Maxima-Funktionen zur Positionsbestimmung aus GPS-Rohdaten

Alle nachfolgenden Funktionen sind in dieser Reihenfolge in der Maxima-Makrodatei *gps.mac* enthalten.

0 Hilfsfunktionen

0.1 Rationalmachen von Werten nicht anzeigen

```
ratprint:false
```

0.2 Matritzenberechnung

Hat *ratmx* den Wert *true*, werden Matrixoperationen in einer CRE-Darstellung (CRE - Canonical Rational Expressions) ausgeführt und das Ergebnis ist ebenfalls in einer CRE-Darstellung.

```
ratmx:true;
```

0.3 Eckige Matrixklammern

```
lmxchar:"["$
rmxchar:"]"$
```

0.4 3D-Grafik einbinden

```
load(draw);
```

0.5 Paket für die Newton-Näherung laden

```
load(mnewton);
```

0.6 Konstanten definieren

```
v_light:299792458
```

1 Einleitung

2 Verwendung von Maxima

2.1 Umrechnung Grad- in Bogenmaß

Aufrufparameter: Gradmaß

Funktionsergebnis: Bogenmaß

```
bogen(grad):=grad*%pi/180
```

2.2 Umrechnung Bogen- in Gradmaß

Aufrufparameter: Bogenmaß

Funktionsergebnis: Gradmaß

```
grad(bogen):=bogen*180/%pi
```

2.3 Hauptwert berechnen

Berechnen des Hauptwerts einer trigonometrischen Funktion

Aufrufparameter: Bogenmaß X

Funktionsergebnis: Hauptwert des Bogenmaßes X

```
hauptwert(X):=float(remainder(X+2*%pi,2*%pi));
```

2.4 GPS-Zeit vor Unter-/Überlauf schützen

Die Zeitangabe Wochensekunden muss im Intervall von -302400 bis 302400 liegen. Durch Berechnungen kann dieses Intervall überschritten werden, davor schützt die nachfolgende Funktion.

Funktionsaufruf: Sekunden

Ergebnisparameter: Sekundenangabe im zulässigen Intervall

```
check_t(sek):=
if sek>302400 then sek-604800
elseif sek<-302400 then sek+604800
else sek;
```

2.5 Differenz-Norm berechnen

Funktionsaufruf: Zwei Vektoren des R3 in Form von zwei Dreierlisten

Ergebnisparameter: Abstand zwischen den Vektorspitzen

```
norm(V1,V2):=
sqrt((V1[1]-V2[1])^2+(V1[2]-V2[2])^2+(V1[3]-V2[3])^2);
```

2.6 Bestimmung des Abstands zwischen zwei in Polarkoordinaten gegebenen Positionen.

Die Funktion rechnet die in dezimalen Polarkoordinaten angegebenen Positionen zurück in ECEF-Koordinaten und bestimmt dann mit Hilfe der Funktion norm() den Abstand zwischen beiden Positionsangaben.

Funktionsaufruf: Zwei Positionsangaben in Dezimalgrad wie von recpos() geliefert

Ergebnisparameter: Abstand zwischen den beiden Positionen

```
entfernung(pos1,pos2):=block(
[phi1,lambda1,r1,phi2,lambda2,r2,x1,y1,z1,x2,y2,z2,p1,p2],
[phi1,lambda1,r1]:pos1,
[phi2,lambda2,r2]:pos2,
z1:[(6378137+r1)*sin(bogen(phi1))],
x1:[(6378137+r1)*cos(bogen(phi1))*cos(bogen(lambda1))],
y1:[(6378137+r1)*cos(bogen(phi1))*sin(bogen(lambda1))],
p1:matrix(x1,y1,z1),
z2:[(6378137+r2)*sin(bogen(phi2))],
x2:[(6378137+r2)*cos(bogen(phi2))*cos(bogen(lambda2))],
y2:[(6378137+r2)*cos(bogen(phi2))*sin(bogen(lambda2))],
p2:matrix(x2,y2,z2),
norm(p1,p2));
```

2.7 URL für Google-Maps erstellen

Aus der mit der Funktion recpos() erstellten Positionsangabe wird eine URL erstellt, welche die errechnete Position in Google-Maps darstellt.

Funktionsaufruf: Mit recpos() errechnete Position

Ergebnisparameter: URL zur Positionsdarstellung in Google-Maps. Achtung: Die nach dem Einfügen in die URL-Zeile des Browsers erscheinenden Anführungszeichen am Anfang und am Ende des Strings müssen manuell entfernt werden!

```
make_url(pos):=block(
[url,nord,east],
url:"https://www.google.de/maps/place/",
nord:string(pos[1]),
east:string(pos[2]),
concat(url,nord,",",east));
```

3 NAVSTAR GPS

4 Zeitangaben im GPS-System

4.1 Julianisches Jahresdatum

Ermittelt das Julianische Datum des letzten Februartags des angegebenen Jahres im Zeitraum zwischen 1900 und 2100.

Aufrufparameter: Jahreszahl

Funktionsergebnis: Julianisches Datum des letzten Februartags 0:00 Uhr des angegebenen Jahres.

```
jdy(year):= floor(365.25*(year-1900))+2415078.5;
```

4.2 Monate umorganisieren

Zur einfachen Berechnung des Julianischen Datums werden der Januar und Februar als 13. und 14. Monat an das Vorjahr angehängt

Aufrufparameter: Jahreszahl und Monatszahl

Funktionsergebnis: Jahreszahl und Monatszahl ggf. neu arrangiert

```
jdf(y,m):=if m<3 then [y-1,m+12] else [y,m];
```

4.3 Tage der vollständigen Monate summieren

Bestimmung der Tageszahl kompletten Monate vom 1. März eines Jahres bis zum Ende des angegebenen Vormanats.

Aufrufparameter: Monatszahl

Funktionsergebnis: Anzahl der Tage

```
jdm(m):=floor(30.61*(m+1))-122;
```

4.4 Julianisches Datum bestimmen

Aus dem angegebenen Datum bestehend aus Jahr, Monat, Tag, Stunde, Minute, Sekunde wird das zugehörige Julianische Datum berechnet. Die Funktion ist vereinfacht und rechnet nur korrekt im Zeitraum vom 1.3.1900 bis zum 28.2.2100.

Aufrufparameter: Jahr, Monat, Tag, Stunde, Minute, Sekunde

Funktionsergebnis: Julianisches Datum des angegebenen Zeitpunkts

```
julday(year,m,day,h,min,sec):=block(
[year,m]:jdf(year,m),
(jdy(year)+jdm(m)+day+h/24+min/1440+sec/86400));
```

4.5 GPS-Woche aus dem Julianischen Datum berechnen

Aus dem angegebenen Julianischen Datum wird die zugehörige Nummer der GPS-Woche ermittelt.

Aufrufparameter: Julianisches Datum

Funktionsergebnis: Laufende Nummer der aktuellen GPS-Woche

```
gps_week(jd):=floor((jd-2444244.5)/7);
```

4.6 Wochentag aus dem Julianischen Datum berechnen

Aus dem angegebenen Julianischen Datum wird der Wochentag bestimmt.

Aufrufparameter: Julianisches Datum

Funktionsergebnis: 0 = So ... 6 = Sa

```
wotag(jd):=mod(floor(jd-2444244.5),7);
```

4.7 Wochensekunde aus dem Julianischen Datum berechnen

Aus dem angegebenen Julianischen Datum wird die Sekunde innerhalb der GPS-Woche mit Start um Samstag/Sonntag-Mitternacht berechnet.

Da das Julianische Datum um die Mittagszeit wechselt, synchronisieren wir es zum „normalen Datum“ durch die Addition von 0,5 und nehmen von dieser Summe die Nachkommastelle. Den so erhaltenen Tagesbruchteil addieren wir zur Anzahl der seit Samstag-/Sonntagnacht verstrichenen ganzen Tage – die wir als „Wochentag“ bestimmt haben – und multiplizieren das Ganze mit 86 400, der Anzahl der Sekunden eines Tages

Aufrufparameter: Julianisches Datum

Funktionsergebnis: Seit Sa/So-Mitternacht aufgelaufene Sekunden

```
sow(jd):=block(
[bt],
bt:mod(jd+0.5,1),
```

```
round((bt+wotag(jd))*86400)
);
```

4.8 GPS-Zeit aus dem Julianischen Datum berechnen

Die GPS-Zeit wird nach den seit dem 6.1.1980 vergangenen Wochen und innerhalb der laufenden Woche mit den seit Mitternacht Samstag/Sontag vergangenen Sekunden angegeben.

Funktionsaufruf: Julianisches Datum

Ergebnisparameter: [woche, wochensekunde]

```
gps_time(jd):=[gps_week(jd),sow(jd)];
```

5 Datenübertragung und Entfernungsbestimmung

5.1 Zuordnung der Satelliten zu deren PRN-Codes

Die PRN-Codes der einzelnen Satelliten werden über bestimmte Zellen des G2-Registers definiert. Die nachfolgende Liste stellt eine Zuordnung zwischen der Satellitennummer und diesen Zellennummern her.

```
SV_PRN:[[2,6],[3,7],[4,8],[5,9],[1,9],[2,10],[1,8],[2,9],[3,10],
[2,3],[3,4],[5,6],[6,7],[7,8],[8,9],[9,10],[1,4],[2,5],[3,6],[4,7],
[5,8],[6,9],[1,3],[4,6],[5,7],[6,8],[7,9],[8,10],[1,6],[2,7],[3,8],
[4,9]];
```

5.2 Schieben G1-Register

Simulation des G1-Registers.

Aufrufparameter: keiner

Funktionsergebnis: Ausgabewert des Registers

```
shift_G1():=block(
[input,erg],
input:mod(G1[3]+G1[10],2),
erg:G1[10],
push(input,G1),
G1:rest(G1,-1),
erg);
```

5.3 Chipmuster des ersten Schieberegisters erzeugen

Aufrufparameter: keiner

Funktionsergebnis: Chipmuster des ersten Schieberegisters

```
make_prn1():=block(
[prn1,chip],
prn1:[],
G1:[1,1,1,1,1,1,1,1,1,1],
for i:1 thru 1023 do
(
    chip:shift_G1(),
    prn1:endcons(chip,prn1)
),
prn1);
```

5.4 Schieben G2-Register

Schieben des G2-Registers.

