Didaktisch-methodische Ausarbeitung eines Lernmoduls zum Thema GPS mit Hilfe von Matlab im Rahmen eines Modellierungstages für Schülerinnen und Schüler der Sekundarstufe II

Schriftliche Hausarbeit im Rahmen der Ersten Staatsprüfung, dem Landesprüfungsamt für Erste Staatsprüfungen für Lehrämter an Schulen vorgelegt von:

Markus Wiener

Aachen, den 18. August 2015
Gutachter: Prof. Dr. Martin Frank

Lehrstuhl für Mathematik
Center for Computational Engineering Science
Rheinisch-Westfälische Technische Hochschule Aachen

Anlage: CD mit Quelltexten und Satellitendaten

Inhaltsverzeichnis

1. Einleitung

> „Der Beitrag des Faches Mathematik zur erweiterten Allgemeinbildung beschränkt sich nicht auf die Bearbeitung verbindlicher Inhalte, sondern zielt auf den Erwerb prozess- und inhaltsbezogener mathematischer Kompetenzen.“ ([9], Ministerium für Schule und Weiterbildung des Landes Nordrhein-Westfalen, 2013, S. 14)

Das *mathematische Modellieren* ist eine solche prozessbezogene Kompetenz. TIMSS und PISA haben jedoch aufgezeigt, dass die deutschen Schüler bei Modellierungsproblemen große Schwierigkeiten haben. Insbesondere schnitten sie bei Aufgaben, die die Anwendung von Mathematik auf außermathematische Kontexte verlangen, im internationalen Vergleich nur schwach ab (vgl. [6], Leuders, 2011, S. 148)

Im Rahmen dieser Arbeit soll der Versuch unternommen werden, das mathematische Modellieren für Schülerinnen und Schüler der Sekundarstufe II am Beispiel des GPS verständlich zu machen. Dazu wird ein einfaches Modell entwickelt, um die Position des Empfängers aus den GPS-Rohdaten berechnen zu können. Es wird sich herausstellen, dass dieses Modell mit einer Abweichung von mehreren hundert Kilometern viel zu ungenau ist. In den folgenden Phasen der Modellverbesserung wird das Modell immer weiter verfeinert, bis man schließlich eine Genauigkeit von 10–20 Metern erhält. So soll an der Berechnung der Empfängerposition auch beispielhaft der Modellkreislauf erfahren werden, insbesondere die Notwendigkeit, den Kreislauf immer wieder neu zu durchlaufen und Verbesserungen am Modell vorzunehmen, bis die Position hinreichend genau ermittelt werden kann.

Als Werkzeug wird hier das Mathematiksystem MATLAB verwendet, welches genügend Hilfsmittel zur Berechnung numerischer Probleme bereitstellt und dabei noch recht einfach in der Verwendung ist. Die besondere Herausforderung liegt darin, einen vollständigen Modellierungskreislauf an nur einem einzigen Modellierungstag mit einem für die meisten Schülerinnen und Schüler unbekanntem Werkzeug erfolgreich durchzuführen. Dabei stellt sich die methodische Frage, welche dosierten Hilfestellungen nötig sind, um ein weitgehend selbstständiges Arbeiten der Schülerinnen und Schüler zu gewährleisten. Denn Menschen wachsen, wenn man ihnen die Chance gibt, etwas selbst zu erarbeiten und festigen so ihr Selbstwertgefühl entscheidend (vgl. [1], Birkenbihl, 2004, S. 46).

2. Mathematisches Modellieren und das Modellierungsprogramm CAMMP

Das mathematische *Modellieren* ist in den Kernlehrplänen des Faches Mathematik für Gymnasien und Gesamtschulen in Nordrhein-Westfalen einer der fünf prozessbezogenen Kompetenzbereiche neben *Problemlösen*, *Argumentieren*, *Kommunizieren* und *Werkzeuge nutzen* (vgl. Ministerium für Schule und Weiterbildung des Landes Nordrhein-Westfalen: [7], 2007, S. 12; [8], 2004, S. 12; [9], 2013, S. 14 f.). Es ist ein mehrstufiger Prozess, der zur Lösung realer Probleme angewendet werden kann. Dieser Prozess wird gewöhnlich als Kreislauf dargestellt, welcher in der Regel mehrfach durchlaufen werden muss, um brauchbare Resultate zu erhalten. In diesem Kapitel wird das mathematische Modellieren am Beispiel des Modellierungskreislaufs nach Blum und Leiss beschrieben und das mathematische und computergestützte Modellierungsprogramm CAMMP der RWTH-Aachen vorgestellt.

2.1. Der Modellierungskreislauf nach Blum und Leiss

In Abbildung 1 wird der Modellierungskreislauf nach Blum und Leiss aus dem Jahr 2005 dargestellt (vorliegende Version abgebildet in [5], Leiss und Tropper, 2014, S. 25). Er unterscheidet drei verschiedene Modellbegriffe und besteht aus insgesamt sieben Schritten, die der Reihe nach angewendet werden müssen, um ein reales Problem mit den Mitteln der Mathematik zu beschreiben und zu lösen. Die einzelnen Schritte lauten im Detail (vgl. [5], Leiss und Tropper, 2014, S. 24 f.):

1. Verstehen der Realsituation und Konstruktion eines Situationsmodells
2. Bilden eines Realmodells durch Vereinfachen und Strukturieren der Situation
3. Bilden eines mathematischen Modells durch Mathematisieren des Realmodells
4. Lösen der dem mathematischen Modell zugrunde liegenden innermathematischen Problemstellung
5. Interpretieren des erzeugten innermathematischen Resultats
6. Validieren der realen Lösung bezüglich der realen Problemstellung, ggf. resultierend in einem erneuten Durchlauf des Kreislaufs
7. Darlegen und Erklären der Aufgabenlösung

Im ersten Schritt muss das Problem verstanden werden. Dabei wird eine mentale Repräsentation der Realsituation gebildet, welche als *Situationsmodell* bezeichnet wird. Es bildet sich unbewusst und automatisch aus, wenn man versucht, einen realen Sachverhalt durch Begriffe zu beschreiben und zu verstehen. Es ist oft unvollständig und enthält viele für die Lösung des Problems irrelevante Informationen.

Im zweiten Schritt wird ein vereinfachtes und strukturiertes Modell der Wirklichkeit erstellt, welches als *Realmodell* bezeichnet wird. Die Vereinfachung der Wirklichkeit ist alleine schon deswegen notwendig, um die Komplexität des Problems soweit zu reduzieren, dass es mit den zur Verfügung stehenden Mitteln in einer realistischen Zeit gelöst werden kann. Damit eine innermathematische Lösung möglich wird, muss es auch so strukturiert werden, dass es mit den sprachlichen Mitteln der Mathematik formuliert werden kann. Dazu kann es nötig sein, eigene Annahmen zu treffen, die in der Problembeschreibung selber so nicht vorgegeben sind. Das Realmodell ist somit ein

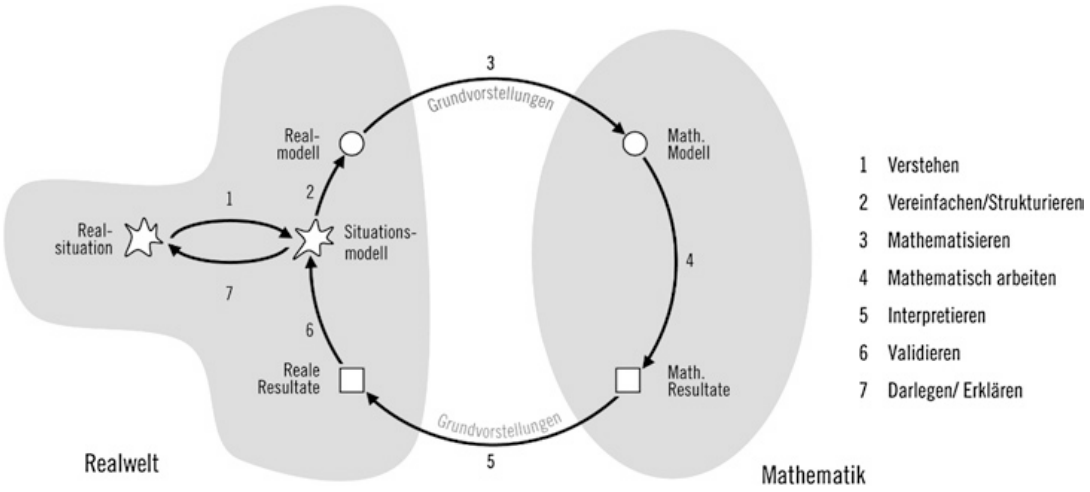


Abbildung 1: Modellierungskreislauf nach Blum und Leiss

vereinfachtes und strukturiertes Abbild der durch das Situationsmodell repräsentierten Realsituation. Es sollte so einfach wie nur möglich sein und dennoch alle wichtigen Aspekte der Realsituation enthalten.

Im dritten Schritt werden alle Bestandteile des Realmodells in die Sprache der Mathematik übersetzt. Dazu müssen Variablen definiert und Gleichungen aufgestellt werden. Auf diese Weise entsteht aus dem Realmodell ein *mathematisches Modell*, welches mit den Mitteln der Mathematik gelöst werden kann.

Im vierten Schritt kommt man durch Analyse oder Simulationen zu den mathematischen Resultaten. Dieses kann nur in den seltensten und einfachsten Fällen rein analytisch geschehen. In der Regel können die Lösungen der Gleichungen nur numerisch mit Computereinsatz approximiert werden. Wenn sogar eine Näherungslösung der Gleichungen nicht möglich ist, kann zumindest das mathematische Modell mit einem Computer simuliert werden, um auf diese Weise zu mathematischen Resultaten zu gelangen.

Im fünften Schritt werden die durch Rechnung oder Simulation erhaltenen Resultate in Bezug zur Realität gesetzt und dabei in reale Resultate umgewandelt. Anstelle von numerischen Ergebnissen erhält man bei der Umwandlung oft physikalische Größen. Dabei kann es sein, dass Ergebnisse kontextabhängig oder auf Grund von Fehlerbetrachtungen auf- oder abgerundet werden müssen.

Im sechsten Schritt muss geprüft werden, ob die realen Resultate auch wirklich eine Lösung der realitätsbezogenen Problemstellung sind. Tatsächlich kann diese Überprüfung nicht mit der Realsituation erfolgen, sondern wird mit dem Situationsmodell, dem mentalen Abbild der Realsituation, durchgeführt. Dadurch können einerseits innermathematische Fehler bei der Berechnung der mathematischen Resultate erkannt und behoben werden. Andererseits sollen in diesem Schritt auch die gewählten Modellannahmen reflektiert werden. So kann es notwendig sein, die Modellannahmen zu verändern und den Modellierungskreislauf erneut zu durchlaufen, um bessere Resultate zu erhalten. In der Regel sind mehrere Durchläufe notwendig, um das Modell soweit zu verändern, bis die realen Resultate auch eine wirkliche Lösung der realitätsbezogenen Problemstellung sind.

Sofern die realen Resultate bei der Validierung akzeptiert wurden, muss im letzten Schritt der vollständige Lösungsprozess mit allen Modellannahmen, Überlegungen und Begründungen dargelegt werden. Dabei sind alle für die Lösung relevanten Aspekte

korrekt, vollständig und verständlich darzulegen, so dass der Lösungsprozess anhand dieser Erklärungen nachvollziehbar wird. Auf diese Weise kann eine dritte Person auch Fehlvorstellungen im Situationsmodell erkennen, wodurch es letztendlich nochmal zu einer weiteren Korrektur der Modellannahmen und zu einem erneuten Durchlauf des Modellierungskreises kommen kann.

2.2. Das Modellierungsprogramm CAMMP

Das computergestützte mathematische Modellierungsprogramm CAMMP (*Computational and Mathematical Modeling Program*[1]) ist ein Education Lab der RWTH Aachen, ein Schülerlabor, wo Schülerinnen und Schüler die Grundlagen der mathematischen Modellierung an realitätsbezogenen Problemstellungen erlernen können. Sie arbeiten dabei selbständig an realen Problemen aus Industrie und Forschung mit Hilfe von mathematischer Modellierung und Computersimulationen und werden von wissenschaftlichen Betreuern unterstützt. Dadurch erhalten sie einen Einblick in die Berufswelt von Mathematikern, Informatikern und Ingenieuren und lernen Werkzeuge zur Problemlösung kennen und nutzen. Durch die Arbeit in Kleingruppen erweitern sie darüber hinaus ihre Teamfähigkeit und ihr Kommunikationsvermögen.

CAMMP wird organisiert vom Lehrstuhl Mathematik CCES (Prof. Dr. Martin Frank und Dr. Christina Roeckerath), der Arbeitsgruppe Molecular Simulations und Transformations (Prof. Dr. Ahmed E. Ismail) und der Graduiertenschule AICES (Dr. Nicole Faber). Im Rahmen von CAMMP gibt es derzeit zwei verschiedene Angebote:

- Der *CAMMP day*, ein computergestützter mathematischer Modellierungstag
- Die *CAMMP week*, eine computergestützte mathematische Modellierungswoche

Bei einem Modellierungstag werden Schülerinnen und Schüler in einem eintägigen Workshop anhand praxisorientierter Fragestellungen in die Grundlagen der mathematischen Modellierung eingeführt. Dabei arbeiten sie zu zweit mit dem Mathematiksystem MATLAB an einem realen Problem und werden von wissenschaftlichen Beratern unterstützt. Durch die zeitliche Begrenzung auf einen Arbeitstag können sie nicht alle Modellierungsschritte selber durchführen, sondern arbeiten entlang eines didaktisch aufbereiteten Arbeitsauftrags an einem MATLAB-Programm und beschäftigen sich mit einzelnen Aspekten der Modellierung.

Bei einer Modellierungswoche lösen die Schülerinnen und Schüler innerhalb von fünf Tagen reale Probleme von Firmen und Universitätsinstituten mit Hilfe von mathematischer Modellierung und Computereinsatz. Sie arbeiten in Kleingruppen an einem Problem und werden dabei von einem Wissenschaftler beraten und unterstützt. Hier müssen sie einen vollständigen Modellierungsprozess selber durchführen. Ihre Ergebnisse werden am letzten Tag von ihnen selbst vor den Problemstellern präsentiert.

[1] `http://www.cammp.rwth-aachen.de/`

3. Funktionsweise der Positionsbestimmung

Das Global Positioning System (GPS, offiziell NAVSTAR GPS) ist ein globales Navigationssatellitensystem zur Positionsbestimmung. Es wird mittlerweile in vielen Bereichen eingesetzt und ist aus unserem Alltag nicht mehr wegzudenken. In diesem Kapitel wird die Mathematik erklärt, die zur Positionsbestimmung notwendig ist.

