs har utvecklats i förhållande till ett index. Exempel. RelativUtveckling := REL(Main.Close, Index); där du i par deklarerat Par(Index : instrument)….

Observera att startdagen är första dagen i tidsserien till skillnad från REL CHNG nedan.

Example 1

R0	100	200	300	200	200	100
0	100	300	100			
R1	10	20	30	20	20	10
0	10	30	10			

```
R2 := REL(R0, R1);
```

R2	1.00	1.00	1.00	1.00	1.00	1.00
#	1.00	1.00	1.00			

Example 2

R0	100	200	300	200	200	100
0	100	300	100			
R1	10	40	60	40	40	20
0	5	15	5			

```
R2 := REL(R0, R1);
```

R2	1.00	0.50	0.50	0.50	0.50	0.50
#	2.00	2.00	2.00			

RELCHNG

Syntax

Y := RELCHNG(X, **p);**

Description

The function calculates the relative change in percent of X over a period p.

$$\mathbf{Y} = \left\{ \frac{x_i}{x_1} \right\}_{i=1}^{n}$$

Example

```
R0          0.0         1.0         2.0         3.0         4.0         5.0
  6.0         7.0         8.0         9.0

R1 := RELCHNG(R0,4);

R1          #           #           #           #           #
  400.00      200.00      133.33      100.00      80.00
```

RELCHNGSYM

Syntax

```
Y := RELCHNGSYM(X, p);
```

Description

The function calculates the relative change of X.

$$\mathbf{Y} = \left\{ \frac{x_i}{x_1} \right\}_{i=1}^{n}$$

Example

```
R0          0.0         1.0         2.0         3.0         4.0         5.0
  6.0         7.0         8.0         9.0

R1 := RELCHNGSYM(R0,4);

R1          #           #           #           #           66.67       57.14
  #           #           #           #
```

OBS Verkar rappakalja! Förklara

RELV

Syntax

```
Y := RELV(X);
```

Description

The function calculates a timeseries Y that is a normalization of X, i e Y shows the relative development of X starting with 1.0.

Den här funktionen kan du använda om du t.ex. vill ha fram den relativa förändringen i en slutkurs från startdagen och framåt.

$$\mathbf{Y} = \left\{ \frac{x_i}{x_1} \right\}_{i=1}^{n}$$

Example

```
R0              100             200             300             200             200             100
  0             100             300             100

R1 := RELV(R0);

R1              1.0             2.0             3.0             2.0             2.0             1.0
  0.0           1.0             3.0             1.0
```

REVERSE

Syntax

```
Y := REVERSE(X);
```

Description

The function reverses the vector X into a vector Y.

$$\mathbf{Y} = \left\{ \frac{x_i}{x_1} \right\}_{i=1}^{n}$$

Example

```
R0              1               2               3               4               5               6
  7             8               9               10
R1 := REVERSE(R0);

R1              10              9               8               7               6               5
  4             3               2               1
```

RKV

Syntax

```
Z := RKV(Y, X, p);
```

Description

The function calculates the r-square value of an linear regression.

$$b = \frac{S_{xy}}{S_{xx}} = \left(n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i \right) \Big/ \left(n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i \right)^2 \right)$$

Example

Y 2	3	9	1	8	7	5
X 6	5	11	7	5	4	4

```
Z := RKV(Y, X, 7);
```

Z	#	#	#	#	#	#
0,06						

ROUND

Syntax

```
Y := ROUND(X);
```

Description

The function calculates a timeseries Y where each element is the integer closest to the corresponding element in X. A value ending with .5 is rounded towards the biggest integer.

Example

R0	1.27	1.49	1.50	1.51	#	-1.49
-1.50	-1.51	2.0	-2.0			

```
R1 := ROUND(R0);
```

R1	1	1	2	2	#	-1
-1	-2	2	-2			

ROUNDS

Syntax

```
i := ROUNDS(r);
```

Description

The function rouns the real r to the integer i.