Aufrufparameter: Angabe der Zellen für die Ausgabeverknüpfung

Funktionsergebnis: Ausgabewert des Registers

```
shift_G2(e1,e2):=block(
[input,erg],
input:mod(G2[2]+G2[3]+G2[6]+G2[8]+G2[9]+G2[10],2),
erg:mod(G2[e1]+G2[e2],2),
push(input,G2),
G2:rest(G2,-1),
erg);
```

5.5 Chipfolge des zweiten Schieberegisters erzeugen

Aufrufparameter: Satellitennummer, für welchen der G2-Code erzeugt werden soll

Funktionsergebnis: Chipmuster des zweiten Schieberegisters

```
make_prn2(SV):=block(
[e1,e2,prn2,chip],
[e1,e2]:SV_PRN[SV],
prn2:[],
G2:[1,1,1,1,1,1,1,1,1,1],
for i:1 thru 1023 do
(
    chip:shift_G2(e1,e2),
    prn2:endcons(chip,prn2)
),
prn2);
```

5.6 Die ersten 50 Chips des zweiten Schieberegisters erzeugen

Aufrufparameter: Satellitennummer, für welchen der G2-Code erzeugt werden soll

Funktionsergebnis: Chipmuster des zweiten Schieberegisters

```
make_prn2_50(SV):=block(
[e1,e2,prn2,chip],
[e1,e2]:SV_PRN[SV],
prn2:[],
G2:[1,1,1,1,1,1,1,1,1,1],
for i:1 thru 50 do
(
    chip:shift_G2(e1,e2),
    prn2:endcons(chip,prn2)
),
prn2);
```

5.7 Mustervergleich des G2-Registers

Es werden jeweils die ersten 50 Chips aus dem G2-Register für alle Satelliten dargestellt.

Aufrufparameter: keine

Funktionsergebnis: Die ersten 50 Chips des G2-Schieberegisters aller Satelliten.
Auf einem breiten Bildschirm können die jeweils erzeugten 50 Chips untereinander dargestellt und die erfolgte Verschiebung deutlich gemacht werden.

```
check_prn2():=block(
for i:1 thru 32 do
(
    G2:[1,1,1,1,1,1,1,1,1,1],
```

```
    print(i,make_prn2_50(SV_PRN[i]))
));
```

5.8 Vollständige PRN-Sequenz aus dem G1- und G2-Register erzeugen

Die Funktion erzeugt aus dem G1- und dem G2-Register regelkonform die PRN-Sequenz des gewünschten Satelliten

Aufrufparameter: Satellitennummer

Funktionsergebnis: PRN-Code des gewünschten Satelliten

```
make_prn(SV):=block(
[e1,e2,prn_liste],
[e1,e2]:SV_PRN[SV],
prn_liste:[],
G1:[1,1,1,1,1,1,1,1,1,1],
G2:[1,1,1,1,1,1,1,1,1,1],
for i:1 thru 1023 do
(
    chip:mod(shift_G1()+shift_G2(e1,e2),2),
    prn_liste:endcons(chip,prn_liste)
),
prn_liste);
```

5.9 Oktale Darstellung

Die ersten zehn Chips einer PRN-Sequenz werden oktal dargestellt.

Aufrufparameter: Satellitennummer als Index auf die Liste SV_PRN

Funktionsergebnis: Oktale Darstellung der ersten 10 Chips

```
oktal(liste):=block(
[s1,s2,s3,s4],
s1:string(first(liste)),
s2:string(liste[2]*4+liste[3]*2+liste[4]),
s3:string(liste[5]*4+liste[6]*2+liste[7]),
s4:string(liste[8]*4+liste[9]*2+liste[10]),
concat(s1,s2,s3,s4));
```

5.10 Chipsumme

Es wird die Anzahl der 1-Chips einer PRN-Sequenz ermittelt

Aufrufparameter: Satellitennummer als Index auf die Liste SV_PRN

Funktionsergebnis: Summe der 1-Chips in der PRN-Sequenz

```
chipsumme(liste):=block(
[summe],
summe:0,
for i in liste do summe:summe+i,
summe);
```

Zur besseren Verdeutlichung der Autokorrelation werden PRN-Codes mit einer geringeren Chipanzahl benötigt, welche auf einen Blick überschaubar sind. Wir verwenden für diesen Zweck die beiden Register B1 und B2 der Länge 5 und erstellen damit Gold-Codes mit einer Länge von 31 Chips. Die dafür nötigen Funktionen mit dem Namensbestandteil „B1“ oder „bsp“ sind nachstehend aufgeführt

5.11 Schieben B1-Register

Simulation des B1-Registers. Das B1-Register ist ein Register der Länge 5, welches eine Chipfolge von nur 31 Chips erzeugt.

Aufrufparameter: keiner

Funktionsergebnis: Ausgabewert des Registers

```
shift_B1():=block(
[input,erg],
input:mod(B1[3]+B1[5],2),
erg:B1[5],
push(input,B1),
B1:rest(B1,-1),
erg);
```

5.12 Chipmuster des ersten Beispielregisters erzeugen

Aufrufparameter: keiner

Funktionsergebnis: Chipmuster des ersten Beispielregisters mit 31 Chips

```
make_bsp1():=block(
[bsp1,chip],
bsp1:[],
B1:[1,1,1,1,1],
for i:1 thru 31 do
(
    chip:shift_B1(),
    bsp1:endcons(chip,bsp1)
),
bsp1);
```

5.13 Schieben B2-Register

Schieben des B2-Registers.

Aufrufparameter: Angabe der Zellen für die Ausgabeverknüpfung

Funktionsergebnis: Ausgabewert des Registers

```
shift_B2(e1,e2):=block(
[input,erg],
input:mod(B2[1]+B2[2]+B2[3]+B2[5],2),
erg:mod(B2[e1]+B2[e2],2),
push(input,B2),
B2:rest(B2,-1),
erg);
```

5.14 Chipfolge des zweiten Beispielregisters erzeugen

Aufrufparameter: Zellennummern für die Ausgabeverknüpfung.

Funktionsergebnis: Chipmuster des zweiten Beispielregisters

```
make_bsp2(e1,e2):=block(
[bsp2,chip],
bsp2:[],
B2:[1,1,1,1,1],
for i:1 thru 31 do
(
    chip:shift_B2(e1,e2),
    bsp2:endcons(chip,bsp2)
),
```

```
bsp2);
```

5.15 /*Gold-Codes der Länge 31 erstellen*/

```
make_bsp([e1,e2]):=block(
[bsp_liste],
bsp_liste:[],
B1:[1,1,1,1,1],
B2:[1,1,1,1,1],
for i:1 thru 31 do
(
    chip:mod(shift_B1()+shift_B2(e1,e2),2),
    bsp_liste:endcons(chip,bsp_liste)
),
bsp_liste
);
```

5.16 Liste um n Stellen zyklisch weiterschieben

Die Zufalls-Liste wird um n Stellen zyklisch nach rechts verschoben

Aufrufparameter: Anzahl der Verschiebungen

Funktionsergebnis: Verschobene Liste

```
shift(liste,n):=block(
for i:1 thru n do
(
    push(last(liste),liste),
    liste:rest(liste,-1)
),
liste);
```

5.17 Anzahl der übereinstimmenden Eins-Werte in zwei Listen feststellen

Aufrufparameter: Zwei zu vergleichende Listen

Funktionsergebnis: Anzahl der übereinstimmenden Eins-Chips

```
korrelation(liste1,liste2):=block(
[summe],
summe:0,
for i:1 thru length(liste1) do
    summe:summe+liste1[i]*liste2[i],
summe
);
```

5.18 Autokorrelation zweier Listen feststellen

Die erstgenannte Liste wird so oft zyklisch nach rechts verschoben, bis sie mit der zweitgenannten Liste übereinstimmt.

Aufrufparameter: Zwei zu vergleichende Listen

Funktionsergebnis: Anzahl der notwendigen Verschiebungen, um Übereinstimmung herzustellen.

```
autokorr(recliste,satliste):=block(
[sum,sum_alt,merker],
sum:0,
sum_alt:0,
for i:1 thru length(recliste) do
(
    sum:korrelation(recliste,satliste),
    if sum>sum_alt then
```

```
    (
        sum_alt:sum,
        merker:i
    ),
    recliste:shift(recliste,1)
),
merker-1);
```

5.19 Empfangssignal erzeugen

Die benannten Listen werden elementweise miteinander addiert, um ein resultierendes Empfangssignal zu simulieren.

Aufrufparameter: Namen der Listen, welche elementweise aufsummiert werden sollen.

Funktionsergebnis: Liste mit den aufsummierten Werten.

```
make_sum([satliste]):=block(
[summenliste],
summenliste:makelist(0,n,1,length(satliste[1])),
for i in satliste do summenliste:summenliste+i,
summenliste);
```

6 GPS-Empfänger und Rohdaten

6.1 RINEX-Dateien lesen

Die mit neueren teqc-Versionen erstellten RINEX-Dateien können nicht mehr als Ganzes mit der Maximafunktion `read_nested_list()` gelesen werden. Es wurde daher eine eigene Funktion implementiert, welche die Datei bis zum Ende des Headers zunächst zeilenweise liest und erst dann die nachfolgenden Daten mittels `read_nested_list()` en block liest.

Um sowohl der Windows- als auch der Mac-Welt gerecht zu werden, muss nach dem Header-Ende wortweise in zwei Versionen gesucht werden.

Aufrufparameter: Pfad- und Dateiname der Datei

Funktionsergebnis: Geschachtelte Liste mit den Nav- bzw. Obs-Daten

```
read_rinex(pfad):=block(
[h,zeile,werte],
h:openr(pfad),
zeile:split(readline(h)),
unless member("END",zeile) and member("OF",zeile) and
        (member("HEADER",zeile) or member(con-
cat("HEADER",ascii(13)),zeile))
        do zeile:split(readline(h)),
werte:read_nested_list(h),
close(h),
werte);
```

7 RINEX-Dateien einbinden

7.1 Ephemeridenmatrix erzeugen

Die verschachtelte Liste (jede Zeile der RINEX-Datei wird als Liste dargestellt) wird so umorganisiert, die die Ephemeriden eines Satelliten jeweils in einer Liste zusammengefasst werden und diese Listen mit den Ephemeriden der einzelnen Satelliten zu einer Liste von Listen zusammengefasst werden.

Aufrufparameter: Verschachtelte Liste mit Ephemeriden ohne Header (NAV-File ohne Header).