3.1. Das Global Positioning System

Das Global Positioning System ist der Nachfolger des ersten Satellitensystems Transit, welches von 1964 bis 1996 von der US-Marine betrieben wurde. Es wurde seid 1973 vom US-Verteidigungsministerium entwickelt und ist seit 1993 im Betrieb. Das System wurde 1995 für voll funktionsfähig erklärt und steht seitdem für militärische und zivile Nutzung zur Verfügung. Für den Betrieb des GPS werden mindestens 24 funktionsfähige Satelliten benötigt. Sie umkreisen in 20 000 km Höhe die Erde und senden jede Millisekunde Nachrichten aus, welche die Sendezeiten und Daten zur Positionsbestimmung enthalten. Durch die besondere Anordnung der Satellitenbahnen können von jedem Ort der Erde zu jedem Zeitpunkt mindestens vier Satelliten empfangen werden. Aus den Daten zur Positionsbestimmung lassen sich die Koordinaten der Satelliten zu den Sendezeitpunkten berechnen. Durch Vergleich der übertragenen Sendezeiten mit den gemessenen Empfangszeiten können die Signallaufzeiten und damit die Entfernungen zu den Satelliten berechnet werden. Liegen die Koordinaten und Entfernungen von mindestens drei Satelliten vor, lässt sich die Position des Empfängers auf der Erde berechnen. Vor der Abschaltung der künstlichen Signalverschlechterung am 2. Mai 2000 waren zivile Messungen nur mit Fehlern über 100 Meter möglich, seitdem beträgt der Fehler ziviler Messungen nur noch 10 bis 15 Meter. Mit den verschlüsselten militärischen Signalen wäre eine wesentlich genauere Positionsbestimmung möglich. Für eine bessere Präzision mit den zivilen Signalen können Korrektursignale von ortsfesten GPS-Empfängern (Differtial-GPS) und Trägerphasenmessungen verwendet werden (vgl. [11], Zogg, 2009, S. 100 ff.).

3.1.1. Die GPS-Zeit und das ECEF-Koordinatensystem

Die GPS-Zeit ist das Zeitsystem für das Global Positioning System. Im Gegensatz zur koordinierten Weltzeit (UTC) gibt es bei der GPS-Zeit keine Schaltsekunden. Deshalb unterscheidet sie sich von der koordinierten Weltzeit um einige Sekunden. Die GPS-Zeit startete in der Nacht von Samstag auf Sonntag am 6. Januar 1980 um 0:00 Uhr mit der GPS-Woche 0 und war zu diesem Zeitpunkt identisch mit der koordinierten Weltzeit. Jeden Sonntag um 0:00 Uhr beginnt seitdem eine neue GPS-Woche. Innerhalb einer Woche wird die Zeit in Sekunden seit Wochenbeginn angegeben. Eine GPS-Woche hat immer genau 604 800 Sekunden.

Für das GPS wird das geozentrische Koordinatensystem ECEF (Earth-Centered, Earth-Fixed) verwendet, welches sich mit der Erde mitdreht (siehe Abbildung 2). In diesem Koordinatensystem hat ein auf der Erde ruhender Empfänger zu allen Zeiten dieselben Koordinaten. Der Ursprung dieses Koordinatensystems liegt im Schwerpunkt der Erde und die z-Achse zeigt entlang der mittleren Rotationsachse nach Norden. Die x-Achse wird durch den Äquator und durch den Nullmeridian festgelegt und die y-Achse vervollständigt die beiden Achsen in Höhe des Äquators zu einem Rechtssystem.

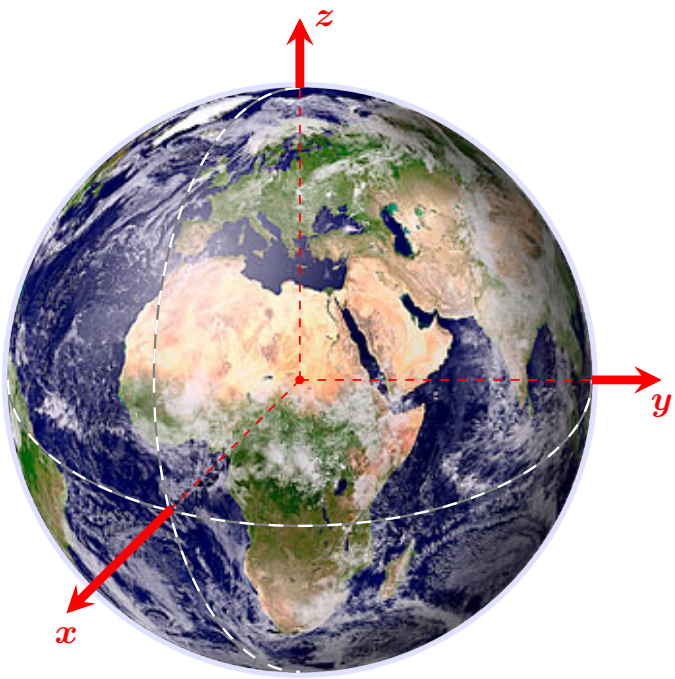


Abbildung 2: Das ECEF-System

3.1.2. Pseudoentfernungen, Ephemeriden und das RINEX-Format

Die Positionsbestimmung beruht darauf, dass zu einem Zeitpunkt die Koordinaten und die Entfernungen von drei Satelliten bekannt sind. Um die Entfernungen der Satelliten bestimmen zu können, wird jede Millisekunde eine Nachricht von den GPS-Satelliten ausgesendet, welche den Sendezeitpunkt enthält. Im Empfänger werden durch Vergleich der Empfangs- und Sendezeiten die Laufzeiten der Signale und die Entfernungen der Satelliten berechnet. Da die Ausbreitungsgeschwindigkeit der Signale mit annähernd 300 000 000 Meter pro Sekunde (Lichtgeschwindigkeit) extrem groß ist, erzeugen schon kleinste Fehler bei der Zeitmessung sehr große Abweichungen bei den Entfernungen. Diese aus Zeitdifferenzen berechneten Entfernungen werden *Pseudoentfernungen* genannt und sind immer mit Fehlern behaftet. Die Berechnung der Pseudoentfernungen findet bereits im GPS-Empfänger statt, und für jede Nachricht werden die Empfangszeitpunkte zusammen mit den Pseudoentfernungen, den Signalstärken, den Daten für die Positionsbestimmung der Satelliten, den Trägerphasen und weiteren Daten für Fehlerkorrekturen der Satellitenuhren abgespeichert. Da jeder Empfänger diese Daten anders abspeichert, ist ein empfängerunabhängiges Austauschformat für GPS-Rohdaten festgelegt worden. Beim sogenannten RINEX-Format (Receiver Independent Exchange Format) werden die empfangenen Satellitendaten in zwei Dateien getrennt abgespeichert. Die *Observations*-Datei enthält die Empfangszeitpunkte, die Pseudoentfernungen, die Trägerphasen und die Signalstärken. In der *Navigations*-Datei befinden sich die Daten zur Positionsbestimmung der Satelliten und zur Korrektur der Satellitenuhrfehler. Die Daten zur Positionsbestimmung bestehen aus 16 Parameter pro Satellit und beschreiben die Form und Lage der Ellipsen im Raum, auf der sich die einzelnen Satelliten bewegen, sowie die Bewegung der Satelliten auf diesen Ellipsen. Zur Korrektur der Satellitenuhrfehler werden vier Parameter pro Satellit benötigt.

M_0	Mean Anomaly at Reference Time (semi-circles)
Δn	Mean Motion Difference From Computed Value (semi-circles/sec)
e	Eccentricity (dimensionless)
$\sqrt{A}$	Square Root of the Semi-Major Axis $\left(\sqrt{\text{meters}}\right)$
Ω_0	Longitude of Ascending Node of Orbit Plane at Weekly Epoch (semi-circles)
i_0	Inclination Angle at Reference Time (semi-circles)
ω	Argument of Perigee (semi-circles)
$\dot{\Omega}$	Rate of Right Ascension (semi-circles/sec)
i_{dot}	Rate of Inclination Angle (semi-circles/sec)
c_{uc}	Amplitude of the Cosine Harmonic Correction Term to the Argument of Latitude (rad)
c_{us}	Amplitude of the Sine Harmonic Correction Term to the Argument of Latitude (rad)
c_{rc}	Amplitude of the Cosine Harmonic Correction Term to the Orbit Radius (meters)
c_{rs}	Amplitude of the Sine Harmonic Correction Term to the Orbit Radius (meters)
c_{ic}	Amplitude of the Cosine Harmonic Correction Term to the Angle of Inclination (rad)
c_{is}	Amplitude of the Sine Harmonic Correction Term to the Angle of Inclination (rad)
t_{oe}	Reference Time Ephemeris (seconds)
a_{f0}	SV Clock Bias Correction Coefficient (seconds)
a_{f1}	SV Clock Drift Correction Coefficient (sec/sec)
a_{f2}	SV Clock Drift Rate Correction Coefficient (sec/sec^2)
t_{oc}	SV Clock Reference Time (seconds)
p_{rn}	SV PRN Number

Tabelle 1: Ephemeridenparameter

Zusammen mit einer Identifikationsnummer werden für jeden Satelliten genau 21 Parameter benötigt, welche als *Ephemeriden* bezeichnet werden (vgl. [2], Borre, 2003, S. 48). Diese Parameter werden alle 2 Stunden korrigiert, da die Satellitenbahnen keine perfekten Ellipsen sind und die Parameter deshalb ständig angepasst werden müssen. Die Bedeutung der verschiedenen Ephemeridenparameter ist in Tabelle 1 aufgeführt und ist den Interface Spezifikationen IS-GPS-200 entnommen (vgl. dazu Table 20-I, Table 20-II und Table 20-III in [3], Global Positioning System Directorate, 2013, S. 95, 102 und 103).

3.1.3. Die Satellitendaten für die Positionsberechnung

Für die Positionsbestimmung werden in diesem Lernmodul nicht die originalen RINEX-Dateien verwendet, da diese schon die im Empfänger berechneten Pseudoentfernungen enthalten und da sie durch die Auftrennung in zwei Dateien schwierig zu handhaben sind. Stattdessen werden die Sendezeiten der Nachrichten aus den Pseudoentfernungen rekonstruiert und anstelle der in den Observationsdaten hinterlegten Pseudoentfernungen verwendet. Die übrigen Satellitendaten werden aus den RINEX-Dateien ausgelesen und zu einer einfachen Liste von Datensätzen zusammengefasst. Jeder Datensatz bezieht sich dabei auf einen bestimmten Messzeitpunkt und besteht aus folgenden Daten:

- Die Anzahl aller gleichzeitig empfangener Satellitennachrichten
- Die Nummer der GPS-Woche, auf die sich die Zeitangaben und die Ephemeridenparameter beziehen.
- Der Empfangszeitpunkt der Nachrichten in Sekunden seit Wochenbeginn
- Die Sendezeiten dieser Nachrichten in Sekunden seit Wochenbeginn
- Die Signalstärken dieser Nachrichten in dBHz (Signalstärken unterhalb von 12 dBHz sind sehr schwach, Signalstärken ab 54 dBHz sind sehr stark, vgl. [4],

Gurtner und Estey, 2009, S. 10).

- Die Ephemeriden der zugehörigen Satelliten

Ausgehend von einem solchen Datensatz soll nun der Standort des Empfängers auf der Erde ermittelt werden.

3.2. Ein erstes Modell

Zuerst werden drei Satelliten ausgewählt und folgende Einträge aus dem Datensatz kopiert:

- Der Empfangszeitenvektor $\vec{t_E}$ mit den drei Komponenten t_{E_1}, t_{E_2} und t_{E_3}
- Der Sendezeitenvektor $\vec{t_S}$ mit den drei Komponenten t_{S_1}, t_{S_2} und t_{S_3}
- Die Ephemeridenmatrix E_{ph} mit den 21×3 Komponenten $E_{ph_{1,1}}$ bis $E_{ph_{21,3}}$

Die Empfangs- und Sendezeiten sind in Sekunden seit dem Beginn der GPS-Woche angegeben. Im Regelfall werden sowohl die GPS-Wochen der empfangenen Nachrichten als auch die Empfangszeiten t_{E_1}, t_{E_2} und t_{E_3} alle gleich sein, da die Nachrichten in einem Datensatz alle gleichzeitig empfangen worden sind. In einer Nacht von Samstag auf Sonntag wechselt jedoch die GPS-Woche. In so einer Nacht kann es vorkommen, dass den einzelnen Ephemeriden verschiedene GPS-Wochen zugeordnet sind. Dann unterscheiden sich die Empfangszeiten t_{E_1}, t_{E_2} und t_{E_3}, da sie sich auf verschiedene GPS-Wochen beziehen. Die Ermittlung der Empfängerposition geschieht in vier aufeinander folgenden Schritten:

- Im ersten Schritt werden die Satellitenkoordinaten zu den Sendezeiten aus den Ephemeridenparametern berechnet.
- Im zweiten Schritt werden die Pseudoentfernungen aus den Sendezeiten und den Empfangszeiten berechnet.
- Im dritten Schritt werden die Empfängerkoordinaten aus den Satellitenkoordinaten und den Pseudoentfernungen berechnet.
- Im vierten Schritt werden die Empfängerkoordinaten in geographische Koordinaten umgewandelt.

3.2.1. Schritt 1: Berechnen der Satellitenkoordinaten

Im ersten Modellschritt werden die Koordinaten der drei Satelliten berechnet. Ihre Koordinaten werden zu den Sendezeiten der Nachrichten benötigt und lassen sich aus dem Vektor $\vec{t_S}$ und der Ephemeridenmatrix E_{ph} berechnen. Der Algorithmus zur Berechnung der Koordinaten x_k, y_k und z_k des k-ten Satelliten ist in Tabelle 2 aufgeführt und ist den Interface Spezifikationen IS-GPS-200 entnommen (vgl. Table 20-IV in [3], Global Positioning System Directorate, 2013, S. 104 f.).

