Eine R-Befehle; Version November 17, 2019

Eine Funktion wird definiert durch

```
nam=function(){}
```

"nam" ist der Name der Funktion, unter dem diese gespeichert wird. Zwischen der Klammer stehen die Argumente der Funktion. Zwischen geschwungener Klammer definiert man die Funktion. Der Name kann frei gewählt werden, sollte aber nicht dem Namen eines R-Befehles entsprechen.

Exemple 1 `f=function(x) 5*x^3-7*x+2`

"f" ist der Name der Funktion. "x" ist das Argument. Es ist nicht nötig, die geschwungenen Klammern zu setzen, wenn die Definition nur aus einer Zeile besteht. Den Funktionswert von 3 kann man nun wie folgt berechnen:

```
f(3)

## [1] 116
```

Nach "##" werden die Ergebnisse des Befehls angegeben.

Exemple 2 `Cap=function(Co,p,t) Co*(1+p)^t`

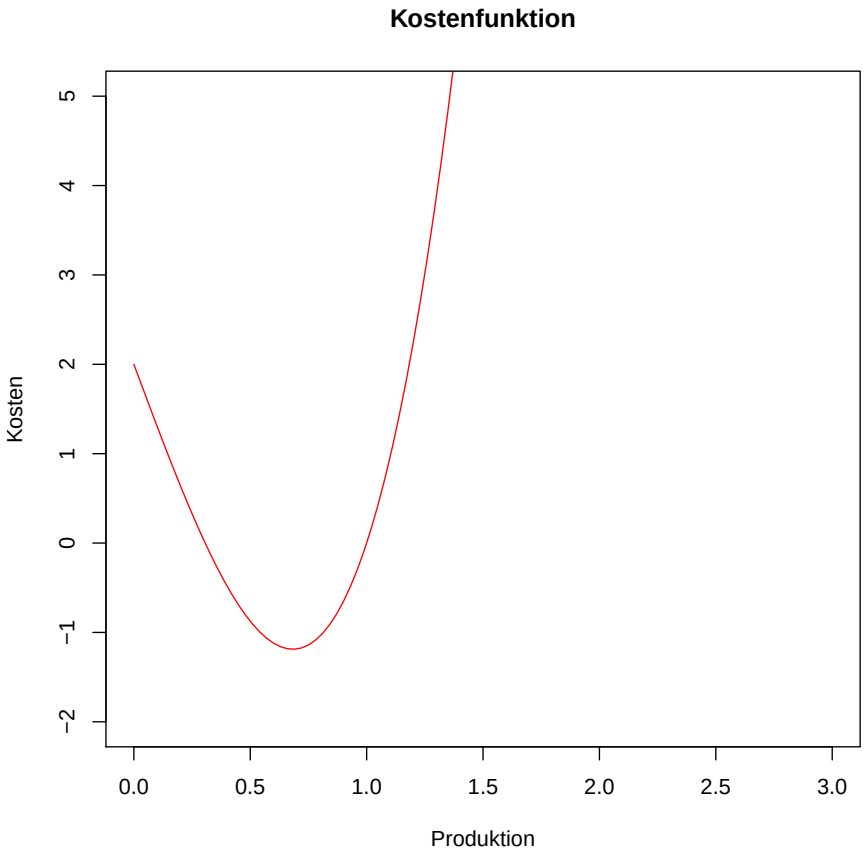
Die Zinseszinsfunktion: ordnet dem Eingangskapital, dem Zinssatz und der Zeit t ein Endkapital zu. Konkret wird im folgenden Beispiel dem Startkapital von 500, dem Zinssatz 3% et der Zeit 5.5 der Endwert 588.2673 zugeordnet):

```
Cap(500,0.03,5.5)

## [1] 588.2673
```