RSI1

Syntax

```
Y := RSI1(X, p);
```

Description

The function calculates a relative strength index as 100 - 200 / (1 + *su*/*sd*), where *su* is the sum of ups during the last *p* elements and *sd* is the sum of downs during the same period.

$$\mathbf{Y} = \left\{ 100 - \frac{200}{1 + \frac{\sum_{k=i-p+2}^{i} \max(x_{k-1} - x_k, 0)}{\sum_{k=i-p+2}^{i} \max(x_k - x_{k-1}, 0)}} \right\}_{i=p}^{n}$$

Example

```
R0                100         200         300         200         200         100
  0               100         300         100

R1 := RSI1(R0, 4);

R1                                                                33.33       -33.33
  -100.00         -33.33      20.00       0.00
```

Du kan lämpligen reducera funktion ovan till 100*(su - sd)/(su + sd). RSI1 svarar mot att du summerar alla uppgångar och drar ifrån summan av absoultvärdena av alla nedgångar. Dessutom multiplicerar du med 100. Vad du har är 100 gånger skillnaden mellan kursen från startdag till slutdag, p dagar, som divideras med summan av volatiliteten under samma antal dagar.

RSI2

Syntax

```
Y := RSI2(X, p);
```

Description

The function calculates a relative strength index as 100 - 200 * (*su* - *sd*)/(*su* + *sd*), where *su* is the sum of ups during the last *p* elements and *sd* is the sum of downs during the same period.

$$\mathbf{Y} = \left\{ 100 - 200 \frac{\sum_{k=i-p+2}^{i} \max\left(x_{k-1} - x_k, 0\right) - \sum_{k=i-p+2}^{i} \max\left(x_k - x_{k-1}, 0\right)}{\sum_{k=i-p+2}^{i} \max\left(x_{k-1} - x_k, 0\right) + \sum_{k=i-p+2}^{i} \max\left(x_k - x_{k-1}, 0\right)} \right\}_{i=p}^{n}$$

Example

```
R0                100          200          300          200          200          100
  0               100          300          100

R1 := RSI2(R0, 4);

R1                                                                    33.33
  166.67       -100.00       166.67        60.00       100.00
```

Kolla varför inte de båda RSI funktionerna ovan ger samma resultat.

RSI3

Syntax

```
Y := RSI3(X, p);
```

Description

The function calculates a relative strength index as *su*/*sd*, where *su* is the sum of ups during the last *p* elements and *sd* is the sum of downs during the same period.

$$\mathbf{Y} = \left\{ \frac{\sum_{k=i-p+2}^{i} \max\left(x_{k-1} - x_k, 0\right)}{\sum_{k=i-p+2}^{i} \max\left(x_k - x_{k-1}, 0\right)} \right\}_{i=p}^{n}$$

Example

```
R0          100         200         300         200         200         100
  0         100         300         100

R1 := RSI3(R0, 4);

R1                                                        2.00        0.50
  0.00      0.50        1.50        1.00

R1          1           2           12          12          12          6
  7         8           9           10
```

SHIFT

Syntax

```
Y := SHIFT(X, p);
```

Description

The function calculates a timeseries Y that is the same as X but shifted *p* elements forward in time (to the right). Elements that are shifted out to the right are discarded.

SHIFT använder du t.ex. när du vill jämföra ett tidigare värde med ett senare. I exemplet nedan så beräknas skillnaden mellan slutkursen idag och den för 5 dagar sedan.

$$\mathbf{Y} = \left\{ x_{i-p+1} \right\}_{i=p}^{n}$$

Example

```
R0          100         200         300         200         200         100
  0         100         300         100

R1 := SHIFT(R0, 4);

R1                                                        100         200
  300       200         200         100
```

Då gör du så här. Skillnaden := Main.Closae - SHIFT(Main.Close, 5);

SIGN

Syntax

```
Y := SIGN(X);
```

Description

The function calculates the sign of the elements in X.

Example

```
X:          3          0          3          -3         3          3
  0         -2

Y := SIGN(X);

Y 1         0          1          -1         1          1          0
  -1
```

SMOOTH

Syntax

```
Y := SMOOTH(X, f);
```

Description

The function calculates a timeseries Y where small changes to a larger trend is filtered away. If the trend is upgoing and there is a small dip it will be substitued with the current level. *f* controls how large (in percent) a change in the trend should be not to be ignored.