$\mu = 3\ 986005 \times 10^{14}\ \text{meters}^3/\text{sec}^2$	GPS value of the earth's gravitational constant
$\dot{\Omega}_e = 7\ 2921151467 \times 10^{-5}\ \text{rad/sec}$	GPS value of the earth's rotation rate
$A = \left(\sqrt{A}\right)^2$	Semi-major axis
$n_0 = \sqrt{\mu/A^3}$	Computed mean motion
$t_k = t - t_{oe}$	Time from ephemeris reference epoch
$n = n_0 + \Delta n$	Corrected mean motion
$M_k = M_0 + nt_k$	Mean anomaly
$E_k - e \sin E_k = M_k$	Kepler's Equation for Eccentric Anomaly
$\tan v_k = \sqrt{1-e^2} \sin E_k / (cosE_k - e)$	True Anomaly
$\Phi_k = v_k + \omega$	Argument of Latitude
$\delta u_k = c_{us} \sin 2\Phi_k + c_{uc} \cos 2\Phi_k$	Argument of Latitude Correction
$\delta r_k = c_{rs} \sin 2\Phi_k + c_{rc} \cos 2\Phi_k$	Radius Correction
$\delta i_k = c_{is} \sin 2\Phi_k + c_{ic} \cos 2\Phi_k$	Inclination Correction
$u_k = \Phi_k + \delta u_k$	Corrected Argument of Latitude
$r_k = A\,(1 - e \cos E_k) + \delta r_k$	Corrected Radius
$i_k = i_0 + \delta i_k + i_{dot} t_k$	Corrected Inclination
$x'_k = r_k \cos u_k$ $y'_k = r_k \sin u_k$	Positions in orbital plane
$\Omega_k = \Omega_0 + \left(\dot{\Omega} - \dot{\Omega}_e\right) t_k - \dot{\Omega}_e t_{oe}$	Corrected longitude of ascending node
$x_k = x'_k \cos \Omega_k - y'_k \cos i_k \sin \Omega_k$ $y_k = x'_k \sin \Omega_k + y'_k \cos i_k \cos \Omega_k$ $z_k = y'_k \sin i_k$	Earth-fixed coordinates

Tabelle 2: Algorithmus zur Berechnung der Satellitenkoordinaten

Bemerkungen

- Obwohl der Gravitationsparameter μ mittlerweile genauer bekannt ist, muss für das GPS aus technischen Gründen der Wert $\mu = 3{,}986005 \times 10^{14}\ \text{m}^3/\text{s}^2$ verwendet werden. Dieser Wert ist in Millionen von GPS-Empfängern fest einprogrammiert und wird von den Kontrollstationen bei der Ermittlung der Ephemeridenparameter berücksichtigt.

- Die Kepler-Gleichung $E_k - e \sin E_k = M_k$ kann nicht direkt nach E_k aufgelöst werden. Man kann die exzentrische Anomalie E_k jedoch durch eine einfache Iteration berechnen:

$$\begin{aligned} E_k^{(0)} &= M_k \\ E_k^{(n)} &= M_k + e \sin E_k^{(n-1)} \quad \text{für } n \in \mathbb{N} \setminus \{0\} \end{aligned}$$

  Die Folge $(E_k^{(n)})_n$ konvergiert gegen die exzentrische Anomalie E_k.

- Zum Berechnung der wahren Anomalie v_k sind entweder Fallunterscheidungen oder die Verwendung der atan2-Funktion nötig (vgl. dazu die Definition des atan2 in Gleichung (9) auf Seite 16):

$$v_k = \text{atan2}\left(\sqrt{1-e^2} \sin E_k,\ \cos E_k - e\right)$$

Im Folgenden werden alle Satellitenkoordinaten zu den drei Vektoren $\vec{x}_S$, $\vec{y}_S$ und $\vec{z}_S$ zusammengefasst. Die Koordinaten des k-ten Satelliten werden mit x_{S_k}, y_{S_k} und z_{S_k} bezeichnet.

3.2.2. Schritt 2: Berechnen der Pseudoentfernungen

Im zweiten Modellschritt werden die Pseudoentfernungen berechnet. Dazu müssen die Laufzeiten der Signale mit der Lichtgeschwindigkeit c multipliziert werden. Da

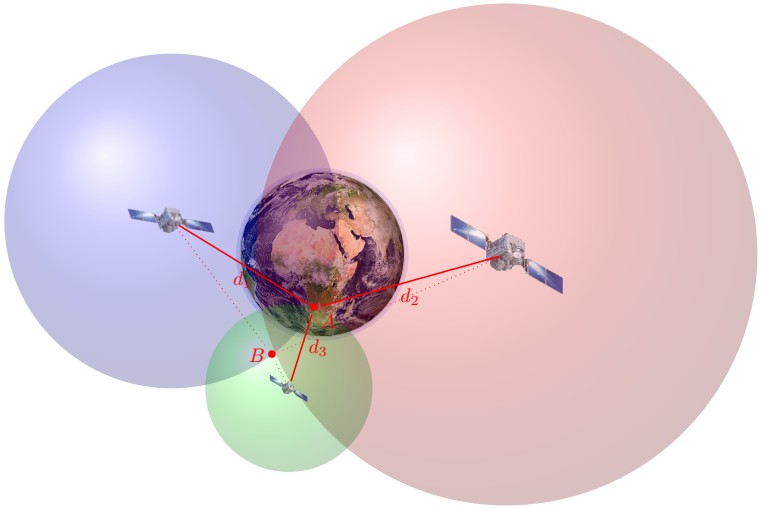


Abbildung 3: Berechnung der Empfängerkoordinaten

die Empfangs- und Sendezeiten der Nachrichten in Sekunden angegeben sind und die Pseudoentfernungen in Meter benötigt werden, muss für die Lichtgeschwindigkeit c der Wert 299 792 458 Meter pro Sekunde verwendet werden. Der Vektor der Pseudoentfernungen $\vec{d_S}$ berechnet sich aus dem Vektor der Empfangszeiten $\vec{t_E}$ und dem Vektor der Sendezeiten $\vec{t_S}$ durch folgende Gleichung:

$$\vec{d_S} = \left(\vec{t_E} - \vec{t_S}\right) \cdot 299792458\,\mathrm{m/s} \tag{1}$$

Die Pseudoentfernung zum k-ten Satelliten wird mit d_{S_k} bezeichnet.

3.2.3. Schritt 3: Berechnen der Empfängerkoordinaten

Im dritten Modellschritt werden die Empfängerkoordinaten aus den Satellitenkoordinaten und den Pseudoentfernungen berechnet. Dazu muss ein nichtlineares Gleichungssystem für die unbekannten Empfängerkoordinaten x_E, y_E und z_E aufgestellt und gelöst werden.

Man kann sich um jeden Satelliten eine Kugel mit dem Radius der zugehörigen Pseudoentfernung vorstellen. Bei drei Satelliten bekommt man somit drei Kugeln mit den Radien d_{S_1}, d_{S_2} und d_{S_3}, die sich gegenseitig durchdringen. Der Empfänger muss sich auf allen drei Kugeln gleichzeitig befinden. Da sich zwei Kugeln in einem Kreis schneiden und dieser Kreis mit der dritten Kugel genau zwei Schnittpunkte hat, gibt es nur zwei mögliche Empfängerpositionen A und B. Von diesen beiden Positionen befindet sich eine in Nähe der Erdoberfläche (A) und die andere weiter von der Erde entfernt im Weltall (B).

Die Gleichung der k-ten Kugel mit Mittelpunkt $(x_{S_k}, y_{S_k}, z_{S_k})$ und Radius d_{S_k} lautet:

$$\sqrt{(x_{S_k} - x)^2 + (y_{S_k} - y)^2 + (z_{S_k} - z)^2} = d_{S_k}$$

Dabei ist der Punkt (x, y, z) ein beliebiger Punkt der Kugeloberfläche. Für die gesuchten Empfängerkoordinaten x_E, y_E und z_E erhält man mit Hilfe von drei Satelliten

folgendes nichtlineares Gleichungssystem:

$$\begin{aligned}
\sqrt{(x_{S_1} - x_E)^2 + (y_{S_1} - y_E)^2 + (z_{S_1} - z_E)^2} &= d_{S_1} \\
\sqrt{(x_{S_2} - x_E)^2 + (y_{S_2} - y_E)^2 + (z_{S_2} - z_E)^2} &= d_{S_2} \\
\sqrt{(x_{S_3} - x_E)^2 + (y_{S_3} - y_E)^2 + (z_{S_3} - z_E)^2} &= d_{S_3}
\end{aligned} \tag{2}$$

Die Lösung dieses Gleichungssystem lässt sich numerisch durch ein Iterationsverfahren berechnen. Ein Standardverfahren zur numerischen Lösung nichtlinearer Gleichungssysteme mit genauso vielen Gleichungen wie Unbekannten ist das Newton-Verfahren. Es benötigt einen Startpunkt in der Nähe einer Nullstelle und nähert sich bei jedem Iterationsschritt dieser Nullstelle immer mehr an. Deswegen muss das Gleichungssystem (2) als Nullstellenproblem dargestellt werden:

$$\begin{aligned}
\sqrt{(x_{S_1} - x_E)^2 + (y_{S_1} - y_E)^2 + (z_{S_1} - z_E)^2} - d_{S_1} &= 0 \\
\sqrt{(x_{S_2} - x_E)^2 + (y_{S_2} - y_E)^2 + (z_{S_2} - z_E)^2} - d_{S_2} &= 0 \\
\sqrt{(x_{S_3} - x_E)^2 + (y_{S_3} - y_E)^2 + (z_{S_3} - z_E)^2} - d_{S_3} &= 0
\end{aligned} \tag{3}$$

Das Newton-Verfahren muss somit auf folgende drei Funktionen f_1, f_2 und f_3 angewendet werden:

$$\begin{aligned}
f_1(x, y, z) &= \sqrt{(x_{S_1} - x)^2 + (y_{S_1} - y)^2 + (z_{S_1} - z)^2} - d_{S_1} \\
f_2(x, y, z) &= \sqrt{(x_{S_2} - x)^2 + (y_{S_2} - y)^2 + (z_{S_2} - z)^2} - d_{S_2} \\
f_3(x, y, z) &= \sqrt{(x_{S_3} - x)^2 + (y_{S_3} - y)^2 + (z_{S_3} - z)^2} - d_{S_3}
\end{aligned} \tag{4}$$

Als Startpunkt für das Iterationsverfahren sollte der Erdmittelpunkt (0,0,0) gewählt werden. Damit stellt man sicher, dass das Verfahren gegen den gesuchten erdnahen Punkt A und nicht gegen den entfernteren Punkt B konvergiert.

3.2.4. Schritt 4: Umrechnen in geographische Koordinaten

Um die Position des Empfängers auf einer Karte anzuzeigen, müssen die Empfängerkoordinaten in geographische Koordinaten umgerechnet werden. Die geographische Breite ϕ ist die im Winkelmaß Grad angegebene nördliche oder südliche Entfernung eines Punktes der Erdoberfläche vom Äquator und nimmt Werte von $-90°$ am Südpol bis $90°$ am Nordpol an. Die geographische Länge λ ist die in Grad angegebene östliche oder westliche Entfernung eines Punktes der Erdoberfläche vom Nullmeridian und nimmt Werte von $-180°$ in westlicher Richtung bis $180°$ in östlicher Richtung an. Da die Erde in diesem Modellschritt durch eine Kugel modelliert wird, gibt die geographische Höhe h die Höhe in Meter über der Kugeloberfläche an. Als Radius der Kugel sollte der Äquatorradius $r = 6378137{,}0$ Meter gewählt werden. Dabei gelten folgende Beziehungen zwischen den kartesischen Koordinaten x, y, z und den geographischen Koordinaten h, ϕ, λ:

$$\begin{aligned}
x &= (r + h) \cos\phi \cos\lambda \\
y &= (r + h) \cos\phi \sin\lambda \\
z &= (r + h) \sin\phi
\end{aligned} \tag{5}$$

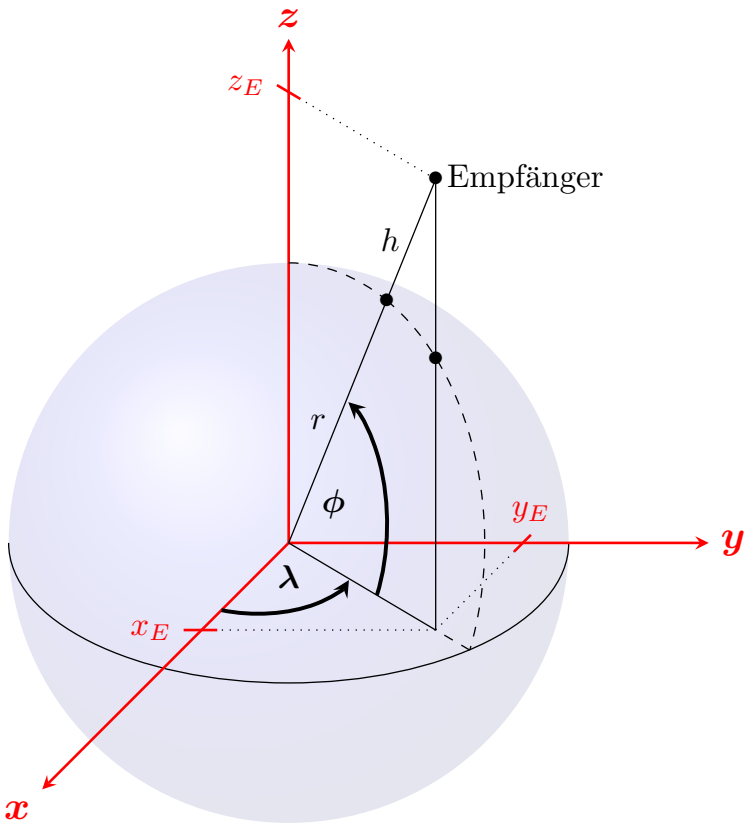


Abbildung 4: Geographische Koordinaten des Empfängers

Für Empfängerkoordinaten x_E, y_E, z_E mit $x_E > 0$ gilt folgende Umkehrung:

$$\begin{aligned} h &= \sqrt{x_E^2 + y_E^2 + z_E^2} - r \\ \phi &= \arctan\left(z_E / \sqrt{x_E^2 + y_E^2}\right) \\ \lambda &= \arctan\left(y_E / x_E\right) \end{aligned} \tag{6}$$

Die Formel für λ ist jedoch nicht korrekt für negative Werte von x_E. Für Empfängerkoordinaten x_E, y_E, z_E mit $x_E < 0$ und $y_E \geq 0$ lautet die richtige Umkehrung:

$$\begin{aligned} h &= \sqrt{x_E^2 + y_E^2 + z_E^2} - r \\ \phi &= \arctan\left(z_E / \sqrt{x_E^2 + y_E^2}\right) \\ \lambda &= \arctan\left(y_E / x_E\right) + 180^\circ \end{aligned} \tag{7}$$

Für Empfängerkoordinaten x_E, y_E, z_E mit $x_E < 0$ und $y_E < 0$ gilt stattdessen:

$$\begin{aligned} h &= \sqrt{x_E^2 + y_E^2 + z_E^2} - r \\ \phi &= \arctan\left(z_E / \sqrt{x_E^2 + y_E^2}\right) \\ \lambda &= \arctan\left(y_E / x_E\right) - 180^\circ \end{aligned} \tag{8}$$

Mit Hilfe der atan2-Funktion lassen sich diese Fälle zusammenfassen. Der atan2 : $\mathbb{R} \times \mathbb{R} \to (-180°, 180°]$ ist dabei wie folgt definiert:

$$\operatorname{atan2}(y,x) := \begin{cases} \arctan(y/x) & x > 0 \\ \arctan(y/x) + 180° & x < 0, y \geq 0 \\ \arctan(y/x) - 180° & x < 0, y < 0 \\ +90° & x = 0, y > 0 \\ -90° & x = 0, y < 0 \\ 0° & x = 0, y = 0 \end{cases} \tag{9}$$

Für Empfängerkoordinaten x_E, y_E, z_E mit $(x_E, y_E) \neq (0,0)$ gilt somit:

$$\begin{aligned} h &= \sqrt{x_E^2 + y_E^2 + z_E^2} - r \\ \phi &= \arctan\left(z_E / \sqrt{x_E^2 + y_E^2}\right) \\ \lambda &= \operatorname{atan2}(y_E, x_E) \end{aligned} \tag{10}$$

Für beliebige Empfängerkoordinaten x_E, y_E, z_E lautet die korrekte Umkehrung:

$$\begin{aligned} h &= \sqrt{x_E^2 + y_E^2 + z_E^2} - r \\ \phi &= \operatorname{atan2}\left(z_E, \sqrt{x_E^2 + y_E^2}\right) \\ \lambda &= \operatorname{atan2}(y_E, x_E) \end{aligned} \tag{11}$$

3.3. Modellverbesserungen

Nach den ersten vier Modellschritten hat man eine Empfängerposition errechnet, welche in der Regel eine Abweichung von mehreren hundert Kilometern zur wirklichen Empfängerposition hat. Um eine bessere Lösung zu erhalten, müssen einige Annahmen bei der Modellbildung überdacht werden. Die Modellschritte beruhen unter anderem auf folgenden Annahmen:

1. Die Uhren in den Satelliten laufen synchron mit der GPS-Zeit.
2. Die Uhr im Empfänger läuft synchron mit der GPS-Zeit.
3. Die Erde ist kugelförmig.