Funktionsergebnis: Liste von Listen mit den Ephemeriden der einzelnen Satelliten

```
make_navmatrix(rinex_navdata):=block(
[num_sv,eph_sv,svlist],
navmatrix:[],
```

```
svlist:[],
num_sv:length(rinex_navdata)/8,
for i:1 step 1 thru num_sv do
   (
   eph_sv:[],
   for k:1 step 1 thru 8 do
       (
        if k=1 then
            if not(numberp(rinex_navdata[k][1])) then
                (
                chr:string(first(rinex_navdata[k])),
                sv:eval_string(sremove("G",chr)),
                rinex_navdata[k][1]:sv
                ),
       eph_sv:append(eph_sv,pop(rinex_navdata))
       ),
   navmatrix:endcons(eph_sv,navmatrix),
   svlist:endcons(navmatrix[i][1],svlist)
   ),
print("Anzahl Ephemeriden:" ,num_sv,sort(svlist)),
navmatrix
);
```

7.2 Überblick über Satelliten in der navmatrix

Die Funktion listet die Nummern derjenigen Satelliten auf, von denen Daten in der navmatrix enthalten sind.

Aufrufparameter: keine

Funktionsergebnis: Liste mit den Nummern derjenigen Satelliten, von denen Daten in der navmatrix vorliegen.

```
make_navsatlist():=block(
[navsatlist],
navsatlist:[],
for i:1 thru length(navmatrix) do navsatlist:endcons(navma-
trix[i][1],navsatlist),
sort(navsatlist));
```

7.3 Zeiger auf einen bestimmten Ephemeriden-Datensatz ermitteln

Ausgehend von der Nummer eines konkreten Satelliten liefert diese Funktion den Index auf die Ephemeriden-Daten des jeweiligen Satelliten in der Datenstruktur *navmatrix*.

Aufrufparameter: Nummer des Satelliten

Funktionsergebnis: Index des zugehörigen Ephemeriden-Datensatzes in der *navmatrix*. Falls sich der angegebene Satellit nicht in der Matrix befindet, wird 0 zurückgegeben.

```
find_ephindex(sat_nr):=block(
[index],
index:0,
for i:1 thru length(navmatrix) do
    if navmatrix[i][1]=sat_nr then index:i,
index);
```

7.4 Satellitennummer auslesen

Die folgende Funktion liefert eine Satellitennummer aus dem jeweiligen Eintrag im Header eines RINEX-Beobachtungsclusters. Es wird überprüft, ob es sich um einen GPS-Satelliten handelt. Ist dies der Fall wird dessen Nummer zurückgegeben, sonst die Zahl 0. Außerdem wird überprüft, ob für den

jeweiligen Satelliten auch ein Eintrag in der navmatrix vorhanden ist. Ist dies nicht der Fall, wird ebenfalls null zurückgeliefert.

Aufrufparameter: String mit dem Eintrag der empfangenen Satelliten. Angabe der Positionsnummer des Satelliten, dessen Nummer geliefert werden soll.

Funktionsergebnis: Nummer des Satelliten.

```
read_satnr(satliste,lfd_nr):=block(
[index,zeichenkette,satnr,z,e,navsatlist],
index:(lfd_nr-1)*3+1,
zeichenkette:string(satliste),
satnr:if charat(zeichenkette,index)="G" then
    (
    z:eval_string(charat(zeichenkette,index+1)),
    e:eval_string(charat(zeichenkette,index+2)),
    z*10+e
    )
    else 0,
navsatlist:make_navsatlist(),
if member(satnr,navsatlist) then satnr else 0);
```

7.5 Auslesen der Pseudoentfernung aus der RINEX-Datei

Es wird die i-te Pseudoentfernung eines Datenclusters ausgelesen. Der jeweilige Datencluster muss über die Zeilennummer seiner Headerzeile angegeben werden. Die Pseudoentfernungen befinden sich in der zweiten oder dritten Spalte.

Aufrufparameter: Name der Rinex-Beobachtungstabelle, Zeilennummer der Headerzeile des jeweiligen Datenclusters, Nummer des Eintrags.

Funktionsergebnis: Pseudoentfernung.

```
read_prange_rinex(rinex_obsdata,cluster,i):=block(
[prange],
prange:rinex_obsdata[cluster+i][2],
if prange<10 then prange:rinex_obsdata[cluster+i][3],
prange);
```

7.6 Beobachtungszeit aus dem Header in GPS-Zeit umrechnen

In der RINEX-Beobachtungsdatei sind die Daten der beobachteten Satelliten in der jeweiligen Headerzeile zusammengefasst, welche auch den genauen Beobachtungszeitpunkt enthält. Dieser dort enthaltene Beobachtungszeitpunkt wird in die GPS-Zeit umgewandelt.

Funktionsaufruf: Header einer Beobachtungssequenz in der RIENEX-Datei

Ergebnisparameter: [woche, wochensekunde]

```
read_obstime(cluster):=
gps_time(julday(cluster[1]+2000,cluster[2],cluster[3],
     cluster[4],cluster[5],cluster[6]));
```

7.7 Liste der Pseudoentfernungen erstellen

In Ergänzung zur oben vorgestellten Funktionsversion werden an den Anfang jeder Liste die Beobachtungszeit und die Anzahl der Satelliten eingefügt.

Aufrufparameter: Name der RINEX-Beobachtungstabelle, Zeilennummer der Headerzeile des jeweiligen Datenclusters.

Funktionsergebnis: Liste mit Satellitennummern und zugehörigen Pseudoentfernungen.

```
make_obslist(rinex_obsdata,cluster):=block(
[header,num_sats,satlist,time,rangelist],
```

```
header:rinex_obsdata[cluster],
num_sats:header[8],
satlist:header[9],
time:read_obstime(header),
rangelist:[time,num_sats],
for i:1 thru num_sats do
    (
    if read_satnr(satlist,i)#0 then
        rangeliste:endcons(
            [read_satnr(satlist,i),
            read_prange_rinex(rinex_obsdata,cluster,i)],
            rangelist)
    ),
rangelist[2]:length(rangelist)-2,
rangelist);
```

7.8 Globale Datenstruktur obsmatrix erstellen

Die Beobachtungsdaten der einzelnen Satelliten werden in eine globale Matrix zusammengefasst

Aufrufparameter: Name der RINEX-Beobachtungstabelle

Funktionsergebnis: Globale Datenstruktor *obsmatrix* mit allen Satellitennummern und zugehörigen Pseudoentfernungen.

```
make_obsmatrix(rinex_obsdata):=block(
[z_nr,zeilenzahl,ranges_at_time,num_sats,svlist],
obsmatrix:[],
z_nr:1,
zeilenzahl:length(rinex_obsdata),
while z_nr<zeilenzahl do
    (
    ranges_at_time:make_obslist(rinex_obsdata,z_nr),
    obsmatrix:endcons(ranges_at_time,obsmatrix),
    num_sats:rinex_obsdata[z_nr][8],
    z_nr:z_nr+num_sats+1
    ),
svlist:[],
print(length(obsmatrix),"Beobachtungen"),
for obs in obsmatrix do
(
     for i:3 thru length(obs) do
          svlist:endcons(obs[i][1],svlist),
     print(obs[1][2],sort(svlist)),
     svlist:[]
),
obsmatrix);
```

7.9 Dateinamen einlesen

Die Funktion erfragt den Namensstamm der RINEX-Dateien über die Tastatur.

Aufrufparameter: keine

Funktionsergebnis: Liste mit den Namen der Beobachtungs- und der Navigationsdatei

```
read_name():=block(
```

```
[name,obsname,navname],
name:(read("Namensstamm der RINEX-Dateien:")),
navname:concat(name,"_nav.txt"),
obsname:concat(name,"_obs.txt"),
[navname,obsname]);
```

7.10 Pfade erstellen

Aufrufparameter: Liste mit den Namen der Beobachtungs- und der Navigationsdatei

Funktionsergebnis: Liste mit den Pfaden zur Beobachtungs- und Navigationsdatei

```
make_path(name):=block(
pfad:"/users/emu/gps/ ",
navpfad:concat(pfad,name[1]),
obspfad:concat(pfad,name[2]),
[navpfad,obspfad]);
```

7.11 RINEX-Dateien einlesen und globale Datenstrukturen erstellen

Die Funktion erfragt die Namen der RINEX-Beobachtungs- und Navigationsdatei, erstellt den Pfad, liest beide Dateien ein und erstellt die globalen Datenstrukturen obsmatrix und navmatrix.

Aufrufparameter: keine

Funktionsergebnis: keines. Die Datenstrukturen obsmatrix und navmatrix werden global erstellt.

```
einlesen():=block(
name:read_names(),
pfad:make_path(name),
rinex_nav:read_rinex(pfad[1]),
make_navmatrix(rinex_nav),
rinex_obs:read_rinex(pfad[2]),
make_obsmatrix(rinex_obs),
return(true));
```

7.12 Beobachtungszeiten ermitteln

Aus den Beobachtungsdaten *obsmatrix* werden die Zeitpunkte der enthaltenen Beobachtungen in eine Liste geschrieben.

Aufrufparameter: keine (es wird auf die globale Struktur *obsmatrix* zugegriffen

Funktionsergebnis. Liste mit den Beobachtungszeitpunkten

```
make_timelist():=block(
[timeline],
timeline:[],
for i:1 thru length(obsmatrix) do
    (
    timeline:endcons(obsmatrix[i][1][2],timeline)
    ),
timeline);
```

7.13 Beobachtungsdatensatz für eine bestimmte Epoche ermitteln

Die Funktion liefert einen Zeiger auf denjenigen Datensatz in der *obsmatrix*, der die Daten für die angegebene Beobachtungszeit enthält.

Aufrufparameter: Beobachtungszeit (sow)

Funktionsergebnis: Zeiger z auf den gewünschten Datensatz für den Zugriff mittels *obsmatrix*[z]

```
find_obsindex(time):=block(
[index],
```

```
index:0,
for i:1 thru length(obsmatrix) do
    if obsmatrix[i][1][2]=time then index:i,
index
);
```

7.14 Verfügbare Satelliten aus der obsmatrix auslesen

Die Funktion erstellt eine Liste der Satellitennummern, für die Pseudoranges zum angegebenen Zeitpunkt in der Beobachtungsdatei vorliegen.

Aufrufparameter: Beobachtungszeitpunkt (*sow*)

Funktionsergebnis: Liste mit den Satellitennummern

```
read_avail_sats(time):=block(
[obs_index,satlist],
obs_index:find_obsindex(time),
satlist:[],
for i:3 thru length(obsmatrix[obs_index]) do
     satlist:endcons(obsmatrix[obs_index][i][1],satlist),
sort(satlist));
```

7.15 Pseudorange eines bestimmten Satelliten in einer bestimmten Epoche finden

Die Funktion gibt den zum angegebenen Satelliten gehörenden Pseudorange einer bestimmten Epoche zurück, wobei die einzelnen Epochen einfach über eine laufende Nummer weitergezählt werden.