Den här funktionen ger dig möjlighet att från t.ex. en sekundärtrend filtrera bort alla andra trender av lägre dignitet. Antag t.ex. att du vill filtrera bort alla sekundärtrenden motriktade mindre trender än de som haft en kursutveckling mot sekundärtrenden på mer än 7 procent. Då gör du så här. Sekundärtrend := SMOOTH(Main.Close, 7);

Example

```
R0          100        110        108        115        112        110
  105       102        103        100

R1 := SMOOTH(R0, 10.0);

R1          100        110        110        115        115        115
  105       102        102        100
```

SQRT

Syntax

```
Y := SQRT(X);
```

Description

The function calculates the square root of every element in X and returns it in Y.

$$\mathbf{Y} = \left\{\sqrt{x_i}\right\}_{i=1}^{n}$$

Example

```
R0          100         25          16          10          36          64
  81        #           -25         0

R1 := SQRT(R0);

R1          10          5           4           3.16        6           8
  9         #           #           0
```

SQRTN

Syntax

```
Y := SQRTN(X, k);
```

Description

The function calculates the k:th root of every element in X and returns it in Y.

$$\mathbf{Y} = \left\{\sqrt[k]{x_i}\right\}_{i=1}^{n}$$

Example

```
R0          8           10          100         1           27          64
  125       #           -8          0

R1 := SQRT(R0, 3);

R1          2.00        2.15        4.64        1.00        3.00        4.00
  5.00      #           #           0.00
```

STDERRXY

Syntax

```
Z := STDERRXY(Y, X, p);
```

Description

The function calculates the standard deviation for the predictions of Y for each value in X.

$$b = \frac{S_{xy}}{S_{xx}} = \left(n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i \right) \Big/ \left(n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i \right)^2 \right)$$

Example

Y 2	3	9	1	8	7	5
X 6	5	11	7	5	4	4

```
Z := STDERRXY(Y, X, 7);
```

Z	#	#	#	#	#	#
3,31						

STDEV

Syntax

```
Y := STDEV(X, p);
```

Description

The function calculates the standard deviation for the previous *p* elements.

$$\mathbf{Y} = \left\{ \sqrt{\frac{1}{p-1} \sum_{k=i-p+1}^{i} \left(x_k - \bar{x} \right)^2} \right\}_{i=p}^{n} = \left\{ \sqrt{\frac{1}{p-1} \left(\sum_{k=i-p+1}^{i} x_k^2 - \frac{1}{p} \left(\sum_{k=i-p+1}^{i} x_k \right)^2 \right)} \right\}_{i=p}^{n}$$

Example

R0	100	110	108	115	112	110
105	102	103	100			

```
R1 := STDEV(R0, 4);
```

R1				6.24	2.99	2.99
4.20	4.57	3.56	2.08			

STEP

Syntax

```
Y := STEP(A, X);
```

Description

The function copies all elements in X for which the corresponding element i A is true to Y. If the corresponding element in A is false the same value that was copied the last time is copied to Y.

$$y_i = \begin{cases} x_i & \text{if } a_i \text{ is true} \\ y_{i-1} & \text{if } a_i \text{ is false} \end{cases}, \text{ for } i = 1,\ldots,n$$

Antag att du vill fixera kursen vid ett köptillfälle för att kunna beräkna utfallet av köpet vid ett senare säljtillfälle. En del i ett sådant förfarande är följande. Köp := FILTERBUY(b1, b2); Köpkursen := STEP(Köp, Main.Close);

Example

B0	T	T	F	T	F	T
T	T	F	T			
R0	100	110	108	115	112	110
105	102	103	100			
R1 := STEP(B0, R0);						
R1	100	110	110	115	115	110
105	102	102	100			

STOPLOSS

Syntax

```
C := STOPLOSS(A, B, X, f1, f2);
```

Description

The function adds sell signals to the sell timeseries B if any stop-loss sell condition controlled by *f1* and *f2* is true for the price timeseries X. A is the buy timeseries. Sell signals are added if the price sinks *f1* percent below the buy price or *f2* percent under the maximum price after the buy.

STOPLOSS är en funktion, som du bör använda i samband med säljvillkor. Den är så smart utformad att du har möjlighet lägga in två stoppvillkor, det ena med utgångspunkt från köpkursen och ytterligare en beräknade på högsta kurs från köptillfället.