Erst durch Verbesserungen dieser drei Annahmen erhält man eine brauchbare Lösung mit einer Abweichung von nur wenigen Metern zur wirklichen Empfängerposition.

3.3.1. Modellverbesserung 1: Fehler der Satellitenuhren

In den Satelliten befinden sich präzise Atomuhren mit einer Gangungenauigkeit von wenigen Nanosekunden pro Monat. Man sollte somit annehmen, dass die Fehler der Satellitenuhren vernachlässigbar wären. Allerdings unterliegt die Zeit in den Satelliten den Effekten der Relativitätstheorie. Sie hängt einerseits vom Gravitationsfeld der Erde und andererseits von der Geschwindigkeit der Satelliten ab. Ein geringeres Gravitationsfeld lässt die Zeit schneller vergehen als auf der Erde, die Bewegung der Satelliten verzögert die Zeit dagegen etwas. Dabei überwiegt der gravitative Effekt

um mehr als das sechsfache, wodurch die Zeit in den Satelliten insgesamt schneller vergeht als auf der Erde.

Zur Korrektur der Satellitenuhrfehler finden sich in den Ephemeriden die drei Parameter a_{f0}, a_{f1} und a_{f2} und die Referenzzeit t_{oc}. Der Zeitfehler Δt einer Satellitenuhr hängt dabei von der Zeit t ab und ergibt sich nach den Interface Spezifikationen IS-GPS-200 wie folgt (vgl. [3], Global Positioning System Directorate, 2013, S. 95 f.):

$$\Delta t = a_{f0} + a_{f1}\,(t - t_{oc}) + a_{f2}\,(t - t_{oc})^2 + \Delta t_r \tag{12}$$

Δt_r ist ein relativistischer Korrekturterm, der die Ellipsenbahn des Satelliten berücksichtigt. Würde der Satellit auf einer Kreisbahn mit konstanter Geschwindigkeit die Erde umkreisen, wäre das Verhältnis von Zeitbeschleunigung aufgrund des Gravitationsfeldes zu Zeitverzögerung aufgrund der Bewegung immer konstant. Auf einer Ellipsenbahn ist dieses Verhältnis jedoch niemals konstant, da sich einerseits die Geschwindigkeit des Satelliten und andererseits der Abstand zur Erde ständig verändert. Die Veränderung des Verhältnisses von Zeitbeschleunigung zu Zeitverzögerung wird durch den Term Δt_r berücksichtigt. Dieser hängt ebenfalls von der Zeit t ab und wird wie folgt berechnet:

$$\Delta t_r = F e \sqrt{A} \sin E_k \tag{13}$$

Die Konstante $F = -4{,}42807633 \times 10^{-10}\ \mathrm{s}/\sqrt{\mathrm{m}}$ kann aus dem Gravitationsparameter $\mu = 3{,}986005 \times 10^{14}\ \mathrm{m}^3/\mathrm{s}^2$ und der Lichtgeschwindigkeit $c = 299792458\ \mathrm{m/s}$ berechnet werden:

$$F = -2\sqrt{\mu}/c^2 \tag{14}$$

Die numerische Exzentrizität e und die Wurzel der großen Halbachse $\sqrt{A}$ sind in den Ephemeriden hinterlegt, die zeitabhängige exzentrische Anomalie E_k muss wie bei der Berechnung der Satellitenkoordinaten aus der Kepler-Gleichung ermittelt werden (siehe Tabelle 2 auf Seite 12).

Der Fehler der Satellitenuhr sollte zur Sendezeit der Nachricht berechnet werden. Da der Fehler vom Sendezeitpunkt abhängt und da der Sendezeitpunkt fehlerbehaftet ist, muss der Satellitenuhrfehler iterativ bestimmt werden. Um das zu veranschaulichen wird die Gleichung (12) zur Berechnung der Satellitenuhrfehler durch eine Funktion f in Abhängigkeit der Zeit t dargestellt:

$$f(t) = a_{f0} + a_{f1}\,(t - t_{oc}) + a_{f2}\,(t - t_{oc})^2 + \Delta t_r(t) \tag{15}$$

Weiterhin werden folgende Bezeichnungen eingeführt:

t_S	:	Falsche Sendezeit des Satelliten (bekannt)
$\Delta t_S = f(t_S)$	:	Fehler der Satellitenuhr zur falschen Sendezeit t_S (bekannt)
t_S^*	:	Wahre Sendezeit des Satelliten (unbekannt)
$\Delta t_S^* = f(t_S^*)$	:	Fehler der Satellitenuhr zur wahren Sendezeit t_S^* (unbekannt)

Der Zusammenhang zwischen der falschen Sendezeit t_S und der wahren Sendezeit t_S^* ist $t_S = t_S^* + f(t_S^*)$. Somit gilt für die wahre Uhrzeit t_S^*:

$$t_S^* = t_S - \Delta t_S^* \tag{16}$$

Leider ist $\Delta t_S^* = f(t_S^*)$ nicht so einfach berechenbar, da t_S^* unbekannt ist. Δt_S^* kann jedoch durch eine Iteration angenähert bestimmt werden:

$$\begin{aligned} \Delta t_S^{(0)} &= f(t_S) \\ \Delta t_S^{(n)} &= f(t_S - \Delta t_S^{(n-1)}) \quad \text{für } n \in \mathbb{N} \setminus \{0\} \end{aligned} \tag{17}$$

Die Folge $(\Delta t_S^{(n)})_n$ konvergiert gegen den Fehler Δt_S^*. Die Konvergenz ist sehr schnell und es gilt bereits in sehr guter Genauigkeit $\Delta t_S^{(1)} \approx \Delta t_S^*$. Man kann sogar ganz auf die Iteration verzichten, denn der Unterschied zwischen Δt_S und Δt_S^* ist praktisch vernachlässigbar. Somit gilt für den falschen Satellitenuhrfehler Δt_S und die wahre Sendezeit t_S^* in ausreichender Genauigkeit:

$$\begin{aligned} \Delta t_S &= f(t_S) \\ t_S^* &\approx t_S - \Delta t_S \qquad \text{(ausreichende Genauigkeit)} \end{aligned} \tag{18}$$

Für den wahren Satellitenuhrfehler Δt_S^* und die wahre Sendezeit t_S^* gilt in sehr guter Genauigkeit:

$$\begin{aligned} \Delta t_S^* &\approx f(t_S - \Delta t_S) \qquad \text{(sehr gute Genauigkeit)} \\ t_S^* &= t_S - \Delta t_S^* \end{aligned} \tag{19}$$

3.3.2. Modellverbesserung 2: Fehler der Empfängeruhr

Auch der Fehler der Empfängeruhr muss korrigiert werden. In diesem Fall liegt die Ursache des Zeitfehlers in der Tatsache, dass im Empfänger eine einfache Quarzuhr mit einer Ungenauigkeit von einigen Sekunden pro Monat verbaut ist. Der Empfängeruhrfehler Δt_E sollte als vierte unbekannte Größe neben den Empfängerkoordinaten x_E, y_E und z_E im Gleichungssystem (2) auf Seite 14 berücksichtigt werden. Dafür wird noch eine zusätzliche Gleichung eines vierten Satelliten benötigt. Der Fehler der Empfängeruhr Δt_E verändert jede Pseudoentfernung um denselben Betrag $c\Delta t_E$, wobei c die Lichtgeschwindigkeit ist. Geht die Empfängeruhr nur um eine Millisekunde vor, so sind alle Pseudoentfernungen um etwa 300 Kilometer zu groß. Das korrigierte Gleichungssystem mit den Unbekannten x_E, y_E, z_E und Δt_E lautet somit:

$$\begin{aligned} \sqrt{(x_{S_1} - x_E)^2 + (y_{S_1} - y_E)^2 + (z_{S_1} - z_E)^2} &= d_{S_1} - c\Delta t_E \\ \sqrt{(x_{S_2} - x_E)^2 + (y_{S_2} - y_E)^2 + (z_{S_2} - z_E)^2} &= d_{S_2} - c\Delta t_E \\ \sqrt{(x_{S_3} - x_E)^2 + (y_{S_3} - y_E)^2 + (z_{S_3} - z_E)^2} &= d_{S_3} - c\Delta t_E \\ \sqrt{(x_{S_4} - x_E)^2 + (y_{S_4} - y_E)^2 + (z_{S_4} - z_E)^2} &= d_{S_4} - c\Delta t_E \end{aligned} \tag{20}$$

Jedoch führt die Multiplikation mit der sehr großen Konstanten c im Gleichungssystem zu einem deutlichen Genauigkeitsverlust bei einer numerischen Lösung mit dem Newton-Verfahren. Deshalb sollte das Gleichungssystem besser mit den vier Unbekannten x_E, y_E, z_E und Δd formuliert werden. Dabei ist $\Delta d = c\Delta t_E$ der Fehler der Pseudoentfernungen, welcher durch den Fehler der Empfängeruhr verursacht wird.

Das für das Newton-Verfahren als Nullstellenproblem dargestellte Gleichungssystem mit den Unbekannten x_E, y_E, z_E und Δd lautet:

$$\begin{aligned} \sqrt{(x_{S_1} - x_E)^2 + (y_{S_1} - y_E)^2 + (z_{S_1} - z_E)^2} - (d_{S_1} - \Delta d) &= 0 \\ \sqrt{(x_{S_2} - x_E)^2 + (y_{S_2} - y_E)^2 + (z_{S_2} - z_E)^2} - (d_{S_2} - \Delta d) &= 0 \\ \sqrt{(x_{S_3} - x_E)^2 + (y_{S_3} - y_E)^2 + (z_{S_3} - z_E)^2} - (d_{S_3} - \Delta d) &= 0 \\ \sqrt{(x_{S_4} - x_E)^2 + (y_{S_4} - y_E)^2 + (z_{S_4} - z_E)^2} - (d_{S_4} - \Delta d) &= 0 \end{aligned} \tag{21}$$

Das Newton-Verfahren ist somit auf die Funktionen f_1, f_2, f_3 und f_4 anzuwenden mit:

$$\begin{aligned} f_1(x,y,z,\delta) &= \sqrt{(x_{S_1}-x)^2+(y_{S_1}-y)^2+(z_{S_1}-z)^2}-(d_{S_1}-\delta) \\ f_2(x,y,z,\delta) &= \sqrt{(x_{S_2}-x)^2+(y_{S_2}-y)^2+(z_{S_2}-z)^2}-(d_{S_2}-\delta) \\ f_3(x,y,z,\delta) &= \sqrt{(x_{S_3}-x)^2+(y_{S_3}-y)^2+(z_{S_3}-z)^2}-(d_{S_3}-\delta) \\ f_4(x,y,z,\delta) &= \sqrt{(x_{S_4}-x)^2+(y_{S_4}-y)^2+(z_{S_4}-z)^2}-(d_{S_4}-\delta) \end{aligned} \tag{22}$$

Als Startpunkt für das Newton-Verfahren wählt man den Punkt (0,0,0,0). Er entspricht dem Erdmittelpunkt und einem Fehler der Pseudoentfernungen von 0 Metern.

3.3.3. Modellverbesserung 3: Form der Erde

Die Erde hat annähernd die Form eines abgeplatteten Rotationsellipsoiden. Die Abweichung von der Kugelform entsteht durch die Fliehkraft aufgrund der Erdrotation. Deshalb ist die Entfernung vom Mittelpunkt der Erde zum Äquator größer als zu den Polen. Der Abstand vom Mittelpunkt zum Äquator ist die große Halbachse a, der Abstand vom Mittelpunkt zu den beiden Polen ist die kleine Halbachse b. Die Abplattung f und die numerische Exzentrizität ε lassen sich aus den beiden Halbachsen a und b berechnen und sind wie folgt definiert:

$$\begin{aligned} f &= (a-b)/a \\ \varepsilon &= \sqrt{a^2-b^2}/a \end{aligned} \tag{23}$$

Der Zusammenhang zwischen den kartesischen Koordinaten x, y, z und den geographischen Koordinaten h, ϕ, λ bei einem Rotationsellipsoiden lautet:

$$\begin{aligned} x &= (N_\varphi + h)\cdot\cos(\varphi)\cdot\cos(\lambda) \\ y &= (N_\varphi + h)\cdot\cos(\varphi)\cdot\sin(\lambda) \\ z &= (N_\varphi(1-\varepsilon^2)+h)\cdot\sin(\varphi) \end{aligned} \tag{24}$$

Dabei gilt für den Krümmungsradius des Ersten Vertikals N_φ, dem Abstand des Lotfußpunktes vom Schnittpunkt des verlängerten Lots mit der z-Achse, folgender Zusammenhang:

$$N_\varphi = a/\sqrt{1-\varepsilon^2\sin^2(\varphi)} \tag{25}$$

Beim Referenzellipsoiden WGS 84 des World Geodetic Systems aus dem Jahre 1984 ist die große Halbachse a und die Abplattung f folgendermaßen definiert:

$$\begin{aligned} a &= 6378137{,}0 \text{ Meter} \\ f &= 1/298{,}257223563 \end{aligned} \tag{26}$$

Daraus ergeben sich wegen $b = (1-f)\,a$ und $\varepsilon = \sqrt{1-(1-f)^2}$ annähernd folgende Werte für die kleine Halbachse b und für die numerische Exzentrizität ε:

$$\begin{aligned} b &\approx 6356752{,}3 \text{ Meter} \\ \varepsilon &\approx 8{,}181919\times10^{-2} \end{aligned} \tag{27}$$