Aufrufparameter: Beobachtungszeit (sow), Satelliten-Nummer

Funktionsergebnis: Gewünschter Pseudorange

```
read_prange(sow,sv_nr):=block(
[obs_index],
obs_index:find_obsindex(sow),
for i:3 thru length(obsmatrix[obs_index]) do
    if obsmatrix[obs_index][i][1]=sv_nr then
         return(obsmatrix[obs_index][i][2]));
```

Durch die Konvertierung mit RTKCONV notwendige gewordene Veränderungen:

7.16 Beobachtungszeit aus dem Header in GPS-Zeit umrechnen

In der RINEX-Beobachtungsdatei sind die Daten der beobachteten Satelliten in der jeweiligen Headerzeile zusammengefasst, welche auch den genauen Beobachtungszeitpunkt enthält. Dieser dort enthaltene Beobachtungszeitpunkt wird in die GPS-Zeit umgewandelt.

Funktionsaufruf: Header einer Beobachtungssequenz in der RIENEX-Datei

Ergebnisparameter: [woche, wochensekunde]

```
read_obstime8(zeile):=
gps_time(julday(zeile[2],zeile[3],zeile[4],
     zeile[5],zeile[6],zeile[7]));
```

7.17 Satellitennummer auslesen

Die folgende Funktion liefert eine Satellitennummer aus dem jeweiligen Eintrag im Header eines RINEX-Beobachtungsclusters. Es wird überprüft, ob es sich um einen GPS-Satelliten handelt. Ist dies der Fall wird dessen Nummer zurückgegeben, sonst die Zahl 0. Außerdem wird überprüft, ob für den jeweiligen Satelliten auch ein Eintrag in der navmatrix vorhanden ist. Ist dies nicht der Fall, wird ebenfalls null zurückgeliefert.

Aufrufparameter: Satellitennummer aus der Beobachtungszeile.

Funktionsergebnis: Nummer des Satelliten.

```
read_satnr8(eintrag):=block(
[zeichenkette,z,e,navsatlist],
```

```
zeichenkette:string(eintrag),
satnr:if charat(zeichenkette,1)="G" then
    (
    z:eval_string(charat(zeichenkette,2)),
    e:eval_string(charat(zeichenkette,3)),
    z*10+e
    )
    else 0,
navsatlist:make_navsatlist(),
if member(satnr,navsatlist) then satnr else 0);
```

7.18 Liste der Pseudoentfernungen erstellen

In Ergänzung zur oben vorgestellten Funktionsversion werden an den Anfang jeder Liste die Beobachtungszeit und die Anzahl der Satelliten eingefügt.

Aufrufparameter: Name der RINEX-Beobachtungstabelle, Zeilennummer der Headerzeile des jeweiligen Datenclusters.

Funktionsergebnis: Liste mit Satellitennummern und zugehörigen Pseudoentfernungen.

```
make_obslist8(rinex_obsdata,z_nr):=block(
[satanzahl,beobachtungen,sat_nr],
satanzahl:rinex_obs[z_nr][9],
time:read_obstime8(rinex_obsdata[z_nr]),
beobachtungen:[time,satanzahl],
for i:z_nr+1 thru z_nr+satanzahl do
    (
    sat_nr:read_satnr8(rinex_obsdata[i][1]),
    prange:rinex_obsdata[i][2],
    if sat_nr#0 then
        beobachtungen:endcons([sat_nr,prange],beobachtungen)
    ),
beobachtungen
);
```

7.19 Globale Datenstruktur obsmatrix erstellen

Die Beobachtungsdaten der einzelnen Satelliten werden in eine globale Matrix zusammengefasst

Aufrufparameter: Name der RINEX-Beobachtungstabelle

Funktionsergebnis: Globale Datenstruktor *obsmatrix* mit allen Satellitennummern und zugehörigen Pseudoentfernungen.

```
make_obsmatrix8(rinex_obsdata):=block(
[z_nr,ranges_at_time,num_sats,svlist],
obsmatrix:[],
z_nr:1,
while z_nr<length(rinex_obsdata) do
    (
    ranges_at_time:make_obslist8(rinex_obsdata,z_nr),
    obsmatrix:endcons(ranges_at_time,obsmatrix),
    num_sats:rinex_obsdata[z_nr][9],
    z_nr:z_nr+num_sats+1
    ),
svlist:[],
```

```
print(length(obsmatrix),"Beobachtungen"),
for obs in obsmatrix do
    (
    for i:3 thru length(obs) do
    svlist:endcons(obs[i][1],svlist),
    print(obs[1][2],sort(svlist)),
    svlist:[]
    ),
obsmatrix);
```

7.20 RINEX-Dateien einlesen und globale Datenstrukturen erstellen

Die Funktion erfragt die Namen der RINEX-Beobachtungs- und Navigationsdatei, erstellt den Pfad, liest beide Dateien ein und erstellt die globalen Datenstrukturen obsmatrix und navmatrix.

Aufrufparameter: keine

Funktionsergebnis: keines. Die Datenstrukturen obsmatrix und navmatrix werden global erstellt.

```
einlesen8():=block(
name:read_name(),
pfad:make_path(name),
rinex_nav:read_rinex(pfad[1]),
make_navmatrix(rinex_nav),
rinex_obs:read_rinex(pfad[2]),
make_obsmatrix8(rinex_obs),
return(true));
```

8 Beschreibung von Satellitenbahnen

8.1 Ephemeriden darstellen

Aufrufparameter: Satellitennummer

Funktionsergebnis true (die einzelnen Parameter werden mit ihren Werten dargestellt)
false, falls der angegebene Satellit nicht in der navmatrix vorhanden ist.

```
show_eph(sv):=block(
[k],
k:find_ephindex(sv),
if k=0 then (print("SV:",sv,"nicht vorhanden"),return()),
print("       SV: ",navmatrix[k][1]),
print("     Date: ",navmatrix[k][4],".",navmatrix[k][3],".",navma-
trix[k][2]),
print("     Time: ",navmatrix[k][5],":",navmatrix[k][6],":",navma-
trix[k][7]),
print("      af0: ",navmatrix[k][8]),
print("      af1: ",navmatrix[k][9]),
print("      af2: ",navmatrix[k][10]),
print("     IODE: ",navmatrix[k][11]),
print("      crs: ",navmatrix[k][12]),
print("  delta_n: ",navmatrix[k][13]),
print("       M0: ",navmatrix[k][14]),
print("      cuc: ",navmatrix[k][15]),
print("      ecc: ",navmatrix[k][16]),
print("      cus: ",navmatrix[k][17]),
print("   sqrt_A: ",navmatrix[k][18]),
print("      toe: ",navmatrix[k][19]),
```

```
print("      cic: ",navmatrix[k][20]),
print("  OMEGA_0: ",navmatrix[k][21]),
print("      cis: ",navmatrix[k][22]),
print("       i0: ",navmatrix[k][23]),
print("      crc: ",navmatrix[k][24]),
print("    omega: ",navmatrix[k][25]),
print("OMEGA_dot: ",navmatrix[k][26]),
print("    i_dot: ",navmatrix[k][27]),
print("    codes: ",navmatrix[k][28]),
print(" gps_week: ",navmatrix[k][29]),
print("  L2_flag: ",navmatrix[k][30]),
print(" sv_accur: ",navmatrix[k][31]),
print("sv_health: ",navmatrix[k][32]),
print("      tgd: ",navmatrix[k][33]),
print("     iodc: ",navmatrix[k][34]),
print("      tom: ",navmatrix[k][35]),
print(" interval: ",navmatrix[k][36]).
return(true));
```

9 Bestimmung der Satellitenposition

9.1 Ephemeriden in globale Variable übertragen

Die Funktion überträgt die zur manuellen schrittweisen Positionsbestimmung benötigten Werte eines bestimmten Ephemeriden-Datensatzes in die zugehörigen globalen Variablen.

Aufrufparameter: Satellitennummer

Funktionsergebnis: true – wenn die Variablen gesetzt werden konnten
false – wenn sich der angegebene Satellit nicht in der navmatrix befindet.

```
set_eph(sv):=block(
[k],
k:find_ephindex(sv),
if k=0 then return(),
SV:navmatrix[k][1],
af0:navmatrix[k][8],
af1:navmatrix[k][9],
af2:navmatrix[k][10],
crs:navmatrix[k][12],
delta_n:navmatrix[k][13],
M0:navmatrix[k][14],
cuc:navmatrix[k][15],
ecc:navmatrix[k][16],
cus:navmatrix[k][17],
sqrt_A:navmatrix[k][18],
toe:navmatrix[k][19],
cic:navmatrix[k][20],
OMEGA_0:navmatrix[k][21],
cis:navmatrix[k][22],
i0:navmatrix[k][23],
crc:navmatrix[k][24],
omega:navmatrix[k][25],
OMEGA_dot:navmatrix[k][26],
i_dot:navmatrix[k][27],
```

```
return(true));
```

9.2 Berechnung der Satellitenposition

Erste Version einer Funktion zur Berechnung der Satellitenposition in dreidimensionalen ECEF-Koordinaten zu einer bestimmten Uhrzeit.