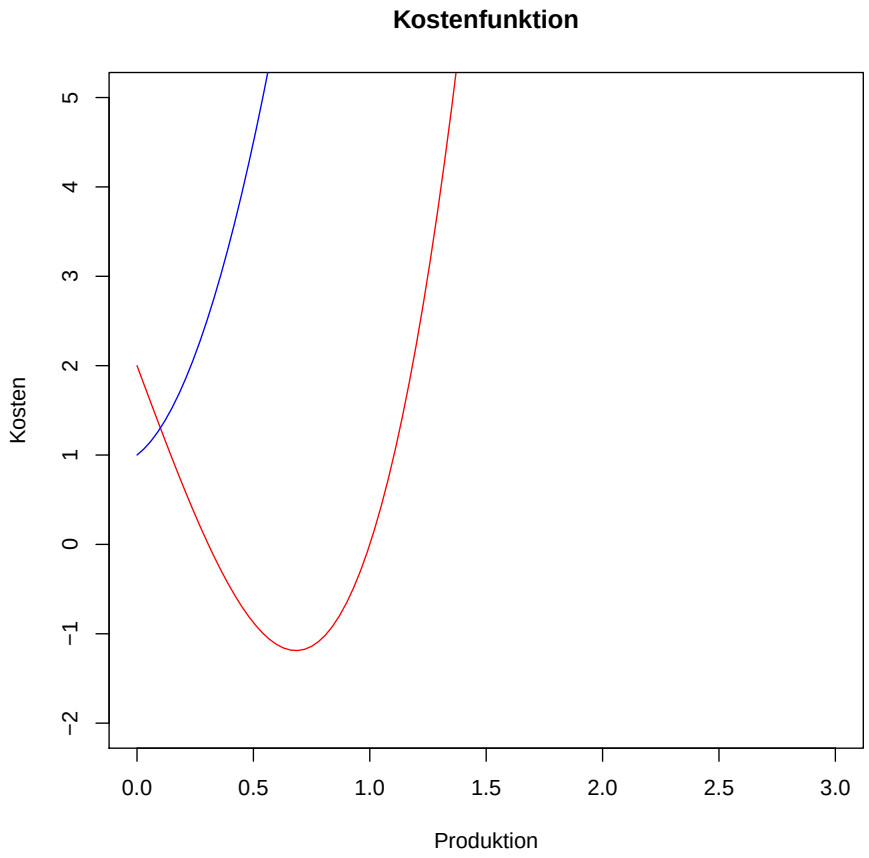
1.1 Graphische Darstellungen

```
plot(f, xlim=c(0,3), ylim=c(-2,5), col = "red",
     xlab="Produktion",ylab= "Kosten",
     main="Kostenfunktion")
```



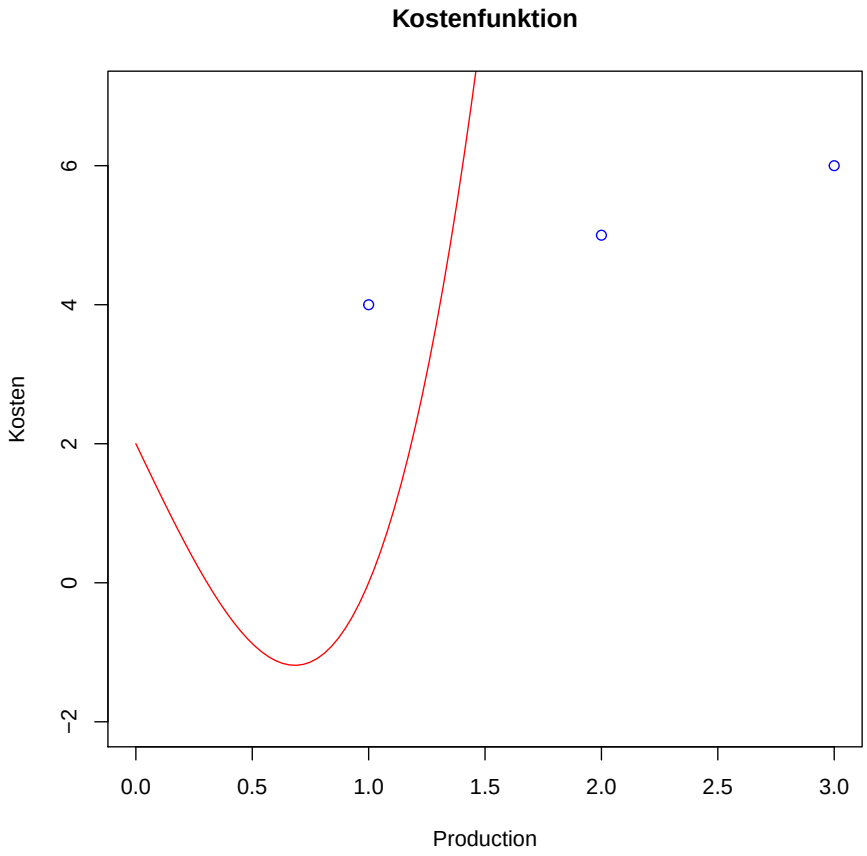
- "plot" öffnet eine graphische Oberfläche, in die die Kurve von f eingezeichnet wird, wenn f eine reelle Funktiion ist (oben definiertes f)
- "xlim" und "ylim" geben die Dimensionen des Plots an (Graduierung der Achsen). Wenn man "xlim" und "ylim" nicht definiert - die entsprechenden Teilbefehle werden einfach weggelassen - so zeichnet "plot" ins Rechteck [0,1]x[0,1]. Im Allgemeinen definiert man nur "xlim".
- Wenn man "col" (für "colour") nicht definiert, zeichnet R in Schwarz.
- Mit "xlab" und "ylab" definiert man die Beschriftungen der Achsen ("lab" für "label").
- "main" gibt der Graphik den Titel.

```
plot(f, xlim=c(0,3), ylim=c(-2,5), col = "red",
     xlab="Produktion",ylab= "Kosten",
     main="Kostenfunktion")
g=function(x) 10*x^2+2*x+1
curve(g,add=T,col="blue")
```