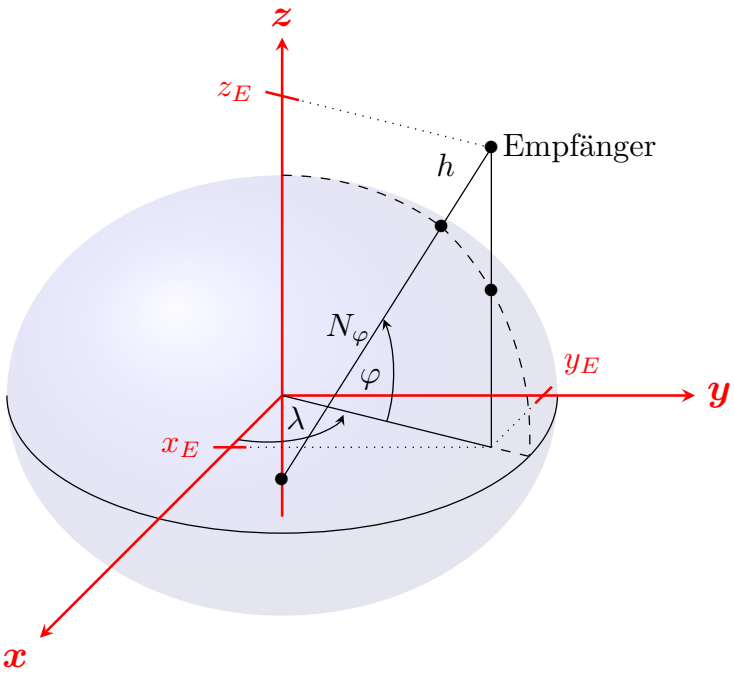


Abbildung 5: Zusammenhang zwischen kartesischen und geographischen Koordinaten

Die Unbekannten h, φ und λ können aus den Empfängerkoordinaten x_E, y_E und z_E numerisch bestimmt werden. Dazu müssen die Gleichungen (24) und (25) als Nullstellenproblem dargestellt werden und mit dem Newton-Verfahren gelöst werden:

$$\begin{aligned}
(a/\sqrt{1-\varepsilon^2\sin^2(\varphi)}+h)\cdot\cos(\varphi)\cdot\cos(\lambda)-x_E&=0\\
(a/\sqrt{1-\varepsilon^2\sin^2(\varphi)}+h)\cdot\cos(\varphi)\cdot\sin(\lambda)-y_E&=0\\
(a(1-\varepsilon^2)/\sqrt{1-\varepsilon^2\sin^2(\varphi)}+h)\cdot\sin(\varphi)-z_E&=0
\end{aligned} \tag{28}$$

Das Newton-Verfahren ist somit auf die Funktionen f_1, f_2 und f_3 anzuwenden mit:

$$\begin{aligned}
f_1(h,\varphi,\lambda)&=(a/\sqrt{1-\varepsilon^2\sin^2(\varphi)}+h)\cdot\cos(\varphi)\cdot\cos(\lambda)-x_E\\
f_2(h,\varphi,\lambda)&=(a/\sqrt{1-\varepsilon^2\sin^2(\varphi)}+h)\cdot\cos(\varphi)\cdot\sin(\lambda)-y_E\\
f_3(h,\varphi,\lambda)&=(a(1-\varepsilon^2)/\sqrt{1-\varepsilon^2\sin^2(\varphi)}+h)\cdot\sin(\varphi)-z_E
\end{aligned} \tag{29}$$

Der Startpunkt $(h_0,\varphi_0,\lambda_0)$ für das Newton-Verfahren darf hier nicht einfach gleich (0,0,0) gewählt werden. Er muss möglichst in der Nähe der Lösung sein und sollte deshalb aus der Gleichung (11) auf Seite 16 bestimmt werden:

$$\begin{aligned}
h_0&=\sqrt{x_E^2+y_E^2+z_E^2}-a\\
\phi_0&=\operatorname{atan2}\left(z_E,\sqrt{x_E^2+y_E^2}\right)\\
\lambda_0&=\operatorname{atan2}\left(y_E,x_E\right)
\end{aligned}$$

3.3.4. Modellverbesserung 4: Verwendung aller Satelliten

Bisher wurden die Daten von drei oder vier Satelliten genutzt. Oftmals hat man jedoch die Daten von acht oder mehr Satelliten vorliegen. Würde man alle empfangenen Daten verwenden, hätte man mehr Gleichungen als Unbekannte und das überbestimmte Gleichungssystem hätte keine Lösung mehr, da die Satellitendaten durch kleine Fehler gestört sind. Verwendet man hingegen nur die Daten von vier Satelliten, so erhält man zwar eine brauchbare Lösung, diese ist jedoch von den ausgewählten Satelliten abhängig und somit nicht eindeutig. Außerdem stellt sich die Frage, welche Satelliten man am besten für die Positionsbestimmung auswählen sollte. Mit den Methoden der Ausgleichsrechnung ist es jedoch möglich, alle Satellitendaten zu verwenden und das überbestimmte Gleichungssystem zwar nicht exakt, aber immerhin näherungsweise zu lösen, so dass die gefundene Näherungslösung in einem gewissen Sinn optimal und eindeutig ist. Bei n empfangenen Satelliten werden folgende Variablen benötigt und müssen aus dem Datensatz kopiert werden:

- Der Vektor $\vec{t}_E$ der Empfangszeiten mit den n Komponenten t_{E_1} bis t_{E_n}
- Der Vektor $\vec{t}_S$ der Sendezeiten mit den n Komponenten t_{S_1} bis t_{S_n}
- Die Ephemeridenmatrix E_{ph} mit den $21 \times n$ Komponenten $E_{ph_{1,1}}$ bis $E_{ph_{21,n}}$

Angenommen, ein Gleichungssystem mit m Unbekannten ξ_1 bis ξ_m besteht aus den n Gleichungen:

$$\begin{aligned} f_1(\xi_1, \ldots, \xi_m) &= 0 \\ f_2(\xi_1, \ldots, \xi_m) &= 0 \\ &\vdots \\ f_n(\xi_1, \ldots, \xi_m) &= 0 \end{aligned} \tag{30}$$

Lässt sich das Gleichungssystem exakt lösen, so erfüllt jede Lösung die Gleichung:

$$f_1^2(\xi_1, \ldots, \xi_m) + f_2^2(\xi_1, \ldots, \xi_m) + \ \ldots \ + f_n^2(\xi_1, \ldots, \xi_m) = 0$$

Wenn das Gleichungssystem keine Lösung hat gilt jedoch immer:

$$f_1^2(\xi_1, \ldots, \xi_m) + f_2^2(\xi_1, \ldots, \xi_m) + \ \ldots \ + f_n^2(\xi_1, \ldots, \xi_m) > 0$$

In diesem Fall kann man keine Lösung finden, für die die Summe der Quadrate gleich Null ist. Stattdessen kann man aber nach einer Näherungslösung suchen, für die diese Summe so klein wie möglich ist:

$$f_1^2(\xi_1, \ldots, \xi_m) + f_2^2(\xi_1, \ldots, \xi_m) + \ \ldots \ + f_n^2(\xi_1, \ldots, \xi_m) \to \min$$

Das Auffinden einer solchen Näherungslösung wird die *Methode der kleinsten Quadrate* genannt und ist das Standardverfahren der Ausgleichsrechnung. Bekannte Verfahren zur Lösung nichtlinearer Ausgleichsprobleme sind das *Gauß-Newton-Verfahren* und der *Levenberg-Marquardt-Algorithmus*. Beides sind Abkömmlinge des Newton-Verfahrens und benötigen ebenfalls einen Startpunkt für die Iteration. Für n empfangene

Satelliten sehen die Funktionen f_1 bis f_n wie folgt aus:

$$\begin{aligned} f_1(x,y,z) &= \sqrt{(x_{S_1}-x)^2+(y_{S_1}-y)^2+(z_{S_1}-z)^2} - d_{S_1} \\ f_2(x,y,z) &= \sqrt{(x_{S_2}-x)^2+(y_{S_2}-y)^2+(z_{S_2}-z)^2} - d_{S_2} \\ &\vdots \\ f_n(x,y,z) &= \sqrt{(x_{S_n}-x)^2+(y_{S_n}-y)^2+(z_{S_n}-z)^2} - d_{S_n} \end{aligned} \tag{31}$$

Auch hier sollte als Startpunkt für das Iterationsverfahren wieder der Erdmittelpunkt (0,0,0) gewählt werden. In Kombination mit der zweiten Modellverbesserung sehen die Funktionen f_1 bis f_n wie folgt aus:

$$\begin{aligned} f_1(x,y,z,\delta) &= \sqrt{(x_{S_1}-x)^2+(y_{S_1}-y)^2+(z_{S_1}-z)^2} - (d_{S_1}-\delta) \\ f_2(x,y,z,\delta) &= \sqrt{(x_{S_2}-x)^2+(y_{S_2}-y)^2+(z_{S_2}-z)^2} - (d_{S_2}-\delta) \\ &\vdots \\ f_n(x,y,z,\delta) &= \sqrt{(x_{S_n}-x)^2+(y_{S_n}-y)^2+(z_{S_n}-z)^2} - (d_{S_n}-\delta) \end{aligned} \tag{32}$$

Als Startpunkt für das Iterationsverfahren ist wieder der Nullvektor (0,0,0,0) zu wählen.

3.3.5. Modellverbesserung 5: Gewichtung der Gleichungen

Eine weitere Modellannahme ist bisher immer gewesen, dass die von den Satelliten ausgesendeten Nachrichten mit der Lichtgeschwindigkeit $c = 299792458$ Meter pro Sekunde zur Erde gelangt sind. Dabei wird jedoch übersehen, dass die Erdatmosphäre die Nachrichten abbremst, und zwar umso mehr, je länger der Weg der Nachrichten durch die Atmosphäre ist. Nachrichten von Satelliten, die im Zenit über dem Empfänger stehen, werden somit etwas weniger abgebremst als Nachrichten, die schräg einfallen. Da die Signalstärken vom Empfänger gemessen und protokolliert werden, können sie als Gewichte für die Ausgleichsrechnung verwendet werden. Es werden für die Berechnungen folgende Variablen benötigt und aus dem Datensatz kopiert:

- Der Vektor $\vec{t}_E$ der Empfangszeiten mit den n Komponenten t_{E_1} bis t_{E_n}
- Der Vektor $\vec{t}_S$ der Sendezeiten mit den n Komponenten t_{S_1} bis t_{S_n}
- Der Vektor $\vec{g}$ der Signalstärken mit den n Komponenten g_1 bis g_n
- Die Ephemeridenmatrix E_{ph} mit den $21 \times n$ Komponenten $E_{ph_{1,1}}$ bis $E_{ph_{21,n}}$

Als einfaches Modell für eine Atmosphärenkorrektur kann man den Nachrichten mit großer Signalstärke eine höhere Gewichtung geben als den Nachrichten mit kleiner Signalstärke. Besteht bei n empfangenen Nachrichten das Gleichungssystem aus den n Gleichungen in (30), so wählt man die Gewichte der Gleichungen g_1, g_2, …, g_n proportional zu den Signalstärken und minimiert die gewichtete Summe:

$$g_1 \cdot f_1^2(\xi_1,\ldots,\xi_m) + g_2 \cdot f_2^2(\xi_1,\ldots,\xi_m) + \ \ldots \ + g_n \cdot f_n^2(\xi_1,\ldots,\xi_m) \to \min$$

Der Proportionalitätsfaktor hat dabei keinen Einfluss auf die Lösung. Denn seien g_1, g_2, …, g_n und g_1^*, g_2^*, …, g_n^* jeweils zu den Signalstärken proportionale Gewichte.

Dann gibt es eine Zahl $\gamma > 0$ mit $g_k^* = \gamma g_k$ für $k \in \{1,2,\ldots,n\}$, und es gilt:

$$\begin{aligned}
&g_1^* \cdot f_1^2(\xi_1,\ldots,\xi_m) + g_2^* \cdot f_2^2(\xi_1,\ldots,\xi_m) + \ \ldots \ + g_n^* \cdot f_n^2(\xi_1,\ldots,\xi_m) \\
&= \gamma g_1 \cdot f_1^2(\xi_1,\ldots,\xi_m) + \gamma g_2 \cdot f_2^2(\xi_1,\ldots,\xi_m) + \ \ldots \ + \gamma g_n \cdot f_n^2(\xi_1,\ldots,\xi_m) \\
&= \gamma \Big(g_1 \cdot f_1^2(\xi_1,\ldots,\xi_m) + g_2 \cdot f_2^2(\xi_1,\ldots,\xi_m) + \ \ldots \ + g_n \cdot f_n^2(\xi_1,\ldots,\xi_m) \Big)
\end{aligned}$$

Wegen $\gamma > 0$ ist die mit g_1^*, g_2^*, …, g_n^* gewichtete Summe genau dann minimal, wenn die mit g_1, g_2, …, g_n gewichtete Summe minimal ist. Um das Minimum der gewichteten Summe zu finden, definiert man neue Funktionen h_1 bis h_n durch:

$$h_k(\xi_1,\ldots,\xi_m) := \sqrt{g_k} \cdot f_k(\xi_1,\ldots,\xi_m) \quad \text{für} \quad k \in \{1,\ldots,n\}$$

Mit dem Gauß-Newton-Verfahren oder dem Levenberg-Marquardt-Algorithmus berechnet man eine Kleinste-Quadrate-Lösung für die Funktionen h_1 bis h_n:

$$h_1^2(\xi_1,\ldots,\xi_m) + h_2^2(\xi_1,\ldots,\xi_m) + \ \ldots \ + h_n^2(\xi_1,\ldots,\xi_m) \to \min$$

Diese Lösung ist gleichzeitig eine Kleinste-Quadrate-Lösung der gewichteten Summe, denn für alle $\xi_1,\ldots,\xi_m$ gilt:

$$\begin{aligned}
&h_1^2(\xi_1,\ldots,\xi_m) + h_2^2(\xi_1,\ldots,\xi_m) + \ \ldots \ + h_n^2(\xi_1,\ldots,\xi_m) \\
&= g_1 \cdot f_1^2(\xi_1,\ldots,\xi_m) + g_2 \cdot f_2^2(\xi_1,\ldots,\xi_m) + \ \ldots \ + g_n \cdot f_n^2(\xi_1,\ldots,\xi_m)
\end{aligned}$$

Für n empfangene Satelliten ergeben sich die durch die Signalstärken g_1 bis g_n gewichteten Funktionen h_1 bis h_n aus den Funktionen f_1 bis f_n aus Gleichung (31) wie dargestellt:

$$\begin{aligned}
h_1(x,y,z) &= \sqrt{g_1} \cdot f_1(x,y,z) \\
h_2(x,y,z) &= \sqrt{g_2} \cdot f_2(x,y,z) \\
&\vdots \\
h_n(x,y,z) &= \sqrt{g_n} \cdot f_n(x,y,z)
\end{aligned} \tag{33}$$

In Kombination mit der zweiten Modellverbesserung ergeben sich dabei die Funktionen h_1 bis h_n aus den Funktionen f_1 bis f_n aus Gleichung (32) wie folgt:

$$\begin{aligned}
h_1(x,y,z,\delta) &= \sqrt{g_1} \cdot f_1(x,y,z,\delta) \\
h_2(x,y,z,\delta) &= \sqrt{g_2} \cdot f_2(x,y,z,\delta) \\
&\vdots \\
h_n(x,y,z,\delta) &= \sqrt{g_n} \cdot f_n(x,y,z,\delta)
\end{aligned} \tag{34}$$

Als Startpunkt für das Iterationsverfahren ist hierbei wieder der Nullvektor (0,0,0) bzw. (0,0,0,0) zu wählen.

