Upprepade beräkningar

Redan för flera tusen år sedan visste Babylonierna hur man beräknade närmevärden för kvadratrötter. Så här gjorde de till exempel när de skulle beräkna $\sqrt{10}$:

De började med en första gissning, t.ex. x_1=7/2=3,5. När vi kvadrerar x_1=7/2 får vi 49/4, som är större än 10. Alltså är $7/2 > \sqrt{10}$. Om vi nu beräknar 10/x_1= 10/(7/2) får vi 20/7, vars kvadrat är 400/49, som är mindre än 10. Vi får då att $10 / x_1 < \sqrt{10}$.

För att få ett *bättre* närmevärde tar vi nu *medelvärdet*, som vi kallar x_2:

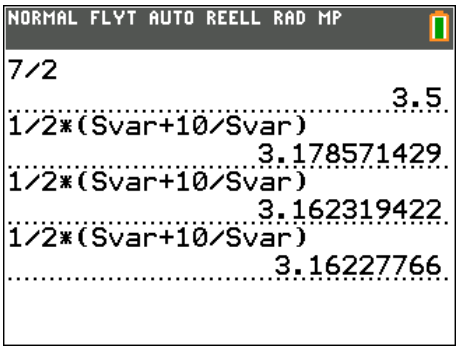
$$x_2 = \frac{1}{2}\left(x_1 + \frac{10}{x_1}\right) = \frac{1}{2}\left(\frac{7}{2} + \frac{10}{7/2}\right) = \frac{89}{28}$$

Vi beräknar nu ett ännu bättre närmevärde x_3 genom att stoppa in värdet på x_2 i formeln igen. Du ska skriva dina tal i *bråkform* och inte som avrundade decimaltal. Vilket värde får du då på x_3 och hur stor blir skillnaden i förhållande till det värde du får om du använder grafräknarens rotknapp.

Vi får $x_3 = \frac{1}{2}\left(\frac{89}{28} + \frac{10}{\frac{89}{28}}\right) = \frac{15761}{4984}$

Vi får: $\frac{15761}{4984} \approx 3{,}162319422$ Räknarens rotknapp ger 3,16227766

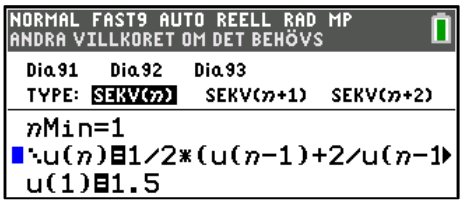
Beräkningarna ovan kan man göra på ett enkelt sätt på räknaren. Börja med att knappa in en första gissning, till exempel 7/2 = 3,5). Följ sedan anvisningarna ovan för att få bättre allt bättre närmevärden.

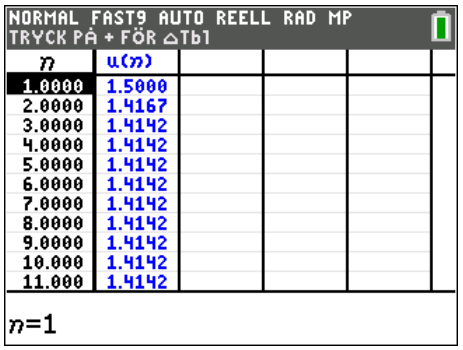


Där det på skärmen nu står svar motsvarar tangenten [ans]. Du trycker på [2nd] [(-)] för att komma åt denna funktion. Ans står för svaret i den sista beräkningen. När du gjort den första beräkningen med formeln trycker du bara på [enter] flera gånger. Pröva gärna med något annat tal, till exempel 10/3, och jämför med det värde du får när du trycker på rotknappen.

Denna övning handlar ju om *rekursiva talföljder* och det tas upp först i kurs 5. Just detta med ett listigt sätt att beräkna kvadratrötter tror vi att du hänger med på om du studerar kurs 3b och 3c. Det visar ju hur man kan använda räknarens ans-funktion. Kontrollera vilken språkinställning du har på grafräknaren. I dessa övningar använder vi hela tiden språkinställningen svenska och det är därför det står Svar på skärmen.

Fördjupning: Man kan utföra samma beräkningar genom att göra inställningen **SEKV** (står för talföljd) och sedan trycka på tangenten [y=] och skriva in enligt skärmen. Här har vi istället beräknat roten ur 2. Efter tre steg får vi 5 korrekta värdesiffror.





Vi ska nu skriva ett program i Python som beräknar ett bra värde. Programmet är kort och från genomgången tidigare så är det knappast något som måste förklaras. Satsen gissa=nyttvärde gör att nästa värde blir en gissning nästa gång som beräkningen utförs.