Example

```
B0          T          F          F          T          T          F
  F         T          F          F
B1          F          F          T          F          F          F
  F         F          F          F
R0          105        104        100        102        120        105
  105       108        105        100

B2 := STOPLOSS(B0, B1, R0, 0.02, 0.05);

B2          F          F          T          F          F          T
  F         F          F          T
```

STRINGCOMPARE

Syntax

x := STRINGCOMPARE(str1,str2);

Description

The function compare two strings and returns -1 if str1 comes before str2, +1 if str1 comes after str2, and zero then str1 equals str2.

Example

```
x := STRINGCOMPARE("jambalaja","rendeveuz");
x -1
```

Utveckla. Förstår ej.

SUM

Syntax

Y := SUM(X, p);

Description

The function sums vector X periodically with period p.

Här har du möjlighet att summera t.ex. unika köpsignaler för ett antal dagar. T.ex. skulle SummaUnikaKöp := SUM(UnikaKöp, 5); ge dig summan av alla unika köpsignaler de senaste 5 börsdagarna.

$$\mathbf{Z} = \left\{ \sum_{k=1}^{i} f(a_k, x_k, y_k) \right\}_{i=1}^{n}, \text{ where } f(a,x,y) = \begin{cases} x, & \text{if } a \text{ is true} \\ y, & \text{if } a \text{ is false} \end{cases}$$

Example

```
R0          0.0         1.0         2.0         3.0         4.0         5.0
  6.0         7.0         8.0         9.0

R1 := SUM(R0,4);

R1          #           #           #           6.0         10.0        14.0
  18.0        22.0        26.0        30.0
```

THETA

Syntax

```
Z:= THETA(X, Y, p, i, c);
```

Description

The function calculates the change of theoretical value of an option one day ahead. For arguments see BS.

TimeUnit

Syntax

```
i := TimeUnit(timevec);
```

Description

The function returns the current time unit as an integer. With the current time unit we mean the type of data the current function in executing on. A value of 2 would mean that the function is using day-data.

1 - Intraday

2 - Day

3 - Week

4 - Month

Example

```
i := TimeUnit(timevec);
```

i 2 (indicating that this is day-data)

TOP

Syntax

```
Y := TOP(X, p);
```

Description

Y[i] = the *p*:th local maximum extreme value of X before X[i].

X[i] is a local maximum extreme value if X[i-1] > X[i] and X[i] <= X[i+1].

TOP ger dig det senaste högsta värdet för t.ex. slutkursen om du skriver Toppvärde1 := TOP(Main.Close, 1); och det näst sista högsta värdet om du skriver Toppvärde2 := TOP(Main.Close,2);

Example

```
R0          3          4          3          5          1          6

R1 := TOP(R0, 1);

R1                                 4          4          5          5
```

TOPD

Syntax

```
Y := TOPD(X, p);
```

Description

Y[i] = the distance to the *p*:th local maximum extreme value of X before X[i].

X[i] is a local maximum extreme value if X[i-1] > X[i] and X[i] <= X[i+1].

TOPD använder du när du vill ha avstånden till de senaste topparna.

Example

```
R0          3          4          3          5          1          6

R1 := TOPD(R0, 1);

  R1                               1          2          1          2
```

TPARAB

Syntax

```
Z := TPARAB(L, H, b, c);
```

Description

The function calculates the limits for going short and going long according to the parabolic trading model. The parabolic growth rate starts at zero and increments with b for each day until it reach its maximum c. Z is the composition of X and Y. See also PARAB3.

TREND, PTREND, NTREND

Syntax

```
X := TREND(H, L, b, f, w, p);
```

```
Y := PTREND(H, L, b, f, w, p);
```

```
Z := NTREND(H, L, b, f, w, p);
```

Description

The functions calculates the strength of the trend. PTREND shows the strength of the upward trend, NTREND of the downward trend, and TREND is the sum of PTREND and NTREND. H and L are high and low closing prices, b i time period to go back to find the trend, f is the period for which to calculate the trend, w is the bandwidth for the band in the backward look, p is the length of the moving average that is used to calculate TREND from PTREND and NTREND.

De här funktionerna används för att konstruera Wilder´s Direktional Movement modell.

Truncate

Syntax

```
K := Truncate(X);
```

Description

The functions truncates every element in the realvector X and converts the vector to an integervector.