Aufrufparameter: Wochensekunde, Satellitennummer

Funktionsergebnis: Satellitenposition in dreidimensionalen ECEF-Koordinaten

```
satpos(sow,sv):=block(
[SV,af0,af1,af2,crs,delta_n,M0,cuc,ecc,cus,sqrt_A,toe,cic,OMEGA_0,
cis,i0,crc,omega,OMEGA_dot,i_dot,
GM,OMEGAe_dot,A,tk,n0,n,M,E,dE,E_old,v,Phi,u,r,i, x1,y1,OMEGA,
sat_x,sat_y,sat_z],
GM:3.986005e14,
OMEGAe_dot:7.2921151467e-5,
set_eph(sv),
A:sqrt_A*sqrt_A,
tk:check_t(sow-toe),
n0:sqrt(GM/A^3),
n:n0+delta_n,
M:M0+n*tk,
M:hauptwert(M),
E:M,
dE:1,
for i:1 thru 10 unless abs(dE)<1*10^-12 do
    (
    E_old:E,
    E:M+ecc*sin(E),
    dE:(mod(E-E_old,2*%pi))
    ),
E:hauptwert(E),
v:atan2(sqrt(1-ecc^2)*sin(E), cos(E)-ecc),
Phi:v+omega,
Phi:hauptwert(Phi),
u:Phi             +cus*sin(2*Phi)+cuc*cos(2*Phi),
r:A*(1-ecc*cos(E))+crs*sin(2*Phi)+crc*cos(2*Phi),
i:i0+i_dot*tk     +cis*sin(2*Phi)+cic*cos(2*Phi),
x1:cos(u)*r,
y1:sin(u)*r,
OMEGA:OMEGA_0+(OMEGA_dot-OMEGAe_dot)*tk-OMEGAe_dot*toe,
OMEGA:hauptwert(OMEGA),
sat_x:x1*cos(OMEGA)-y1*cos(i)*sin(OMEGA),
sat_y:x1*sin(OMEGA)+y1*cos(i)*cos(OMEGA),
sat_z:y1*sin(i),
[sat_x,sat_y,sat_z])$
```

9.3 Berechnung der Satellitenposition in Polarkoordinaten

Gibt die errechnete Satellitenposition in gerundeten Polarkoordinaten aus.

Aufrufparameter: Wochensekunde, Satellitennummer

Funktionsergebnis: Liste mit den gerundeten Polarkoordinaten

```
satpos_polar(sow,sv):=block(
[ecef,polar],
ecef:satpos(sow,sv),
polar:ecef_polar_grad(ecef),
float([round(polar[1]*10)/10,round(polar[2]*10)/10]));
```

10 Almanach

10.1 Almanach-Textdatei einlesen

Liest die als Textdatei vorliegende Almanach Datei in der YUMA-Version ein.

Aufrufparameter: Pfad zur Almanach-Textdatei

Funktionsergebnis: Liste mit den Textzeilen der Almanachdatei

```
read_almanach(pfad):=block(
[h,almanach_txt],
h:openr(pfad),
almanach_txt:read_nested_list(h),
close(h),
almanach_txt);
```

10.2 Matrix aus den Almanach-Textdaten erstellen

Erstellt eine Matrix (doppelt geschachtelte Liste) aus den Almanachdaten. Für jeden Satelliten ist eine Liste mit seinen jeweiligen Daten enthalten.

Aufrufparameter: Almanach-Textdatei

Funktionsergebnis: Globale Matrix *almatrix* mit den Almanachdaten aller Satelliten.

```
make_almatrix(alm_txt):=block(
[sv,ecc,time,i0,OMEGA_dot,sqrt_A,OMEGA_0,omega,M0,
af0,af1,almlist,svlist],
almatrix:[],
svlist:[],
for i:1 thru length(alm_txt) step 15 do
(
    sv:last(alm_txt[i+1]),
    ecc:last(alm_txt[i+3]),
    time:last(alm_txt[i+4]),
    i0:last(alm_txt[i+5]),
    OMEGA_dot:last(alm_txt[i+6]),
    sqrt_A:last(alm_txt[i+7]),
    OMEGA_0:last(alm_txt[i+8]),
    omega:last(alm_txt[i+9]),
    M0:last(alm_txt[i+10]),
    af0:last(alm_txt[i+11]),
    af1:last(alm_txt[i+12]),

almlist:[sv,ecc,time,i0,OMEGA_dot,sqrt_A,OMEGA_0,omega,M0,af0,
af1],
```

```
    almatrix:endcons(almlist,almatrix),
    svlist:endcons(sv,svlist)
),
print("Im Almanach vertretene Satelliten:"),
print(sort(svlist)),
almatrix);
```

10.3 Almanachdaten komfortabel einlesen

Erfragt den Namen der gewünschten Almanachdatei, liest die Datei ein und erstellt die *almatrix*.

Aufrufparameter: keine – Der Namensstamm der Almanachdatei wird erfragt.

Funktionsergebnis: Globale Struktur *almatrix*

```
almanach_einlesen():=block(
[alm_name_list,alm_name,almanach_txt],
alm_name_list:[],
alm_name:read("Namensstamm der Almanach-Datei:"),
pfad:concat("/users/emu/gps/",alm_name,"_alm.txt"),
almanach_txt:read_almanach(pfad),
make_almatrix(almanach_txt),
return(true));
```

10.4 Almanachdaten eines bestimmten Satelliten finden

Liefert den Index auf die Daten eines bestimmten Satelliten in der globalen Almanach-Matrix *almatrix*.

Aufrufparameter: Nummer des gewünschten Satelliten

Funktionsergebnis: Index auf die Almanachdaten des angegebenen Satelliten in der *almatrix*. Sollte der Satellit nicht im Almanach vertreten sein, wird 0 zurückgeliefert.

```
find_almanach(sv):=block(
[index],
index:0,
for i:1 thru length(almatrix) do
    if almatrix[i][1]=sv then index:i,
index);
```

10.5 Almanach eines Satelliten darstellen

Die Almanachdaten eines Satelliten werden dargestellt

Aufrufparameter: Satellitennummer

Funktionsergebnis: true – wenn der Satellit im Almanach vertreten ist
false – wenn dar Satellit nicht im Almanach vertreten ist.

```
show_alm(sv):=block(
[index],
index:find_almanach(sv),
if index=0 then (print("SV:",sv,"nicht vorhanden"),return()),
print("SV:       ",almatrix[index][1]),
print("ecc:      ",almatrix[index][2]),
print("time:     ",almatrix[index][3]),
print("i0:       ",almatrix[index][4],
     "in Grad:",float(almatrix[index][4]*180/%pi)),
```

```
print("OMEGA_dot:",almatrix[index][5]),
print("sqrt_A:   ",almatrix[index][6],
     "quadriert:",almatrix[index][6]^2),
print("OMEGA_0:  ",almatrix[index][7],
     "in Grad:",float(almatrix[index][7]*180/%pi)),
print("omega:    ",almatrix[index][8]),
print("M0:       ",almatrix[index][9]),
return(true));
```

10.6 Satelitenposition aus den Almanachdaten berechnen

Bestimmt die Satellitenposition eines Satelliten zur angegebenen Zeit in ECEF-Koordinaten.

Aufrufparameter: Wochensekunde und Satellitennummer

Funktionsergebnis: ECEF-Koordinaten des gewünschten Satelliten zur angegebenen Zeit.

```
satpos_alm(sow,sv):=block(
[M0,ecc,sqrt_A,toa,OMEGA_0,i0,omega,OMEGA_dot,GM,OME-
GAe_dot,A,tk,n0,
M,E,E_old,dE,v,Phi,u,r,i,OMEGA,x1,y1,sat_x,sat_y,sat_z],
GM:3.986005e14,
OMEGAe_dot:7.2921151467e-5,
index:find_almanach(sv),
M0:almatrix[index][9],
ecc:almatrix[index][2],
sqrt_A:almatrix[index][6],
toa:almatrix[index][3],
OMEGA_0:almatrix[index][7],
i0:almatrix[index][4],
omega:almatrix[index][8],
OMEGA_dot:almatrix[index][5],
A:sqrt_A*sqrt_A,
tk:check_t(sow-toa),
n0:sqrt(GM/A^3),
M:M0+n0*tk,
M:hauptwert(M),
E:M,
dE:1,
for i:1 thru 10 unless abs(dE)<1*10^-12 do
    (
    E_old:E,
    E:M+ecc*sin(E),
    dE:(mod(E-E_old,2*%pi))
    ),
E:hauptwert(E),
v:atan2(sqrt(1-ecc^2)*sin(E),cos(E)-ecc),
Phi:v+omega,
```

```
Phi:hauptwert(Phi),
u:Phi,
r:A*(1-ecc*cos(E)),
i:i0,
x1:cos(u)*r,
y1:sin(u)*r,
OMEGA:OMEGA_0+(OMEGA_dot-OMEGAe_dot)*tk-OMEGAe_dot*toa,
OMEGA:hauptwert(OMEGA),
sat_x:x1*cos(OMEGA)-y1*cos(i)*sin(OMEGA),
sat_y:x1*sin(OMEGA)+y1*cos(i)*cos(OMEGA),
sat_z:y1*sin(i),
[sat_x,sat_y,sat_z]);
```

10.7 Positionen aller Satelliten zum Almanach-Zeitpunkt

Erstellt eine Liste mit den Positionen aller im Almanach vertretenen Satelliten zum Ausgabezeitpunkt des Almanachs.

Aufrufparameter: keine – Es wird auf die almatrix zugegriffen

Funktionsergebnis: Liste mit den ECEF-Koordinaten aller Satelliten

```
positionen():=block(
[pos_list,time,pos],
pos_list:[],
time:almatrix[1][3],
for i:1 thru length(almatrix) do
    (
    pos:satpos_alm(time,almatrix[i][1]),
    pos_list:endcons(pos,pos_list)
    ),
pos_list);
```

10.8 Umrechnung von ECEF- in Dezimal-Polarkoordinaten im Kugelmodell im Gradmaß

Aufrufparameter: Empfängerposition in ECEF-Koordinaten.

Funktionsergebnis: Empfängerposition in dezimalen Polarkoordinaten in Grad.

```
ecef_polar_grad(pos):=
[float(atan2(pos[3],sqrt(pos[1]^2+pos[2]^2))*180/%pi),
float(atan2(pos[2],pos[1])*180/%pi),
float(sqrt(pos[1]^2+pos[2]^2+pos[3]^2)-6378137)];
```

10.9 Liste mit Paaren [Länge,Breite] zum Plotten erstellen

Erstellt aus einer Liste mit Polarkoordinaten der Form [Breite,Länge,Höhe] eine Liste mit Elementen der Form [Länge,Breite].

Aufrufparameter: Geschachtelte Liste mit Polarkoordinaten von Positionen.

Funktionsergebnis: Geschachtelte Liste mit Paaren aus geographischer Länge und Breite.

```
make_polarlist(liste):=block(
[listeneu],
listeneu:[],
```

```
for i in liste do
    listeneu:endcons([i[2],i[1]],listeneu),
listeneu
);
```

10.10 Umrechnung von Kugelkoordinaten in kartesische Koordinaten

Die Funktion berechnet aus der geographischen Breite und Länge eines Orts auf der Erdoberfläche die zugehörigen ECEF-Koordinaten.