4. Didaktisch-methodische Realisierung des Modellierungstages

In diesem Kapitel soll das entwickelte Lernmodul vorgestellt werden. Zuvor wird kurz der Ablauf eines typischen Modellierungstages beschrieben und Besonderheiten des Mathematiksystems MATLAB erklärt, da der Modellierungsschritt „mathematisch arbeiten“ in diesem Lernmodul größtenteils von MATLAB übernommen wird. Die Verwendung geeigneter Werkzeuge zur Lösung der mathematischen Problemstellung steht dabei in Einklang mit den Kernlehrplänen, in denen sie sogar als eigener Kompetenzbereich verankert ist. Dadurch kann hier der Fokus vom Lösen der mathematischen Probleme auf den eigentlichen Modellbildungsprozess verschoben werden und die im Rahmen eines Modellierungstages begrenzte Zeit für die notwendigen Modellverbesserungen genutzt werden. Bei der Besprechung des entwickelten MATLAB-Programms wird ausführlich auf didaktische und methodische Entscheidungen bei der Programmentwicklung eingegangen. Insbesondere wird die Überprüfungsfunktion besprochen, die ein wichtiger Baustein des Lernmoduls ist.

4.1. Ablauf eines Modellierungstages

Nach der Begrüßung beginnt der Tag mit einem Vortrag über das mathematische Modellieren, in dem der Modellierungskreislauf vorgestellt und erläutert wird. Danach wird das inhaltliche Thema des Tages bekannt gegeben: die Positionsbestimmung mit Hilfe von GPS. Dies geschieht in Form einer kurzen Präsentation und einem Filmausschnitt, in dem die Positionsbestimmung mit drei Satelliten an einem Modell der Erde veranschaulicht wird. Im Anschluss an diesen Film werden die Schülerinnen und Schüler aufgefordert, sich in Zweiergruppen vor einen Computer zu setzen. Dann erhalten sie eine interaktive Einführung in das Mathematiksystem MATLAB. Dabei wird insbesondere auf den Umgang mit Matrizen und Vektoren eingegangen, welche von zentraler Bedeutung für das Verständnis des später zu bearbeitenden MATLAB-Programms sind. Danach werden die Arbeitsblätter ausgeteilt und die Schülerinnen und Schüler beginnen mit der Bearbeitung der Aufgaben. In der ersten Phase der Modellschritte wird ein Modell zur Berechnung der Empfängerposition schrittweise von den Schülerinnen und Schülern vervollständigt. Dazu müssen die nötigen Modellschritte der Reihe nach entlang des Arbeitsblattes und der vom Programm ausgegebenen Fehlermeldungen bearbeitet werden. Wenn alle Aufgaben gelöst und das Programm vervollständigt worden ist, beginnt die Phase der Modellverbesserungen. Diese brauchen nicht mehr der Reihe nach bearbeitet zu werden. Stattdessen können die Schülerinnen und Schüler in Expertengruppen für die Modellverbesserungen aufgeteilt werden. Die einzelnen Lösungen können später unter den verschiedenen Expertengruppen ausgetauscht und miteinander kombiniert werden. Natürlich können leistungsstarke Gruppen auch mehrere Modellverbesserungen selbstständig bearbeiten. Der Tag endet mit einer kurzen Präsentation der Lösungen der verschiedenen Gruppen.

4.2. MATLAB und die Optimization Toolbox

Dieses Lernmodul wurde mit dem interaktiven Mathematiksystem MATLAB® unter Verwendung der Optimization Toolbox™ erstellt. MATLAB ist eine kommerzielle Software des Unternehmens MathWorks® und wurde ursprünglich für numerische Berechnungen mit Hilfe von Matrizen entwickelt. Durch eine eingebaute Programmierumgebung können auch aufwendigere Berechnungen durchgeführt werden. Dabei ist

der Umgang mit MATLAB leicht zu erlernen, so dass es auch von Schülerinnen und Schülern ohne Vorkenntnisse benutzt werden kann. Die Funktionalität von MATLAB kann durch zahlreiche Funktionssammlungen, sogenannte Toolboxen, für viele Anwendungsgebiete erweitert werden. Für dieses Lernmodul wird die Optimization Toolbox benötigt, welche die beiden Programme `fsolve` und `lsqnonlin` zum Lösen nichtlinearer Gleichungssysteme enthält.

Als Zahlen werden in MATLAB standardmäßig Gleitkommazahlen mit doppelter Genauigkeit verwendet, wobei die Norm IEEE 754 für binäre Gleitkommazahlen berücksichtigt wird. Durch sie werden die zusätzlichen Werte `Inf` für *Infinity* und `NaN` für *Not a Number* vorgeschrieben. Dadurch werden z. B. Divisionen durch Null möglich, und es gelten die Gleichungen $1/0 =$ `Inf` und $0/0 =$ `NaN`. Der Wert `NaN` wird als Platzhalter in den Aufgaben verwendet.

Neben dem Rechnen mit Zahlen beherrscht MATLAB auch den Umgang mit Matrizen und Vektoren. Tatsächlich werden Zahlen und Vektoren nur als Spezialfälle von Matrizen behandelt. In diesem Lernmodul werden Vektoren und Matrizen verwendet, um den Programmcode einfach und überschaubar zu halten. Um verschiedene Matrizen und Vektoren übersichtlich zusammenfassen zu können, werden in MATLAB sogenannte strukturierte Arrays verwendet. Die Satellitendaten werden in einem strukturierten Array abgespeichert.

Schließlich lassen sich in MATLAB neben einfachen reellen Funktionen auch vektorwertige Funktionen mehrerer Veränderlicher erstellen. Diese lassen sich unter anderem zur Darstellung von Gleichungssystemen nutzen und finden dafür auch in diesem Lernmodul Verwendung. Umfangreichere Funktionsdeklarationen können in eigene Funktionsdateien ausgelagert werden. So werden die Funktionen zum Einlesen der Satellitendaten und zur Berechnung der Satellitenkoordinaten und Satellitenuhrfehler in Funktionsdateien ausgelagert, um das Hauptprogramm übersichtlich zu lassen. Um Programmcode zu verbergen kann dieser in eine geschützte Funktionsdatei ausgelagert werden, wo er ausgeführt, jedoch nicht eingesehen werden kann. Eine geschützte Funktionsdatei wird für eine Funktion zum Überprüfen der Lösungen verwendet.

4.3. Der Aufbau des Lernmoduls

Das Lernmodul umfasst ein MATLAB-Programm, ein Arbeitsblatt für die erste Phase der Modellschritte, ein weiteres Arbeitsblatt für die zweite Phase der Modellverbesserungen, Hilfekarten zu den einzelnen Aufgaben und Musterlösungen für die Betreuer. Die Arbeitsblätter, Hilfekarten und Musterlösungen können im Anhang eingesehen werden. Das MATLAB-Programm besteht aus folgenden Dateien:

- `gps.m`: Das Hauptprogramm, welches von den Schülerinnen und Schülern bearbeitet werden soll
- `114255.obs` und `114255.nav`: Die RINEX-Dateien einer GPS-Messung
- `einlesen_Satellitendaten.m`: Eine Funktionsdatei für eine Funktion zum Einlesen der Satellitendaten aus den beiden RINEX-Dateien
- `berechne_Satellitenkoordinaten.m`: Eine Funktionsdatei für eine Funktion zum Berechnen der Satellitenkoordinaten
- `berechne_Satellitenuhrfehler.m`: Eine Funktionsdatei für eine Funktion zum Berechnen der Satellitenuhrfehler

- `ueberpruefe_Loesung.p`: Eine geschützte Funktionsdatei für eine Funktion zum Überprüfen der Lösungen

Das vollständige Hauptprogramm `gps.m` und die Funktionsdateien sind ebenfalls im Anhang aufgeführt. Die beiden RINEX-Dateien sind aus Platzgründen nicht abgedruckt, können aber auf der beiliegenden CD eingesehen werden. Für das Einlesen der RINEX-Dateien wird eine modifizierte Version der MATLAB-Funktion `readrnx.m` aus dem quelloffenen Softwareprojekt RTKLIB[2] verwendet (vgl. [10], T. Takasu, `http://gpspp.sakura.ne.jp/prog/rtkdemo/readrnx.m`).

4.3.1. Die Überprüfungsfunktion

Die Funktion `ueberpruefe_Loesung` aus der geschützten Funktionsdatei `ueberpruefe_Loesung.p` erfüllt in diesem Lernmodul eine zentrale Schlüsselrolle. Sie steuert durch den Modellbildungsprozess und unterbricht die Programmausführung immer in genau dem Abschnitt, in dem sich die gerade zu bearbeitende Aufgabe befindet. Dadurch wird man Schritt für Schritt durch die einzelnen Stationen des Programms geführt. Außerdem überprüft sie die eingegebenen Lösungen auf Korrektheit und erkennt viele typische und häufig gemachte Fehler. Bei einem falschen Winkel prüft sie z. B. ob nur ein falsches Winkelmaß verwendet wurde und gibt eine entsprechende Fehlermeldung aus. Indem sie leicht fehlerhafte Lösungen nicht als „falsch“, sondern als „fast-richtig“ erkennt, fördert sie die Motivation und Leistungsbereitschaft der Schülerinnen und Schüler. Da sie in eine geschützte Funktionsdatei ausgelagert wurde, kann sie von den Schülerinnen und Schüler nicht eingesehen werden. So wird verhindert, dass die Lösungen der Aufgaben aus der Überprüfungsfunktion erschlossen werden können. Die ungeschützte Funktionsdatei `ueberpruefe_Loesung.m` ist im Anhang abgedruckt.

4.4. Die Modellschritte

Die Schülerinnen und Schüler arbeiten nur mit dem Hauptprogramm `gps.m`. Dieses ist unterteilt in die folgenden sechs Abschnitte:

- Einlesen der Satellitendaten
- Schritt 1 | Berechnen der Satellitenkoordinaten
- Schritt 2 | Berechnen der Pseudoentfernungen
- Schritt 3 | Berechnen der Empfängerkoordinaten
- Schritt 4 | Umrechnen in geographische Höhe, Breite und Länge
- Anzeigen der Empfängerposition

Das Hauptprogramm könnte von MATLAB abschnittsweise ausgeführt werden, jedoch ist dies durch die Überprüfungsfunktion gar nicht notwendig, denn sie unterbricht den Programmablauf automatisch im richtigen Abschnitt und gibt zudem die zu bearbeitende Aufgabe auf dem Bildschirm aus.

[2] `http://www.rtklib.com/`

4.4.1. Einlesen der Satellitendaten

Zuerst werden die beiden RINEX-Dateien in das strukturierte Array `Satellitendaten` eingelesen. Danach wird ein Datensatz ausgewählt und die Nummern aller Satelliten, deren Nachrichten empfangen wurden, in die Variable `Satelliten` übertragen. Durch die Variable `Satellitenauswahl` werden drei dieser Satelliten ausgewählt und die zugehörigen Empfangszeiten, Sendezeiten und Ephemeriden in die Variablen `tE`, `tS` und `Ephemeriden` übertragen. Schließlich werden alle wichtigen Variablen auf dem Bildschirm ausgegeben. Der MATLAB-Code für diesen Abschnitt lautet folgendermaßen:

```
%% Einlesen der Satellitendaten
display('Einlesen der Satellitendaten')

%   Einlesen der Satellitendaten falls nicht vorhanden
if ~exist('Satellitendaten','var')
    Dateien = {'114255.obs','114255.nav'};
    Satellitendaten = einlesen_Satellitendaten(Dateien);
end

%   Auswählen eines Datensatzes
Datensatznummer = 1;
Datensatz = Satellitendaten(Datensatznummer);

%   Satelliten, deren Nachrichten empfangen wurden
Satelliten = Datensatz.Satelliten;

%   Auswählen der Satelliten
Satellitenauswahl = [1 2 3];

%   Auslesen der ausgewählten Satellitendaten
tE = Datensatz.Empfangszeiten(Satellitenauswahl);
tS = Datensatz.Sendezeiten(Satellitenauswahl);
Ephemeriden = Datensatz.Ephemeriden(:,Satellitenauswahl);

%   Anzeigen der Ergebnisse
format shortG
display(Datensatz,sprintf('Datensatz Nummer %d',Datensatznummer))
display(Satelliten, ...
    'Satelliten, deren Nachrichten empfangen wurden [Satelliten]')
display(Satellitenauswahl, ...
    'Nummern der ausgewählten Satelliten [Satellitenauswahl]')
format bank
display(tE, ...
    'Empfangszeiten der ausgewählten Nachrichten in Sekunden [tE]')
display(tS,'Sendezeiten der ausgewählten Nachrichten in Sekunden [tS]')
display(['Die Positionsdaten der Satelliten werden nicht angezeigt ' ...
    '[Ephemeriden]'])
clearvars Dateien Datensatznummer
```

Die Bildschirmausgabe nach dem Ausführen dieses Programmabschnitts ist wie folgt:

```
Einlesen der Satellitendaten

Lese Rinex Datei ... : 114255.obs
Lese Rinex Datei ... : 114255.nav

Datensatz Nummer 1 =

        Satelliten: [1 2 3 4 5 6 7 8]
     Wochennummern: [1749 1749 1749 1749 1749 1749 1749 1749]
    Empfangszeiten: [389495 389495 389495 389495 389495 389495 ...]
```

```
      Sendezeiten: [3.8949e+05 3.8949e+05 3.8949e+05 3.8949e+05 ...]
   Signalstaerken: [31 32 29 32 28 47 26 44]
      Ephemeriden: [21x8 double]

Satelliten, deren Nachrichten empfangen wurden [Satelliten] =

     1     2     3     4     5     6     7     8

Nummern der ausgewählten Satelliten [Satellitenauswahl] =

     1     2     3

Empfangszeiten der Nachrichten in Sekunden seit Wochenbeginn [tE] =

     389495.00     389495.00     389495.00

Sendezeiten der Nachrichten in Sekunden seit Wochenbeginn [tS] =

     389494.92     389494.92     389494.93

Die Positionsdaten der Satelliten werden nicht angezeigt [Ephemeriden]
```

Didaktisch-methodische Anmerkungen In diesem Abschnitt werden die Schülerinnen und Schüler schon mit dem Gebrauch von Vektoren in MATLAB vertraut gemacht. Alle für die Bearbeitung der Aufgaben relevanten Informationen sind in Vektoren abgespeichert und werden auch auf dem Bildschirm ausgegeben. Durch Vergleich der Bildschirmausgabe mit dem Programm können die Schülerinnen und Schüler selber erkennen, wie man in MATLAB mit Vektoren umgehen muss. Dafür brauchen sie nicht einmal Vorwissen aus der Schule, denn sie können sich Vektoren ohne geometrischen Hintergrund einfach als Listen von Zahlen vorstellen. Die Ephemeridenmatrix wird nur intern für die Berechnung der Satellitenkoordinaten benötigt. Sie würde unverhältnismäßig viel Platz auf dem Bildschirm beanspruchen, ohne dass ihr Inhalt von Nutzen wäre. Deshalb wird sie nicht angezeigt, jedoch wird durch eine entsprechende Bildschirmausgabe angedeutet, dass sich hinter den Ephemeriden die Positionsdaten der Satelliten verbergen. Generell wird nicht einfach nur der Variablenname, sondern immer ein kurzer, erklärender Text zusammen mit dem Variablennamen ausgegeben. Dadurch tragen die Bildschirmausgaben alleine schon zum Verständnis der Zusammenhänge bei. Daneben erleichtern auch aussagekräftige, eingedeutschte Namen für die Variablen und Funktionen sowie kurze Kommentare im Programm das Verständnis für die einzelnen Berechnungsschritte. Nur die für Rechnungen wichtigen Variablen werden mit kurzen und aus dem Unterricht bekannten Symbolen bezeichnet. So heißen die Vektoren der Empfangs- und Sendezeiten nicht `Empfangszeiten` und `Sendezeiten`, sondern `tE` und `tS`. Dadurch werden die Rechnungen in den nachfolgenden Schritten natürlich und vertraut aussehen.