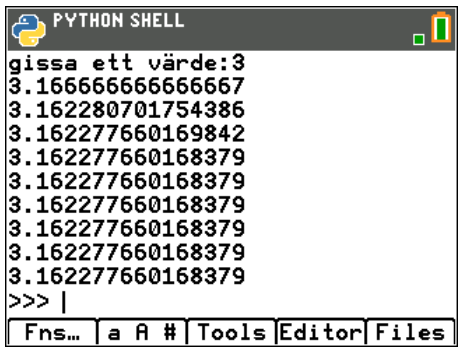
Kör nu programmet och titta på hur många siffror du får i svaret efter 10 upprepade beräkningar. Se skärmbilden till höger nedan. Efter hur många iterationer stabiliseras värdet på kvadratroten?

Jämför nu med om du beräknar roten ur 10 med rottangenten på räknaren.

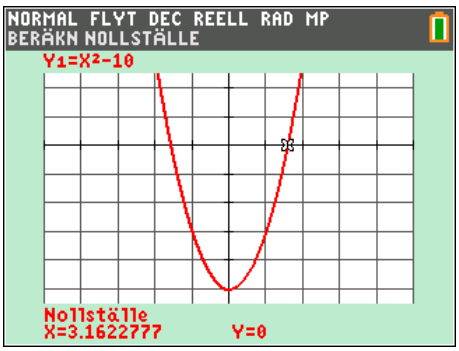
Nu ska du utvidga programmet så att man kan beräkna roten ur ett valfritt tal som du matar in när du kör programmet. Du ska alltså lägga till en input-sats.

Vid körning av programmet för beräkning av roten ur 5 kan det se ut så här:

vilket tal: 5
gissa ett värde: 2.5
..
..



Finns det andra sätt att skapa ett program som beräknar kvadratroten ur ett tal? Titta på funktionen $f(x)=x^2-10$. Denna funktion har ett positivt nollställe för $x=\sqrt{10}$. Se graf till höger. Vi ska nu använda en helt annan metod för att beräkna nollstället. Se nästa sida.



I kapitel 3, övning 1 för Koda med TI går vi igenom en metod för att söka *nollställen* till en tredjegradsfunktion. Metoden kan även användas på andra funktioner. Den metod som används heter *intervallhalvering*. Nedan finns ett utdrag från den senare delen av denna aktivitet.

Efter den här ganska långa genomgången kan man börja skriva in sitt skript.

Först definierar man sin funktion och sedan definierar man en funktion med argumenten *a*, *b* och *prec*. *a* och *b* är gränserna i intervallen och *prec* är precisionen, dvs differensen mellan *a* och *b*.

Så länge som (*while*) avståndet b-a är större än den precision man vill ha så utförs beräkningar av medelvärde och sedan kommer *if …else*-sats med beräkning av vilket tecken (positivt eller negativt värde) som f(a)*f(m) har. Är det negativt så har vi en teckenväxling och nollstället ligger i det vänstra delintervallet. Är det positivt så ligger nollstället i det högra delintervallet.

Tips för inskrivning: Olikhetstecken kommer du åt genom att trycka [2nd] [math]. while finns i kontrollmenyn. Tryck f1 (Fns..) och sedan Ctl. Där finns också villkorsinstruktionen if .. else.

Tryck nu på f4 (Run) för att köra programmet. Trycka på [vars] och välj dela() och tryck sedan på Ok. Fyll nu i argumenten (10,15, 0.001). Tryck sedan på [enter].

Nu får vi beräknade värden på ett smalt intervall. Med två korrekta decimaler blir svaret 11.38.

```
EDITOR: HALVERA
PROGRAM LINE 0001
def f(x):
    return 0.03*x**3-0.3*x**2-0.6*
        x+1.45
def dela(a,b,prec):
    while (b-a)>prec:
        m=(a+b)/2
        if f(a)*f(m)<=0:
            b=m
        else:
            a=m
    return a,b
Fns… | a A # | Tools | Run | Files
```

```
PYTHON SHELL

>>> # Shell Reinitialized
>>> # Running HALVERA
>>> from HALVERA import *
>>> dela(10,15,0.001)
(11.3836669921875, 11.3842773437
5)
>>> |
Fns… | a A # | Tools | Editor | Files
```

Lägg nu in funktionen $f(x) = x^2 - 10$ istället och ringa in ett värde på $\sqrt{10}$. Se skärmbilden här till höger.

```
PYTHON SHELL

>>> # Shell Reinitialized
>>> # Running HALVERA
>>> from HALVERA import *
>>> dela(3,4,0.000000000001)
(3.162277660168002, 3.1622776601
68911)
>>> |
Fns… | a A # | Tools | Editor | Files
```