Aufrufparameter: phi (geographische Breite) und lambda (geographische Länge)

Funktionsergebnis: ECEF-Koordinaten des angegebenen Orts

```
kugel_cart(phi,lambda):=block(
[Phi,r,x,y,z],
Phi:90-phi,
r:6370000,
x:r*sin(bogen(Phi))*cos(bogen(lambda)),
y:r*sin(bogen(Phi))*sin(bogen(lambda)),
z:r*cos(bogen(Phi)),
[x,y,z]);
```

10.11 Parameter der Kreisgleichung für den Satellitenorbit berechnen

Aus der Lage des aufsteigenden Knotens Omega und der Bahninklination i werden die Parameter für die Parametergleichung eines Kreises errechnet und als Vektor ausgegeben.

Aufrufparameter: Omega (Länge des aufsteigenden Knotens) und i (Inklination)

Funktionsergebnis: Kreisparameter als Vektor

```
kreisparameter(Omega,i):=block(
[e10,e20,e1,e2],
e10:matrix([1],[0],[0]),
e20:matrix([0],[cos(i)],[sin(i)]),
D:matrix([cos(Omega),-
sin(Omega),0],[sin(Omega),cos(Omega),0],[0,0,1]),
e1:D.e10,
e2:D.e20,
26560000*cos(alpha)*e1+26560000*sin(alpha)*e2);
```

10.12 Liste mit den Inklinationsdaten aller Satelliten

Erstellt eine Liste mit den Inklinationsdaten aller im Almanach vertretenen Satelliten

Aufrufparameter: keine – Es wird auf die almatrix zugegriffen

Funktionsergebnis: Sortierte Liste mit den Inklinationswerten aller Satelliten

```
make_inklinationlist():=block(
[ilist],
ilist:[],
for i in almatrix do
    ilist:endcons(float(i[4]),ilist),
sort(ilist));
```

10.13 Liste mit den Knotenlängen aller Satelliten

Erstellt eine geordnete Liste mit den Werten der aufsteigenden Knoten aller im Almanach vertretenen Satelliten.

Aufrufparameter: keine – Es wird auf die almatrix zugegriffen

Funktionsergebnis: Sortierte Liste mit den Knotenlängen aller Satelliten

```
make_omegalist():=block(
[omegalist],
omegalist:[],
for i in almatrix do
    omegalist:endcons(float(i[7]),omegalist),
sort(omegalist));
```

10.14 Liste mit den Längen der Halbachsen

Erstellt eine geordnete Liste mit den Werten der Wurzeln der großen Halbachsen aller im Almanach vertretenen Satelliten.

Aufrufparameter: keine – Es wird auf die almatrix zugegriffen

Funktionsergebnis: Sortierte Liste mit den Wurzeln aus den großen Halbachsen aller Satelliten

```
make_sqrtalist():=block(
[sqrtalist],
sqrtalist:[],
for i in almatrix do
    sqrtalist:endcons(float(i[6]),sqrtalist),
sort(sqrtalist));
```

10.15 Gleichartige Daten clustern

Fasst numerisch innerhalb eines 10°-Intervalls beieinanderliegende Daten zu Clustern zusammen und errechnet die arithmetischen Mittelwerte dieses Cluster. So es – wie bei den Knotenlängen – keine verschiedenen einzelnen Cluster gibt, wird der Mittelwert aller Daten berechnet (Inklination und Wurzel aus der großen Halbachse)

Aufrufparameter: Liste mit den zu clusternden Daten

Funktionsergebnis: Liste mit den Mittelwerten der geclusterten Daten

```
clustern(liste):=block(
[cluster,summe,meanomega],
cluster:[],
meanomega:[],
for i:1 thru length(liste)-1 do
    if abs(liste[i]*180/%pi-liste[i+1]*180/%pi)>10
        then push(i,cluster),
push(length(liste),cluster),
cluster:sort(cluster),
start:1,
summe:0,
for i in cluster do
    (
    for k:start thru i do
        summe:summe+liste[k],
```

```
    print(i-start+1,float(summe/((i-start)+1)*180/%pi)),
    if i-start+1#1 then
     meanomega:endcons(float(summe/((i-start)+1)),meanomega),
    start:i+1,
    summe:0
    ),
meanomega);
```

11 Erster Ansatz zur Positionsbestimmung des Empfängers

11.1 Umrechnung Dezimalgrad in Grad, Minute und Sekunde

Aufrufparameter: Empfängerposition in dezimalen Polarkoordinaten.

Funktionsergebnis: Empfängerposition in Grad, Minuten und Sekunden.

```
polar_gms(pos):=block(
[b,bg,bmr,bm,bs,l,lg,lmr,lm,ls,h],
[b,l,h]:pos,
bg:floor(b),
bmr:((b-bg)*60),
bm:floor(bmr),
bs:floor((bmr-bm)*60),
lg:floor(l),
lmr:((l-lg)*60),
lm:floor(lmr),
ls:floor((lmr-lm)*60),
[[bg,bm,bs],[lg,lm,ls],h]);
```

11.2 Bestimmung des Empfängerstandorts aus vier Satellitenpositionen

Die Funktion nähert den Empfängerstandort iterativ – beginnend im Erdmittelpunkt – an. Die Iteration wird beendet, wenn die Beträge der Abweichungen in Summe kleiner als eins geworden sind. Es werden die ersten vier in der obsmatrix aufgeführten Satelliten für die Berechnung verwendet.

Aufrufparameter: Wochensekunde, für welche die Position berechnet werden soll.

Funktionsergebnis: Empfängerposition in Dezimalgrad.

```
recpos_4(sow):=block(
[estpos,L,svpos,obsindex,d,J,sv,pr,dist,A,pos],
estpos:[0.0,0.0,0.0],
L:matrix([1.0],[1.0],[1.0],[0.0]),
svpos:[],
obsindex:find_obsindex(sow),
unless abs(L[1][1])+abs(L[2][1])+abs(L[3][1])<0.1 do
    (
    d:matrix(),
    J:matrix(),
    for i:3 thru 6 do
        (
        sv:obsmatrix[obsindex][i][1],
        pr:read_prange(sow,sv),
        svpos:satpos(sow,sv),
```

```
        dist:pr-norm(svpos,estpos)-L[4][1],
        d:addrow(d,[dist]),
        A:[(estpos[1]-svpos[1])/norm(svpos,estpos),
           (estpos[2]-svpos[2])/norm(svpos,estpos),
           (estpos[3]-svpos[3])/norm(svpos,estpos),
            1],
        J:addrow(J,A)
        ),
    L:float(invert(J).d),
    estpos[1]:estpos[1]+L[1][1],
    estpos[2]:estpos[2]+L[2][1],
    estpos[3]:estpos[3]+L[3][1]
    ),
ecef_polar_grad(estpos));
```

11.3 Alle 4-Kombinationen empfangener Satelliten erstellen

Die Funktion erstellt eine Liste mit allen Viererkombinationen der empfangenen Satelliten.

Aufrufparameter: Wochensekunde, für welche die Position berechnet werden soll.

Funktionsergebnis: Liste mit den Kombinationen.

```
make_svmix(sow):=block(
[svnrlist,svnrset,ps],
svnrlist:read_avail_sats(sow),
svnrset:setify(svnrlist),
ps:powerset(svnrset,4),
print("Anzahl der Kombinationen:",cardinality(ps)),
full_listify(ps));
```

11.4 Alternative Bestimmung des Empfängerstandorts aus vier Satellitenpositionen

Diese Funktion ermittelt ebenfalls den Empfängerstandort über die Auswertung der Signale von vier Satelliten. Allerdings können die Nummern der zu berücksichtigenden Satelliten in einer Liste übergeben werden.

Aufrufparameter: Wochensekunde, für welche die Position berechnet werden soll und eine Liste mit den Nummern der vier Satelliten, welche für die Berechnung verwendet werden sollen.

Funktionsergebnis: Errechneter Empfängerstandort.

```
recpos_vier(sow,svliste):=block(
[estpos,L,svpos,d,J,sv,pr,dist,A,pos],
estpos:[0.0,0.0,0.0],
L:matrix([1.0],[1.0],[1.0],[0.0]),
svpos:[],
unless abs(L[1][1])+abs(L[2][1])+abs(L[3][1])<0.1 do
    (
    d:matrix(),
    J:matrix(),
    for i:1 thru 4 do
        (
```

```
        sv:svliste[i],
        pr:read_prange(sow,sv),
        svpos:satpos(sow,sv),
        dist:pr-norm(svpos,estpos)-L[4][1],
        d:addrow(d,[dist]),
        A:[(estpos[1]-svpos[1])/norm(svpos,estpos),
           (estpos[2]-svpos[2])/norm(svpos,estpos),
           (estpos[3]-svpos[3])/norm(svpos,estpos),
            1],
        J:addrow(J,A)
        ),
    L:float(invert(J).d),
    estpos[1]:estpos[1]+L[1][1],
    estpos[2]:estpos[2]+L[2][1],
    estpos[3]:estpos[3]+L[3][1],
    if norm(estpos,[0,0,0])>10^10 then return(false)
    ),
ecef_polar_grad(estpos));
```

11.5 Mittelwert aller Satellitenquartette berechnen

Aufrufparameter: Wochensekunde, für welche die Position berechnet werden soll und die Liste mit allen Satellitenquartetten.

Funktionsergebnis: Gemittelter Empfängerstandort.

```
mean_recpos(sow,mix):=block(
[posliste,x_wert,y_wert],
posliste:[],
x_wert:0,
y_wert:0,
for i:1 thru length(mix) do
    (
    pos:recpos_vier(sow,mix[i]),
    x_wert:x_wert+pos[1],
    y_wert:y_wert+pos[2]
    ),
anzahl:length(mix),
[x_wert/anzahl,y_wert/anzahl]
);
```

12 Verbesserte Zeitbestimmung

12.1 Zeitkorrektur

Die Funktion berücksichtigt die Laufzeit des Signals vom Satellit zum Empfänger und die im Ephemeriden-Datensatz angegebenen Korrekturwerte für eine verbesserte Angabe der Beobachtungszeit.