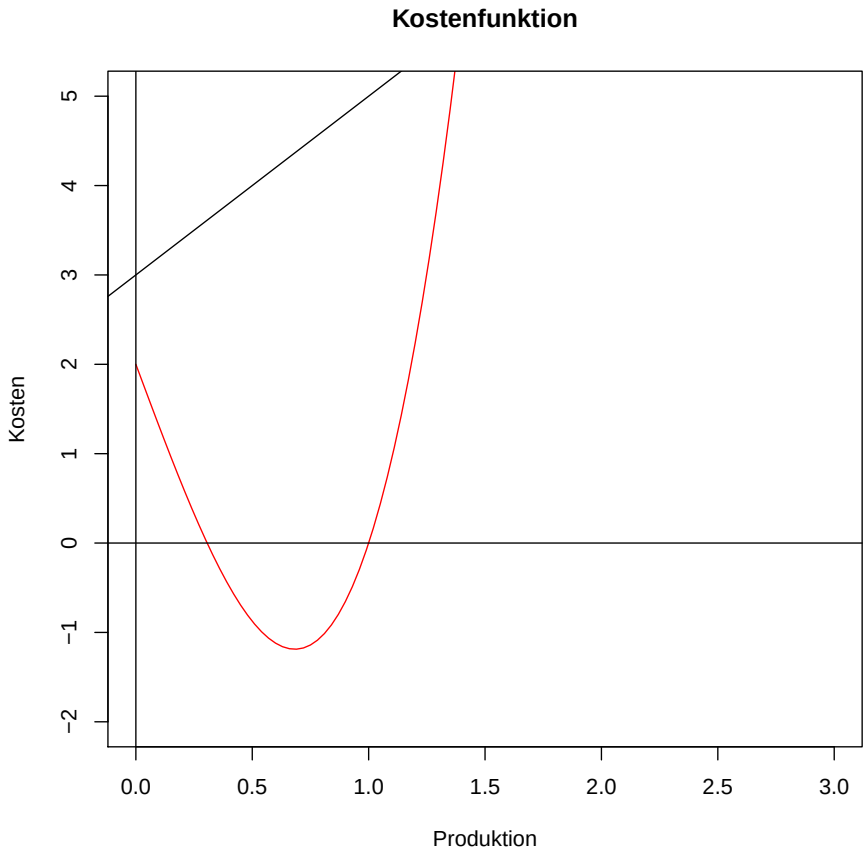
"curve" fügt einem Plot die graphische Darstellung der Funktion g hinzu.

```
plot(f, xlim=c(0,3), ylim=c(-2,7), col = "red",
     xlab="Production",ylab= "Kosten",
     main="Kostenfunktion")
points(x=c(1,2,3),y=c(4,5,6),col="blue")
```