4.4.2. Berechnen der Satellitenkoordinaten

Im zweiten Abschnitt werden die Koordinaten der ausgewählten Satelliten berechnet. Dazu wird die Funktion `[x,y,z] = berechne_Satellitenkoordinaten(t,Ephemeriden)` aus der Funktionsdatei `berechne_Satellitenkoordinaten.m` aufgerufen. Sie berechnet aus einen Zeitvektor `t` und der Ephemeridenmatrix die Satellitenkoordinaten zu den gewünschten Zeitpunkten und speichert diese in drei Vektoren `x`, `y` und `z`. Die Berechnung der Satellitenkoordinaten geschieht dabei wie in Tabelle 2 auf Seite 12 dargestellt.

Die zugehörige MATLAB-Funktionsdatei ist im Anhang auf Seite 70 aufgeführt. Im Hauptprogramm werden die Satellitenkoordinaten zur besseren Unterscheidung von den Empfängerkoordinaten mit `xS`, `yS` und `zS` bezeichnet. Die Schüleraufgabe liegt in der Überlegung, zu welchen Zeitpunkten die Satellitenkoordinaten benötigt werden und wie diese Zeitpunkte korrekt übergeben werden müssen. Dazu ist folgender Programmabschnitt zu bearbeiten:

```
%% Schritt 1 | Berechnen der Satellitenkoordinaten
display('Schritt 1 | Berechnen der Satellitenkoordinaten')

%  Berechnen der Satellitenkoordinaten
[xS,yS,zS] = berechne_Satellitenkoordinaten(NaN,Ephemeriden);

%  Überprüfen der Satellitenkoordinaten
ueberpruefe_Loesung(Phase,'Schritt 1', ...
    xS,yS,zS,Datensatz,Satellitenauswahl);

%  Anzeigen der Ergebnisse
format bank
display([xS(1) yS(1) zS(1)], ...
    'Koordinaten des ersten Satelliten in Meter [xS(1) yS(1) zS(1)]')
display([xS(2) yS(2) zS(2)], ...
    'Koordinaten des zweiten Satelliten in Meter [xS(2) yS(2) zS(2)]')
display([xS(3) yS(3) zS(3)], ...
    'Koordinaten des dritten Satelliten in Meter [xS(3) yS(3) zS(3)]')
```

Die Bildschirmausgabe nach dem Ausführen dieses Programmabschnitts ist wie folgt:

```
Schritt 1 | Berechnen der Satellitenkoordinaten

Bitte bearbeite Schritt 1
Berechne die Satellitenkoordinaten zu den richtigen Zeitpunkten.

Error using ueberpruefe_Loesung (line 43)

Bitte bearbeite Schritt 1
Ersetze das "NaN" in der Zeile
    [xS,yS,zS] = berechne_Satellitenkoordinaten(NaN,Ephemeriden);
durch die richtigen Zeiten.
```

An dieser Stelle wird durch die Überprüfungsfunktion eine Fehlermeldung erzeugt, weil die erste Aufgabe noch nicht bearbeitet worden ist. Es sollen im ersten Schritt die Satellitenkoordinaten zu den „richtigen“ Zeitpunkten berechnet werden. Der Aufruf dafür ist schon vorbereitet, nur ist an der Stelle der Zeitpunkte der Platzhalter `NaN` an die Funktion `berechne_Satellitenkoordinaten` übergeben worden. Dieser Platzhalter soll durch die richtigen Zeiten ersetzt werden.

Die Schülerinnen und Schüler sollen zuerst herausfinden, dass die Satellitenkoordinaten zu den Sendezeiten der Nachrichten benötigt werden. Danach müssen sie das Problem lösen, die Sendezeiten korrekt als Funktionsparameter zu übergeben. Das ist insofern eine kleine Hürde, da die Sendezeiten im Gegensatz zu den Empfangszeiten alle verschieden sind und deshalb nicht durch eine einzige Zahl dargestellt werden können. Stattdessen muss der Vektor `tS` verwendet werden.

Ein häufig gemachter Fehler liegt darin, die Sendezeiten der ausgewählten Nachrichten aus der Bildschirmausgabe zu kopieren und zu einem Vektor zusammenzufassen. Diese Zeiten sind jedoch nicht genau genug, da sie für die Ausgabe auf die hundertstel Sekunde gerundet wurden. Dieser Sachverhalt wird von der Überprüfungsfunktion erkannt und entsprechend kommentiert. Auch die Verwendung des falschen Zeitvek-

tors `tE` wird durch die Überprüfungsfunktion entsprechend kommentiert. Die korrekte Lösung lautet einfach:

```
%   Berechnen der Satellitenkoordinaten
[xS,yS,zS] = berechne_Satellitenkoordinaten(tS,Ephemeriden);
```

Bei der richtigen Lösung erhält man folgende Bildschirmausgabe:

```
Schritt 1  |  Berechnen der Satellitenkoordinaten

Schritt 1 scheint korrekt gelöst zu sein.

Koordinaten des ersten Satelliten in Meter [xS(1) yS(1) zS(1)] =

    8687804.07   20463717.04   15116871.21

Koordinaten des zweiten Satelliten in Meter [xS(2) yS(2) zS(2)] =

   -3482859.98   16258573.19   20371183.79

Koordinaten des dritten Satelliten in Meter [xS(3) yS(3) zS(3)] =

   11174291.61   13291625.12   20112588.81
```

Didaktisch-methodische Anmerkungen: Dieser erste Schritt stellt kein wirkliches Problem dar, sondern ist als leichter Einstieg in MATLAB und das Thema GPS gedacht. Da die genaue Vorschrift zur Berechnung dieser Koordinaten wegen ihrer Komplexität nicht Thema dieses Lernmoduls sein kann, sollen die Schülerinnen und Schüler nur die richtigen Zeitpunkte erkennen. Die Verwendung einer der beiden Vektoren `tE` oder `tS` ist dabei durch den vorigen Abschnitt schon nahegelegt worden. Aus technischer Sicht ist hier die Tatsache interessant, dass es in MATLAB keinen großen Unterschied macht mit Zahlen oder Vektoren zu rechnen. Genau wie die Sendezeiten `tS` sind auch die Satellitenkoordinaten `xS`, `yS` und `zS` Vektoren mit jeweils drei Komponenten.

Später soll aus den Komponenten dieser Vektoren ein Gleichungssystem aufgebaut werden. Damit die Teilnehmenden erkennen können, wie in MATLAB auf diese Komponenten zugegriffen werden kann, werden in der Bildschirmausgabe nicht einfach die Vektoren `xS`, `yS` und `zS` ausgegeben, obwohl das aus programmiertechnischer Sicht kürzer und eleganter wäre. Stattdessen werden die einzelnen Koordinaten aus den Vektoren ausgelesen und nach Satelliten geordnet zusammengefasst. So erkennt man sowohl im Programm als auch in der Bildschirmausgabe, dass der erste Satellit die Koordinaten `xS(1)`, `yS(1)` und `zS(1)`, der zweite Satellit die Koordinaten `xS(2)`, `yS(2)` und `zS(2)` und der dritte Satellit die Koordinaten `xS(3)`, `yS(3)` und `zS(3)` hat. Dadurch ist unmittelbar ersichtlich, wie die Komponenten eines Vektors in MATLAB angesprochen werden müssen. Als kleiner Nachteil dieser Vorgehensweise bleibt jedoch, dass die Bildschirmausgabe später bei der Verwendung von mehr als drei Satelliten angepasst werden muss, wenn man alle Satellitenkoordinaten angezeigt bekommen möchte.

Innerhalb der Überprüfungsfunktion wird für diesen Modellschritt die Testfunktion `teste_Satellitenkoordinaten` verwendet. Sie ist im Anhang auf Seite 73 abgedruckt.

4.4.3. Berechnen der Pseudoentfernungen

Im dritten Abschnitt werden die Pseudoentfernungen aus den Sendezeiten `tS` und den Empfangszeiten `tE` berechnet. Dazu ist folgender Programmabschnitt zu bearbeiten:

```
%% Schritt 2 | Berechnen der Pseudoentfernungen
display('Schritt 2 | Berechnen der Pseudoentfernungen')

%  Aufstellen der Berechnungsformel
berechne_Pseudoentfernungen = @(tS,tE) NaN;

%  Berechnen der Pseudoentfernungen
dS = berechne_Pseudoentfernungen(tS,tE);

%  Überprüfen der Pseudoentfernungen
ueberpruefe_Loesung(Phase,'Schritt 2',dS,Datensatz,Satellitenauswahl);

%  Anzeigen der Ergebnisse
format bank
display([dS(1) dS(2) dS(3)], ...
    'Pseudoentfernungen zu den Satelliten in Meter [dS(1) dS(2) dS(3)]')
```

Die Bildschirmausgabe nach dem Ausführen dieses Programmabschnitts ist wie folgt:

```
Schritt 2 | Berechnen der Pseudoentfernungen

Bitte bearbeite Schritt 2
Stelle die Formel für die Pseudoentfernungen auf.

Error using ueberpruefe_Loesung (line 43)

Bitte bearbeite Schritt 2
Ersetze das "NaN" in in der Zeile
    berechne_Pseudoentfernungen = @(tS,tE) NaN;
durch die richtige Formel.
```

Die Schülerinnen und Schüler sollen eine Formel für die Pseudoentfernungen herausfinden und ihre Lösung anstelle des Wertes `NaN` in die Funktion `berechne_Pseudoentfernungen` einsetzen. Sie ergibt sich aus Gleichung (1) auf Seite 13 wie folgt:

```
%  Aufstellen der Berechnungsformel
berechne_Pseudoentfernungen = @(tS,tE) (tE-tS)*299792458;
```

Bei der richtigen Lösung erhält man folgende Bildschirmausgabe:

```
Schritt 2 | Berechnen der Pseudoentfernungen

Schritt 2 scheint korrekt gelöst zu sein.

Pseudoentfernungen zu den Satelliten in Meter [dS(1) dS(2) dS(3)] =

   22717609.10   23165457.21   21000054.55
```

Bei einer falschen Formel weist die Überprüfungsfunktion auf einen Fehler hin und gibt in einigen Fällen auch deutliche Hinweise, z. B. wenn es sich um einen Vorzeichenfehler handelt oder wenn die Lichtgeschwindigkeit mit 300 000 000 m/s zu ungenau angegeben wurde. Infolge Ungenauigkeiten der Empfängeruhr kann es in seltenen Fällen vorkommen, dass die Empfangszeit `tE` vor der Sendezeit `tS` liegt. In so einem Fall nimmt die Pseudoentfernung `dS` negative Werte an. Deshalb darf bei der Berechnung der Pseudoentfernungen nicht die Betragsfunktion verwendet werden, da sonst eine spätere Korrektur der Uhrfehler nicht mehr möglich ist. Sollten negative Pseudoentfernungen durch die Verwendung der Betragsfunktion positiv geworden sein, wird das von der Überprüfungsfunktion erkannt und entsprechend kommentiert.

Didaktisch-methodische Anmerkungen: Die Berechnung der Pseudoentfernungen geschieht in diesem Abschnitt nicht direkt, sondern in zwei Schritten. Im ersten Schritt wird eine Funktion zur Berechnung definiert, im zweiten Schritt werden die Pseudoentfernungen mit dieser Funktion berechnet. Diese Aufteilung in zwei Schritte wurde bewusst vorgenommen. Zuerst erleichtert diese Trennung das Verständnis des Programmcodes, da der Name der Funktion `berechne_Pseudoentfernungen` selbsterklärend ist. Darüber hinaus bleibt innerhalb des Hauptprogramms eine gewisse Kontinuität bewahrt, denn in allen vier Schritten werden die wichtigen Größen über Funktionen berechnet. Hier können sich die Schülerinnen und Schüler an einer einfachen Funktion mit der Syntax von Funktionsdefinitionen in MATLAB vertraut machen, bevor im nächsten Schritt eine kompliziertere Funktion zur Darstellung eines Gleichungssystems aufgestellt werden muss.

Ein weiterer Vorteil von Funktionen ist, dass sie sich besser auf Korrektheit überprüfen lassen als Zahlen oder Vektoren. So ist z. B. folgende Lösung aufgrund der schon erwähnten fehlerhaften Verwendung der Betragsfunktion zwar nicht korrekt, führt jedoch in den meisten Fällen dennoch zum richtigen Ergebnis:

```
%   Aufstellen der Berechnungsformel
berechne_Pseudoentfernungen = @(tS,tE) abs(tE-tS)*299792458;
```

Nur in den seltenen Fällen, wo durch einen hinreichend großen Uhrfehler die Empfangszeit `tE` vor der Sendezeit `tS` liegt, erhält man ein falsches Ergebnis. Die fehlerhafte Verwendung der Betragsfunktion kann in den meisten Fällen anhand der berechneten Pseudoentfernungen `dS` somit nicht erkannt werden. Würde man jedoch die Funktion `berechne_Pseudoentfernungen` auf Korrektheit überprüfen, so könnte man diesen Fehler sehr einfach durch geeignete Testwerte für `tE` und `tS` aufspüren.