Example

```
R0          3.21        4.9         3.031       #           198.50      31.51

I1 := TOPD(R0, 1);

 I1         3           4           3           #           198         31
```

VMAVN

Syntax

Z := VMAVN(X, Y**);**

Description

The function calculates a normal moving average of length for the timeseries X and returns it in Z. The period for the moving average in each element of Z is given by the corresponding element in Y.

$$\mathbf{Z} = \left\{ \frac{1}{y_i} \sum_{k=i-y_i+1}^{i} x_k \right\}_{i=1}^{n}$$

I VMAVN har du möjlighet att ha en medelvärdeslängd som varierar över tiden.

Example

```
R0          100         105         103         101         110         112
  108       115         118         120
R1          3           3           2           4           3           4
  2         3           4           3

R2 := VMAVN(R0, R1);

 R2         #           #           104.00      102.25      104.67
 106.50     110.00      111.67      113.25      117.67
```

VRSI1

Syntax

Z := VRSI1(X, Y**);**

Description

The function calculates a relative strength index as 100 - 200 / (1 + *su*/*sd*), where *su* is the sum of ups during a period whose length is detemined by the corresponding element in Y and *sd* is the sum of downs during the same period.

Example

```
R0          100     105     103     101     110     112
  108       115     118     120
R1          3       3       2       4       3       4
  2         3       4       3

R2 := VRSI1(R0, R1);

  R2        #       #       42.86   #       38.46   46.67
  -33.33    38.46   50.00   100.00
```

Kolla mot RSI funktionerna ovan. Plocka bort onödiga.

VSHIFT

Syntax

```
Z := VSHIFT(X, Y);
```

Description

Variable shift.

$$z_i = x_{i-y_i}$$

Här har du möjlighet att variera längden på SHIFT. Så ger t.ex. Köpkursen := VSHIFT(Main.Close, TOPD(Köp, 1); om Köp := ONE(BUY);

Example

```
R0          100     101     102     103     104     105
  106       107     108     109
R1          3       3       2       4       3       4
  2         3       4       3

R2 := VSHIFT(R0, R1);

  R2        #       #       100     #       101     101
  104       104     104     106
```

WSUM

Syntax

```
Y := WSUM(X, p);
```

Description

The function sums vector X weighted periodically with period p.

$$\mathbf{Z} = \left\{ \sum_{k=1}^{i} f\left(a_k, x_k, y_k\right) \right\}_{i=1}^{n}, \text{ where } f\left(a,x,y\right) = \begin{cases} x, & \text{if } a \text{ is true} \\ y, & \text{if } a \text{ is false} \end{cases}$$

Example

```
R0          0.0         1.0         2.0         3.0         4.0         5.0
  6.0       7.0         8.0         9.0

R1 := WSUM(R0,4);

R1          #           #           #           20.0        30.0        40.0
  50.0      60.0        70.0        80.0
```

Förklara. Förstår ej.

XOR

Syntax

```
C := XOR(A, B);
```

Description

The function calculates the logical exclusive or between the boolean vectors A and B.

Example

```
B0          T           F           F           F           F           T
  F         F           F           F
B1          F           F           F           T           F           F
  F         F           T           F

B2 := XOR(B0, B1);

B2          T           F           F           T           F           T
  F         F           T           F
```

XPAF

Syntax

```
Y := XPAF(H, L, X, a, b, c, d, e);
```

Description

The function calculates a lineversion of Point and Figure. It takes the high prices, H, and low prices, L, and close prices, X. The parameters are use box size in percentage (1 – yes), a, and use different box size in different intervals (1 – yes), b, and box size, c, and boxes needed for reversal, d, and use the main rule (1 – yes), e.

YIELD

Syntax

```
Y := YIELD(A, B, X, d);
```

Description

The function calculates the yield when buying and selling a security at the price given in X and following the buy and sell signals in the buy timeseries A and the sell timeseries B. The result is a timeseries Y with the yield as yearly effective interest given in percent. *d* is the number of days in a year.

YIELD kan du använda för att beräkna avkastningen på din trading i form av årsränta för det kapital och för de perioder som du varit lång i aktien ifråga.

Example

```
B0          T          F          F          F          F          T
  F         F          F          F
B1          F          F          F          T          F          F
  F         F          T          F
R0          100        105        110        105        102        110
  130       135        110        112

R1 := YIELD(B0, B1, R0, 240);

R1          0          0          0          300        300        300
  300       300        150        150
```

Kolla att det inte är effektiv utan enkel årsränta.