Aufrufparameter: Beobachtungszeitpunkt, Nummer des Satelliten

Funktionsergebnis: Liste mit dem korrigierten Beobachtungszeitpunkt und dem Korrekturwert aufgrund der Ungenauigkeit der Uhr

```
timecorrect(time,sv):=block(
[pseudorange,eph_list,signallaufzeit,tx_RAW,toe,dt,
af0,af1,af2,tcorr,tx_GPS],
v_light:299792458,
pseudorange:read_prange(time,sv),
eph_list:navmatrix[find_ephindex(sv)],
signallaufzeit:pseudorange/v_light,
tx_RAW:check_t(time-signallaufzeit),
toe:eph_list[19],
dt:check_t(tx_RAW-toe),
af0:eph_list[8],
af1:eph_list[9],
af2:eph_list[10],
tcorr:(af2*dt+af1)*dt+af0,
tx_GPS:check_t(tx_RAW-tcorr),
dt:check_t(tx_GPS-toe),
tcorr:(af2*dt+af1)*dt+af0,
tx_GPS:tx_RAW-tcorr,
[tx_GPS,tcorr]);
```

12.2 Positionsbestimmmung mit Zeitkorrektur

Berechnung des Empfängerstandorts aus einem beliebigen Satellitenquartett mit Berücksichtigung des Zeitfehlers.

Aufrufparameter: Wochensekunde, für welche die Position berechnet werden soll.

Funktionsergebnis: Empfängerposition in Dezimalgrad.

```
recpos_vier_t(sow,svliste):=block(
[estpos,L,svpos,d,J,sv,pr,t_GPS,tcorr,dist,A,pos],
estpos:[0.0,0.0,0.0],
L:matrix([1.0],[1.0],[1.0],[0.0]),
svpos:[],
unless abs(L[1][1])+abs(L[2][1])+abs(L[3][1])<0.1 do
    (
    d:matrix(),
    J:matrix(),
    for i:1 thru 4 do
        (
        sv:svliste[i],
        [t_GPS,tcorr]:timecorrect(sow,sv),
        pr:read_prange(sow,sv),
        svpos:satpos(t_GPS,sv),
        dist:pr-norm(svpos,estpos)-L[4][1]+v_light*tcorr,
        d:addrow(d,[dist]),
        A:[(estpos[1]-svpos[1])/norm(svpos,estpos),
           (estpos[2]-svpos[2])/norm(svpos,estpos),
           (estpos[3]-svpos[3])/norm(svpos,estpos),
            1],
        J:addrow(J,A)
        ),
    L:float(invert(J).d),
```

```
    estpos[1]:estpos[1]+L[1][1],
    estpos[2]:estpos[2]+L[2][1],
    estpos[3]:estpos[3]+L[3][1],
    if norm(estpos,[0,0,0])>10^10 then return(false)
    ),
ecef_polar_grad(estpos));
```

13 Methode der kleinsten Quadrate

13.1 Positionsbestimmung mit Berücksichtigung aller Beobachtungsdaten

Die Funktion berechnet nach der Methode der kleinsten Quadrate die Empfängerposition aus allen in der Datenstruktur zu einem bestimmten Zeitpunkt vorliegenden Daten unter Berücksichtigung der Zeitkorrektur.

Aufrufparameter: Wochensekunde, für welche die Position berechnet werden soll.

Funktionsergebnis: Empfängerposition in Dezimalgrad.

```
recpos_lsf(sow):=block(
[estpos,L,svpos,obsindex,d,J,sv,pr,t_GPS,tcorr,dist,A,pos],
estpos:[0.0,0.0,0.0],
L:matrix([1.0],[1.0],[1.0],[0.0]),
svpos:[],
obsindex:find_obsindex(sow),
unless abs(L[1][1])+abs(L[2][1])+abs(L[3][1])<0.1 do
    (
    d:matrix(),
    J:matrix(),
    for i:3 thru length(obsmatrix[obsindex]) do
        (
        sv:obsmatrix[obsindex][i][1],
        [t_GPS,tcorr]:timecorrect(sow,sv),
        pr:read_prange(sow,sv),
        svpos:satpos(t_GPS,sv),
        dist:pr-norm(svpos,estpos)-L[4][1]+v_light*tcorr,
        d:addrow(d,[dist]),
        A:[(estpos[1]-svpos[1])/norm(svpos,estpos),
           (estpos[2]-svpos[2])/norm(svpos,estpos),
           (estpos[3]-svpos[3])/norm(svpos,estpos),
            1],
        J:addrow(J,A)
        ),
    L:float(invert(transpose(J).J).transpose(J).d),
    estpos[1]:estpos[1]+L[1][1],
    estpos[2]:estpos[2]+L[2][1],
    estpos[3]:estpos[3]+L[3][1]
    ),
ecef_polar_grad(estpos));
```

14 Verbesserung des verwendeten Erdmodells

14.1 Umrechnung im Kugelmodell von ECEF- in Dezimal-Polarkoordinaten im Bogenmaß

Das Ergebnis der Umrechnung wird im Bogenmaß bestimmt. Dies ist nötig, da innerhalb der Funktion ecef_polar_ellipse() die für das Newton-Verfahren benötigten Startwerte im Bogenmaß vorliegen müssen.

Aufrufparameter: Empfängerposition in ECEF-Koordinaten.

Funktionsergebnis: Empfängerposition in dezimalen Polarkoordinaten im Bogenmaß.

```
ecef_polar_bogen(pos):=
[float(atan2(pos[3],sqrt(pos[1]^2+pos[2]^2))),
float(atan2(pos[2],pos[1])),
float(sqrt(pos[1]^2+pos[2]^2+pos[3]^2)-6378137)];
```

14.2 Umrechnung im ellipsoidalen Modell von ECEF-Koordinaten in Polarkoordinaten

Diese Funktion verbessert die Umrechnung von ECEF-Koordinaten in Polarkoordinaten dahingehend, dass für die Umrechnung das ellipsoidale Erdmodell nach WGS-84 verwendet wird.

Aufrufparameter: Empfängerposition in ECEF-Koordinaten.

Funktionsergebnis: Empfängerposition in dezimalen Polarkoordinaten im Bogenmaß.

```
ecef_polar_ellipse(ecef):=block(
[a,eps,xE,yE,zE,phi,lambda,h,gl1,gl2,gl3,start,loes,erg],
a:6378137.0,
eps:0.08181919,
[xE,yE,zE]:ecef,
gl1:(a/sqrt(1-eps^2*(sin(phi))^2)+h)*cos(phi)*cos(lambda)-xE,
gl2:(a/sqrt(1-eps^2*(sin(phi))^2)+h)*cos(phi)*sin(lambda)-yE,
gl3:(a*(1-eps^2)/sqrt(1-eps^2*(sin(phi))^2)+h)*sin(phi)-zE,
start:ecef_polar_bogen(ecef),
loes:mnewton([gl1,gl2,gl3],[phi,lambda,h],start),
erg:matrixmap(rhs,loes)[1],
float([erg[1]*180/%pi,erg[2]*180/%pi,erg[3]]));
```

14.3 Empfängerposition im verbesserten Erdmodell

Die Erde geht nicht als Kugel, sondern als Ellipsoid in die Berechnung mit ein. Dies führt zu einer deutlichen Nordverlagerung der berechneten Empfängerposition um rund 21 km.

Aufrufparameter: Wochensekunde, für welche die Position berechnet werden soll.

Funktionsergebnis: Empfängerposition in Dezimalgrad.

```
recpos_ell(sow):=block(
[estpos,L,svpos,obsindex,d,J,sv,pr,t_GPS,tcorr,dist,A,pos],
estpos:[0.0,0.0,0.0],
L:matrix([1.0],[1.0],[1.0],[0.0]),
svpos:[],
obsindex:find_obsindex(sow),
unless abs(L[1][1])+abs(L[2][1])+abs(L[3][1])<0.1 do
    (
    d:matrix(),
```

```
    J:matrix(),
    for i:3 thru length(obsmatrix[obsindex]) do
        (
        sv:obsmatrix[obsindex][i][1],
        [t_GPS,tcorr]:timecorrect(sow,sv),
        pr:read_prange(sow,sv),
        svpos:satpos(t_GPS,sv),
        dist:pr-norm(svpos,estpos)-L[4][1]+v_light*tcorr,
        d:addrow(d,[dist]),
        A:[(estpos[1]-svpos[1])/norm(svpos,estpos),
           (estpos[2]-svpos[2])/norm(svpos,estpos),
           (estpos[3]-svpos[3])/norm(svpos,estpos),
            1],
        J:addrow(J,A)
        ),
    L:float(invert(transpose(J).J).transpose(J).d),
    estpos[1]:estpos[1]+L[1][1],
    estpos[2]:estpos[2]+L[2][1],
    estpos[3]:estpos[3]+L[3][1]
    ),
ecef_polar_ellipse(estpos));
```

15 Berücksichtigung der Erddrehung

15.1 Rotationskorrektur

Während der Laufzeit des Signals zur Erde dreht sich diese ein kleines Stück weiter um ihre Achse. Die Satellitenposition beim Eintreffen des Signals ist daher eine andere als zum Zeitpunkt des Aussendens. Mit dieser Funktion wird daher die Satellitenposition auf denjenigen Ort zurückgedreht, an dem sich der Satellit zum Zeitpunkt des Aussendens des Signals befand.

Aufrufparameter: Errechnete Satellitenposition beim Eintreffen des Signals, Zeitspanne für den Signallauf

Funktionsergebnis: Zurückgedrehte Satellitenposition

```
rot_corr(satellitenpos,dt):=block(
[OMEGAe_dot,phi,satvektor,R3,new_pos],
OMEGAe_dot:7.292115147e-5,
phi:OMEGAe_dot*dt,
satvektor: matrix(
 [satellitenpos[1]],
 [satellitenpos[2]],
 [satellitenpos[3]]
),
R3: matrix(
 [cos(phi),sin(phi),0],
 [-sin(phi),cos(phi),0],
 [0,0,1]
```

```
),
new_pos:R3.satvektor,
[float(new_pos[1][1]),float(new_pos[2][1]),float(new_pos[3][1]
)]);
```

15.2 Bestimmung des Empfängerorts mit Rotationskorrektur

Aufrufparameter: Wochensekunde, für welche die Position berechnet werden soll.