”points” fügt im Beispiel die Punkte (1,4), (2,5), et (3,6) zu einem Plot hinzu (die Anzahl der Komponenten von x und y müssen identisch sein).

```
plot(f, xlim=c(0,3), ylim=c(-2,5), col = "red",
     xlab="Produktion",ylab= "Kosten",
     main="Kostenfunktion")
abline(h=0)
abline(v=0)
abline(3,2)
```



"abline(h=0)" fügt eine horizontale Gerade, die durch (0,0) verläuft, hinzu. "abline (v=0)" fügt eine vertikale Gerade, die durch (0,0) verläuft, hinzu. "abline(3,2)" fügt eine Gerade mit Ordinatenabschnitt 3 und Steigung 3 hinzu.

1.2 n-Tupel

"c()" wird für n-Tupel (= Vektoren) verwendet, z.B.

```
c(5,6,3,8)

## [1] 5 6 3 8
```

stellt das 4-Tupel (5,6,3,8) dar. Man kann n-Tupel unter einem Namen oder einer beliebigen Buchstaben (ausser c, e) speichern (bei Namen ohne Leerzeichen) z.B. unter "a"

```
a=c(2,1,8.2,39,8)
```

Nach der Ausführung von "a=c(2,1,8.2,39,8)" passiert scheinbar nichts. In der Tat wird aber das 5-Tupel c(2,1,8.2,39,8) unter "a" abgespeichert. Führt man "a" aus, erhält man:

```
a

## [1]  2.0  1.0  8.2 39.0  8.0
```

Es ist nicht nötig, "a" nochmals zu schreiben. Man kann nach der Ausführung des obigen Befehls in diesem "a" beleuchten und dann ausführen.
Um in einem n-Tupel "a" die dritte Zahl zu wählen, schreibt man

```
a[3]

## [1] 8.2
```

Wenn man die zweite und die vierte Komponente von "a" wählen will, schreibt man

```
a[c(2,4)]

## [1]  1 39
```

In umgekehrter Ordnung

```
a[c(4,2)]

## [1] 39  1
```

"c(sd,sg[1])" bildet aus dem m-Tupel "sd" und der ersten Komponente des k-Tupels "sg" und m+1-Tupel in der Ordnung (sd, sg[1]). Wenn man (4,1,5,8,9.9) aus a=c(4,1,5,5,6.5,6.7) und de b=c(8,7,9.9,7.5) bilden will, schreibt man

```
a=c(4,1,5,5,6.5,6.7)
b=c(8,7,9.9,7.5)
c(a[c(1,2,3)],b[c(1,3)])

## [1] 4.0 1.0 5.0 8.0 9.9
```

"order()" ordnet einem n-Tupel "a" ein $n-$Tupel von natürlichen Zahlen zu, das an erster Stelle angibt, an welcher Stelle in "a" sich die kleinste Zahl befindet, an zweiter Stelle angibt, an welcher Stelle in "a" sich die zweitkleinste Zahl befindet, etc.

```
a=c(5.2,2.1,8.9,3.3)
order(a)

## [1] 2 4 1 3
```

Die kleinste Zahl 2.1 befindet sich an der zweiten Stelle von "a", die zweitkleinste Zahl 3.3 befindet sich an der vierten Stelle von "a". u.s.w. Entsdprechend kann man mit folgenden Befehlen ein n-Tupel von Zahlen der Grösse nach ordnen:

```
a=c(5.2,2.1,8.9,3.3)
a[order(a)]

## [1] 2.1 3.3 5.2 8.9
```

1.3 Nullstellen von Funktionen

Für Polynome $a_0 + a_1x + a_2x^2 +, ... +, a_nx^n$ gibt man die Koeffizienten in aufsteigender Ordnung in den Befehl "polyroot()" ein (als n-Tupel). Es müssen alle n+1-Koeffizienten eingegeben werden. Für die Nullstellen von $f(x) = 5x^5 - x^3 + 4x^2 + x$ gibt man also folgenden Befehl ein:

```
polyroot(c(0,1,4,-1,0,5))

## [1]  0.0000000+0.0000000i -0.2397594+0.0000000i -0.9142078-0.0000000i
## [4]  0.5769836-0.7612755i  0.5769836+0.7612755i
```