Funktionsergebnis: Empfängerposition in Dezimalgrad.

```
recpos_rot(sow):=block(
[estpos,L,svpos,traveltime,svpos_rot,ob-
sindex,d,J,sv,pr,t_GPS,tcorr,dist,A,pos],
estpos:[0.0,0.0,0.0],
L:matrix([1.0],[1.0],[1.0],[0.0]),
svpos:[],
obsindex:find_obsindex(sow),
unless abs(L[1][1])+abs(L[2][1])+abs(L[3][1])<0.1 do
    (
    d:matrix(),
    J:matrix(),
    for i:3 thru length(obsmatrix[obsindex]) do
        (
        sv:obsmatrix[obsindex][i][1],
        [t_GPS,tcorr]:timecorrect(sow,sv),
        pr:read_prange(sow,sv),
        svpos:satpos(t_GPS,sv),
        traveltime:pr/v_light,
        svpos_rot:rot_corr(svpos,traveltime),
        dist:pr-norm(svpos_rot,estpos)-L[4][1]+v_light*tcorr,
        d:addrow(d,[dist]),
        A:[(estpos[1]-svpos_rot[1])/norm(svpos_rot,estpos),
           (estpos[2]-svpos_rot[2])/norm(svpos_rot,estpos),
           (estpos[3]-svpos_rot[3])/norm(svpos_rot,estpos),
            1],
        J:addrow(J,A)
        ),
    L:float(invert(transpose(J).J).transpose(J).d),
    estpos[1]:estpos[1]+L[1][1],
    estpos[2]:estpos[2]+L[2][1],
    estpos[3]:estpos[3]+L[3][1]
    ),
ecef_polar_ellipse(estpos));
```

16 Berücksichtigung der Relativität

16.1 Berechnung der relativistischen Zeitkorrektur

Aufrufparameter: Wochensekunde, für welche die Position berechnet werden soll, sowie die Nummer des Satelliten.

Funktionsergebnis: Zeitkorrektur in Sekunden.

```
dt_rel(sow,sv):=block(
[GM,F,k,delta_n,M0,ecc,sqrt_A,toe,A,tk,n0,n,M,E,dE,E_old],
GM:3.986005e14,
F:-4.442807633e-10,
k:find_ephindex(sv),
delta_n:navmatrix[k][13],
M0:navmatrix[k][14],
ecc:navmatrix[k][16],
sqrt_A:navmatrix[k][18],
toe:navmatrix[k][19],
A:sqrt_A*sqrt_A,
tk:check_t(sow-toe),
n0:sqrt(GM/A^3),
n:n0+delta_n,
M:M0+n*tk,
M:hauptwert(M),
E:M,
dE:1,
for i:1 thru 10 unless abs(dE)<1*10^-12 do
    (
    E_old:E,
    E:M+ecc*sin(E),
    dE:(mod(E-E_old,2*%pi))
    ),
E:hauptwert(E),
F*ecc*sqrt_A*sin(E));
```

16.2 Veränderte Funktion zur Zeitkorrektur

Aufrufparameter: Beobachtungszeitpunkt, Nummer des Satelliten

Funktionsergebnis: Liste mit dem korrigierten Beobachtungszeitpunkt und dem Korrekturwert aufgrund der Ungenauigkeit der Uhr sowie der relativistischen Zeitverschiebung.

```
timecorrect2(time,sv):=block(
[pseudorange,eph_list,signallaufzeit,tx_RAW,toe,dt,
af0,af1,af2,tcorr,tx_GPS],
pseudorange:read_prange(time,sv),
eph_list:navmatrix[find_ephindex(sv)],
signallaufzeit:pseudorange/v_light,
tx_RAW:check_t(time-signallaufzeit),
toe:eph_list[19],
```

```
dt:check_t(tx_RAW-toe),
af0:eph_list[8],
af1:eph_list[9],
af2:eph_list[10],
tcorr:(af2*dt+af1)*dt+af0+dt_rel(time,sv),
tx_GPS:check_t(tx_RAW-tcorr),
dt:check_t(tx_GPS-toe),
tcorr:(af2*dt+af1)*dt+af0+dt_rel(time,sv),
tx_GPS:tx_RAW-tcorr,
[tx_GPS,tcorr]);
```

16.3 Empfängerposition bestimmen unter Berücksichtigung relativistischer Effekte

Aufrufparameter: Wochensekunde, für welche die Position berechnet werden soll.

Funktionsergebnis: Empfängerposition in Dezimalgrad.

```
recpos_trel(sow):=block(
[estpos,L,svpos,traveltime,svpos_rot,obsindex,d,J,sv,pr,t_GPS,
tcorr,dist,A,pos],
estpos:[0.0,0.0,0.0],
L:matrix([1.0],[1.0],[1.0],[0.0]),
svpos:[],
obsindex:find_obsindex(sow),
unless abs(L[1][1])+abs(L[2][1])+abs(L[3][1])<0.1 do
    (
    d:matrix(),
    J:matrix(),
    for i:3 thru length(obsmatrix[obsindex]) do
        (
        sv:obsmatrix[obsindex][i][1],
        [t_GPS,tcorr]:timecorrect2(sow,sv),
        pr:read_prange(sow,sv),
        svpos:satpos(t_GPS,sv),
        traveltime:pr/v_light,
        svpos_rot:rot_corr(svpos,traveltime),
        dist:pr-norm(svpos_rot,estpos)-L[4][1]+v_light*tcorr,
        d:addrow(d,[dist]),
        A:[(estpos[1]-svpos_rot[1])/norm(svpos_rot,estpos),
           (estpos[2]-svpos_rot[2])/norm(svpos_rot,estpos),
           (estpos[3]-svpos_rot[3])/norm(svpos_rot,estpos),
            1],
        J:addrow(J,A)
        ),
    L:float(invert(transpose(J).J).transpose(J).d),
    estpos[1]:estpos[1]+L[1][1],
    estpos[2]:estpos[2]+L[2][1],
```

```
    estpos[3]:estpos[3]+L[3][1]
    ),
ecef_polar_ellipse(estpos));
```

17 Skyplot

17.1 Satellitenposition in ECEF-Koordinaten

Aufrufparameter: Wochensekunde, für welche die Position berechnet werden soll.

Funktionsergebnis: Satellitenposition in ECEf-Koordinaten.

```
recpos_trel_ecef(sow):=block(
[estpos,L,svpos,traveltime,svpos_rot,ob-
sindex,d,J,sv,pr,t_GPS,tcorr,dist,A,pos],
estpos:[0.0,0.0,0.0],
L:matrix([1.0],[1.0],[1.0],[0.0]),
svpos:[],
obsindex:find_obsindex(sow),
unless abs(L[1][1])+abs(L[2][1])+abs(L[3][1])<0.1 do
    (
    d:matrix(),
    J:matrix(),
    for i:3 thru length(obsmatrix[obsindex]) do
        (
        sv:obsmatrix[obsindex][i][1],
        [t_GPS,tcorr]:timecorrect2(sow,sv),
        pr:read_prange(sow,sv),
        svpos:satpos(t_GPS,sv),
        traveltime:pr/v_light,
        svpos_rot:rot_corr(svpos,traveltime),
        dist:pr-norm(svpos_rot,estpos)-L[4][1]+v_light*tcorr,
        d:addrow(d,[dist]),
        A:[(estpos[1]-svpos_rot[1])/norm(svpos_rot,estpos),
           (estpos[2]-svpos_rot[2])/norm(svpos_rot,estpos),
           (estpos[3]-svpos_rot[3])/norm(svpos_rot,estpos),
            1],
        J:addrow(J,A)
        ),
    L:float(invert(transpose(J).J).transpose(J).d),
    estpos[1]:estpos[1]+L[1][1],
    estpos[2]:estpos[2]+L[2][1],
    estpos[3]:estpos[3]+L[3][1]
    ),
estpos);
```

17.2 Elevation bestimmen

Schnittwinkel zwischen einer Ebene und einer diese Ebene schneidenden Geraden.

Aufrufparameter: n ... Normalenvektor der Ebene, u ... Richtungsvektor der Geraden.

Funktionsergebnis: Schnittwinkel im Bogenmaß.

```
elevation(n,u):=block(
[elev],
elev:asin(abs(u.n)/(mat_norm(u,frobenius)*mat_norm(n,fro-
benius))),
if sign(n.u)=neg then -elev else elev);
```

17.3 Parallelprojektion

Aufrufparameter: x ... zu projizierender Punkt, n ... Normalenvektor und r ... Aufpunkt der Ebene.

Funktionsergebnis: Koordinaten des Bildpunkts.

```
projektion(x,n,r):=x-(((x-r).n)/(n.n)).n;
```

17.4 Winkel zwischen zwei Geraden in einer Ebene

Aufrufparameter: u und v ... Richtungsvektoren der beiden Geraden.

Funktionsergebnis: Schnittwinkel im Bogenmaß.

```
azimut(u,v):=block(
acos((u.v)/(mat_norm(u,frobenius)*mat_norm(v,frobenius))));
```

17.5 Bestimmung von Azimut und Elevation eines Satelliten

Aufrufparameter: Wochensekunde und Satellitennummer.

Funktionsergebnis: Liste mit der Satellitennummer und dessen Azimut und Elevation im Gradmaß zum angegebenen Zeitpunkt.

```
az_elev(sow,sv):=block(
[recpos_liste,recpos_v,satpos_liste,satpos_v,
satvect,elev,projektion_satvect,nordpol,
projektion_nordpol,nordrichtung,satrichtung,
nordwinkel,osten,projektion_osten,ostrichtung,
ostwinkel,az],
recpos_liste:recpos_trel_ecef(sow),
recpos_v:transpose(matrix(recpos_liste)),
satpos_liste:satpos(sow,sv),
satpos_v:transpose(matrix(satpos_liste)),
satvect:satpos_v-recpos_v,
elev:elevation(recpos_v,satvect),
projektion_satvect:projektion(satpos_v,recpos_v,recpos_v),
nordpol:matrix([0],[0],[3*637000]),
projektion_nordpol:projektion(nordpol,recpos_v,recpos_v),
nordrichtung:projektion_nordpol-recpos_v,
satrichtung:projektion_satvect-recpos_v,
nordwinkel:azimut(nordrichtung,satrichtung),
osten:matrix([0],[3*637000],[0]),
projektion_osten:projektion(osten,recpos_v,recpos_v),
ostrichtung:projektion_osten-recpos_v,
ostwinkel:azimut(ostrichtung,satrichtung),
if ostwinkel>%pi/2 then az:2*%pi-nordwinkel
```

```
                    else az:nordwinkel,
[sv,float(grad(az)),float(grad(elev))]);
```

17.6 Umrechnung von Azimut und Elevation in kartesische Koordinaten

Aufrufparameter: Ergebnisliste der Funktion az_elev().

Funktionsergebnis: Liste mit der Satellitennummer und dessen Azimut und Elevation als kartesische Koordinaten zum Eintrag in den Skyplot.

```
az_elev_cart(plotlist):=block(
[sv,az,el,r,phi],
[sv,az,el]:plotlist,
r:abs(el/90-1),
phi:az,
x:float(r*cos(bogen(phi))),
y:float(r*sin(bogen(phi))),
[sv,y,x]);
```