Im Beispiel hat es drei relle Nullstellen (und zwei komplexe). In "x+yi" nennt man "x" den reellen Teil und "yi" den imaginären Teil. Eine Zahl ist reell, wenn der imaginäre Teil 0 ist (= 0.00000i). Die Zahl ist komplex, wenn der imaginäre Teil von 0 verschieden ist (z.B. 0.2345i). Wir verwenden nur die reellen Nullstellen. Um sie aus dem R-Resultat herauszufiltrieren, kann man wie folgt vorgehen:

```
sol=polyroot(c(0,1,4,-1,0,5))
solr = Re(sol)[abs(Im(sol)) < 1e-6]
solr[order(solr)]

## [1] -0.9142078 -0.2397594  0.0000000
```

"Re" behält nur den reellen Teil bei und streicht den imaginären. "abs" berechnet den Betrag. "Im" behält nur den imaginären Teil bei und streicht den reellen. Der Befehl in der zweiten Zeile behält also vom Vektor "sol" den reellen Teil an den Stellen bei, an denen der imaginäre Teil kleiner als 0.000006 ist.

1.4 Matrizen

Die Matrix

$$\begin{array}{cccccc} 4 & 6 & 8 & 3 & 7 & 2 \\ 5 & 11 & \sqrt{2} & \log_2 3 & 5 & 6 \\ 7 & 1 & 1 & 0 & 2 & 2 \end{array}$$

wird eingeben mittels

```
matrix(c(4,6,8,3,7,2,5,11,2^(1/2),log(3,2),5,6,7,1,1,0,2,2),
       ncol=6,byrow=T)
```

```
##      [,1] [,2]     [,3]     [,4] [,5] [,6]
## [1,]    4    6 8.000000 3.000000    7    2
## [2,]    5   11 1.414214 1.584963    5    6
## [3,]    7    1 1.000000 0.000000    2    2
```

Man kann sie unter einem beliebigen Namen abspeichern, z.B. m1 = matrix(....). Man kann auch die Anzahl der Zeilen angeben, im Beispiel mittels "nrow=3"). Wenn man die Daten spaltenweise eingibt, darf byrow=T nicht angegeben werden. Will man auf eine Zeile in der Matrix zugreifen, die unter "m1" abgespeichert wurde, schreibt man m1[3,4] (für die Zelle in der Kreuzung der dritten Zeile und vierten Spalte.
Die Matrix

$$\begin{array}{cccc} 1 & 5 & 5^2 & 5^3 \\ 1 & 4 & 4^2 & 4^3 \\ 1 & 3 & 3^2 & 3^3 \\ 1 & 6 & 6^2 & 6^3 \end{array}$$

kann man effizient eingeben durch

```
x=c(5,4,3,6)
mat1=cbind(x^0,x,x^2,x^3)
mat1
```

```
##      x
## [1,] 1 5 25 125
## [2,] 1 4 16  64
## [3,] 1 3  9  27
## [4,] 1 6 36 216
```

"cbind" verwandelt die Vektoren in Spalten und bildet daraus eine Matrix. Die Multiplikation einer Matrix $\mathbf{A}$ mit einem n-Tupel $\mathbf{y}$:

```
y=c(5,4,3,6)
A=cbind(x^0,x,x^2,x^3)
A%*%y
```

```
##      [,1]
## [1,]  850
## [2,]  453
## [3,]  206
## [4,] 1433
```

Die Lösung des GleichungssystemsAx=y

```
x=c(4,5,6,7)
y=c(5,4,3,6)
A=cbind(x^0,x,x^2,x^3)
solve(A,y)

##                          x
## -71.0000000  48.3333333 -10.0000000   0.6666667
```

oder mittels

```
solve(A)%*%y

##           [,1]
##   -71.0000000
## x  48.3333333
##   -10.0000000
##     0.6666667
```

"solve(A)" berechnet die Umkehrmatrix von A.
Der Befehl ("ls" für "list")

```
ls()

##  [1] "a"    "A"    "b"    "Cap"  "f"    "g"    "mat1" "nam"  "sol"  "solr"
## [11] "x"    "y"
```

gibt die in der Konsole abgespeicherten Objekte an. Der Befehl ("rm" für "remove")

```
rm(a)
```

entfernt das "a" aus der Konsole.