In der ersten Version der Überprüfungsfunktion wurden tatsächlich die Funktionen und nicht die Ergebnisse überprüft. Die spätere Version jedoch testet bewusst nur die Ergebnisse, damit die Schülerinnen und Schüler mehr Freiheiten bei den Lösungen haben. So kann man beispielsweise die Variablen `tS` und `tE` vertauschen:

```
%   Aufstellen der Berechnungsformel
berechne_Pseudoentfernungen = @(tE,tS) 299792458*(tE-tS);

%   Berechnen der Pseudoentfernungen
dS = berechne_Pseudoentfernungen(tE,tS);
```

Oder man kann eine weitere Variable für die Lichtgeschwindigkeit einführen:

```
%   Aufstellen der Berechnungsformel
berechne_Pseudoentfernungen = @(c,tS,tE) c*(tE-tS);

%   Berechnen der Pseudoentfernungen
dS = berechne_Pseudoentfernungen(299792458,tS,tE);
```

Bei der Überprüfung der Pseudoentfernungen `dS` spielen solche Veränderungen keine Rolle. Würden man jedoch die Funktion `berechne_Pseudoentfernungen` auf Korrektheit überprüfen, so müssten in der Überprüfungsfunktion sehr viele Fälle beachten werden um alle Möglichkeiten abzufangen, Variablen zu vertauschen, umzubenennen oder zusätzliche Variablen einzuführen. Man kann natürlich Fehlermeldungen erzeugen, wenn die Anzahl der Variablen nicht mehr stimmt oder wenn Variablen umbenannt oder vertauscht wurden. Eine solche Vorgehensweise würde jedoch die Freiheit der Schülerinnen und Schüler bei ihrer Lösungsfindung sehr einschränken. Deswegen wird hier bewusst darauf verzichtet, alle möglichen Fehlerquellen in der Berechnungsformel abzufangen. Die Formel wird einfach als korrekt anerkannt, wenn die Ergebnisse

richtig sind, wohl wissend, dass dies nicht immer der Fall sein muss. Darum sagt die Bildschirmausgabe bei einem richtigen Ergebnis auch nur, dass die Lösung korrekt zu sein scheint und nicht, dass sie wirklich richtig ist.

Innerhalb der Überprüfungsfunktion wird für diesen Modellschritt die Testfunktion `teste_Pseudoentfernungen` verwendet. Sie ist im Anhang auf Seite 74 abgedruckt.

4.4.4. Berechnen der Empfängerkoordinaten

Im vierten Abschnitt werden die Koordinaten des Empfängers berechnet. Dazu ist folgender Programmabschnitt zu bearbeiten:

```
%% Schritt 3 | Berechnen der Empfängerkoordinaten
display('Schritt 3 | Berechnen der Empfängerkoordinaten')

% Aufstellen des Gleichungssystems
Gleichungssystem = @(xE,yE,zE) [NaN
                                NaN
                                NaN];

% Überprüfen des Gleichungssystems
ueberpruefe_Loesung(Phase,'Schritt 3',Gleichungssystem,xS,yS,zS,dS);

% Definiere Funktion und Startpunkt für fsolve
Funktion = @(X) Gleichungssystem(X(1),X(2),X(3));
Startpunkt = [0 0 0];

% Berechnen einer Lösung in der Nähe des Startpunktes
X = fsolve(Funktion,Startpunkt);

% Speichern der berechneten Lösung
xE = X(1);
yE = X(2);
zE = X(3);

% Überprüfen der Empfängerkoordinaten
ueberpruefe_Loesung(Phase,'Schritt 3',xE,yE,zE,xS,yS,zS,dS);

% Anzeigen der Ergebnisse
format bank
display([xE yE zE],'Koordinaten des Empfängers in Meter [xE yE zE]')
clearvars Funktion Startpunkt X
```

Die Bildschirmausgabe nach dem Ausführen dieses Programmabschnitts ist wie folgt:

```
Schritt 3 | Berechnen der Empfängerkoordinaten

Bitte bearbeite Schritt 3
Stelle das Gleichungssystem auf.

Error using ueberpruefe_Loesung (line 43)

Bitte bearbeite Schritt 3
Ersetze die drei "NaN" in den Zeilen
    Gleichungssystem = @(xE,yE,zE) [NaN
                                    NaN
                                    NaN];
durch die richtigen Gleichungen.
Die Gleichungen müssen nach Null aufgelöst werden und
ohne Gleichheitszeichen und Nullen eingegeben werden.
```

Es muss ein Gleichungssystem für die unbekannten Koordinaten `xE`, `yE` und `zE` aufgestellt und gelöst werden. Die Lösung aus den Gleichungen (3) von Seite 14 kann direkt in eine MATLAB-Funktion umgeschrieben werden:

```
%   Aufstellen des Gleichungssystems
Gleichungssystem = @(xE,yE,zE) ...
    [sqrt((xS(1)-xE)^2+(yS(1)-yE)^2+(zS(1)-zE)^2)-dS(1)
     sqrt((xS(2)-xE)^2+(yS(2)-yE)^2+(zS(2)-zE)^2)-dS(2)
     sqrt((xS(3)-xE)^2+(yS(3)-yE)^2+(zS(3)-zE)^2)-dS(3)];
```

Dieses Gleichungssystem kann nicht direkt mit `fsolve` gelöst werden. Dazu muss es zuerst in eine Funktion von einem einzigen Vektor `X` mit den Komponenten `X(1)`, `X(2)` und `X(3)` umgewandelt werden. Außerdem muss ein Startpunkt für das Iterationsverfahren gewählt werden, in diesem Fall einfach der Erdmittelpunkt mit den Koordinaten `[0 0 0]`. Das geschieht durch folgenden Programmabschnitt:

```
%   Definiere Funktion und Startpunkt für fsolve
Funktion = @(X) Gleichungssystem(X(1),X(2),X(3));
Startpunkt = [0 0 0];
```

Die Lösung des nichtlinearen Gleichungssystems geschieht danach durch den Aufruf von `fsolve` aus der Optimization Toolbox, wobei als Parameter die Funktion und der Startpunkt übergeben werden müssen. Bei der richtigen Lösung erhält man folgende Bildschirmausgabe:

```
Schritt 3  |  Berechnen der Empfängerkoordinaten

Das Gleichungssystem scheint korrekt zu sein.

Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>

Schritt 3 scheint korrekt gelöst zu sein.

Koordinaten des Empfängers in Meter [xE yE zE] =

    3993459.69      725545.77     4896426.57
```

Fehler im Gleichungssystem werden dabei von der Überprüfungsfunktion erkannt und kommentiert. Sollte mindestens eine der drei Gleichungen richtig sein, so wird das ebenfalls erkannt und kommentiert. Folgendes zur Musterlösung äquivalentes Gleichungssystem wird ebenfalls von der Überprüfungsfunktion als richtig erkannt:

```
%   Aufstellen des Gleichungssystems
Gleichungssystem = @(xE,yE,zE) ...
    [(xS(1)-xE)^2+(yS(1)-yE)^2+(zS(1)-zE)^2-dS(1)^2
     (xS(2)-xE)^2+(yS(2)-yE)^2+(zS(2)-zE)^2-dS(2)^2
     (xS(3)-xE)^2+(yS(3)-yE)^2+(zS(3)-zE)^2-dS(3)^2];
```

Alternativ ist auch folgende Darstellung mit dem Vektorbetrag `norm` möglich:

```
%   Aufstellen des Gleichungssystems
Gleichungssystem = @(xE,yE,zE) ...
    [norm([xS(1) yS(1) zS(1)]-[xE yE zE])-dS(1)
     norm([xS(2) yS(2) zS(2)]-[xE yE zE])-dS(2)
     norm([xS(3) yS(3) zS(3)]-[xE yE zE])-dS(3)];
```

Diese Darstellung sieht zwar sehr elegant aus, hat jedoch den Nachteil, dass sie später bei der vierten Modellverbesserung nicht für eine Verallgemeinerung auf beliebig viele Satelliten geeignet ist. Die anderen Darstellungen lassen sich für jede beliebige Anzahl von Satelliten in MATLAB durch explizite Verwendung der Vektoren `xS`, `yS`, `zS` und `dS` kompakt ausdrücken:

```
%   Aufstellen des Gleichungssystems
Gleichungssystem = @(xE,yE,zE) sqrt((xS-xE).^2+(yS-yE).^2+(zS-zE).^2)-dS;
```

Das äquivalente Gleichungssystem in quadrierter Form lautet für eine beliebige Anzahl von Satelliten:

```
%   Aufstellen des Gleichungssystems
Gleichungssystem = @(xE,yE,zE) (xS-xE).^2+(yS-yE).^2+(zS-zE).^2-dS.^2;
```

Eine solche von der Anzahl der Satelliten unabhängige Darstellung ist bei der Verwendung der `norm`-Funktion nicht möglich.

Didaktisch-methodische Anmerkungen: In diesem Abschnitt befindet sich die erste Aufgabe mit einem höheren Schwierigkeitsgrad. Ein Gleichungssystem für die drei Unbekannten `xE`, `yE` und `zE` muss von den Schülerinnen und Schülern aufgestellt und für die Verarbeitung mir MATLAB geeignet eingegeben werden. Für diese Aufgabe stehen vier Hilfekarten zur Verfügung, die von den Betreuern bei Bedarf an die Schülerinnen und Schüler verteilt werden. Die Hilfekarten sind im Anhang auf den Seiten 95 und 96 abgebildet.

Nach dem Aufstellen des Gleichungssystems muss es so in MATLAB eingeben werden, dass es durch `fsolve` gelöst werden kann. Da `fsolve` nur Nullstellen berechnen kann, muss das Gleichungssystem als Nullstellenproblem dargestellt werden. Außerdem müssen für `fsolve` alle unbekannten Variablen zu einem Vektor zusammengefasst werden. Damit die Schülerinnen und Schüler nicht die drei Unbekannten `xE`, `yE` und `zE` durch einen Vektor `X` mit den Komponenten `X(1)`, `X(2)` und `X(3)` im Gleichungssystem ersetzen müssen, wird hier ein Umweg beschritten. Im Gleichungssystem werden die Unbekannten `xE`, `yE` und `zE` verwendet:

```
Gleichungssystem = @(xE,yE,zE) ...
    [sqrt((xS(1)-xE)^2+(yS(1)-yE)^2+(zS(1)-zE)^2)-dS(1)
     sqrt((xS(2)-xE)^2+(yS(2)-yE)^2+(zS(2)-zE)^2)-dS(2)
     sqrt((xS(3)-xE)^2+(yS(3)-yE)^2+(zS(3)-zE)^2)-dS(3)];
```

Danach wird aus dem Gleichungssystem eine Funktion erzeugt, die nur von einem Vektor `X` abhängig ist:

```
Funktion = @(X) Gleichungssystem(X(1),X(2),X(3));
```

Durch diesen Umweg bleibt das Gleichungssystem gut lesbar, welches ansonsten folgende, durch die vielen Indizes schwer zu verstehende Darstellung hätte:

```
Gleichungssystem = @(X) ...
    [sqrt((xS(1)-X(1))^2+(yS(1)-X(2))^2+(zS(1)-X(3))^2)-dS(1)
     sqrt((xS(2)-X(1))^2+(yS(2)-X(2))^2+(zS(2)-X(3))^2)-dS(2)
     sqrt((xS(3)-X(1))^2+(yS(3)-X(2))^2+(zS(3)-X(3))^2)-dS(3)];
```

In diesem Abschnitt finden zwei Überprüfungen statt. Hier genügt es nicht, nur die Lösungen zu überprüfen, denn wenn es Fehler im Gleichungssystem gibt, bricht die Funktion `fsolve` selber das Programm ab und gibt Fehlermeldungen aus, die nicht immer einfach zu verstehen sind. So würde bei vorhandenen Platzhaltern `NaN` nur

die Meldung erscheinen, dass es undefinierte Werte am Startpunkt gibt und nicht die Aufforderung, das Gleichungssystem zu bearbeiten. Deswegen wird das System vor der Anwendung von `fsolve` überprüft, um verständlichere Fehlermeldungen anzeigen zu können. Erst wenn das Gleichungssystem einige Tests besteht und nicht mehr ganz falsch sein kann, wird die erste Überprüfung beendet und das Hauptprogramm weiter ausgeführt. Es wird innerhalb der Überprüfungsfunktion bewusst kein Aufruf von `fsolve` an dem möglicherweise dennoch fehlerhaften Gleichungssystem durchgeführt, weil es dabei zu nicht-interpretierbaren Fehlermeldungen kommen kann. Stattdessen wird `fsolve` nur im Hauptprogramm aufgerufen und die berechnete Lösung durch einen zweiten Aufruf der Überprüfungsfunktion auf Korrektheit überprüft. Sollten es im Hauptprogramm wegen eines fehlerhaften Gleichungssystems zu einem Abbruch von `fsolve` kommen, lassen sich die ausgegebenen Fehlermeldungen leichter interpretieren, als wenn die nicht einsehbare Überprüfungsfunktion bei dem Versuch, das System zu lösen in ein Problem gerät und mit einer nicht nachvollziehbaren Fehlermeldung abbricht.

Innerhalb der Überprüfungsfunktion werden für diesen Modellschritt die Testfunktionen `teste_Gleichungssystem` und `teste_Empfaengerkoordinaten` verwendet. Sie sind im Anhang auf den Seiten 76 und 79 abgedruckt.

4.4.5. Umrechnen in geographische Höhe, Breite und Länge

Im fünften Abschnitt werden die Koordinaten des Empfängers in geographische Höhe, Breite und Länge umgerechnet. Dazu ist folgender Programmabschnitt zu bearbeiten:

```
%% Schritt 4 | Umrechnen in geographische Höhe, Breite und Länge
display('Schritt 4 | Umrechnen in geographische Höhe, Breite und Länge')

% Aufstellen der Umrechnungsformeln
berechne_h      = @(xE,yE,zE) NaN;
berechne_phi    = @(xE,yE,zE) NaN;
berechne_lambda = @(xE,yE,zE) NaN;

% Berechnen der geographischen Koordinaten
h      = berechne_h(xE,yE,zE);
phi    = berechne_phi(xE,yE,zE);
lambda = berechne_lambda(xE,yE,zE);

% Überprüfen der geographischen Koordinaten
ueberpruefe_Loesung(Phase,'Schritt 4',h,xE,yE,zE);
ueberpruefe_Loesung(Phase,'Schritt 4',phi,xE,yE,zE);
ueberpruefe_Loesung(Phase,'Schritt 4',lambda,xE,yE,zE);

% Anzeigen der Ergebnisse
format shortG
display(h,'Geographische Höhe des Empfängers in Meter [h]')
display([phi lambda], ...
    'Geographische Breite und Länge des Empfängers in Grad [phi lambda]')
```

Die Bildschirmausgabe nach dem Ausführen dieses Programmabschnitts lautet wie folgt:

```
Schritt 4 | Umrechnen in geographische Höhe, Breite und Länge

Bitte bearbeite Schritt 4
Stelle die Umrechnungsformel für die geographische Höhe auf.

Error using ueberpruefe_Loesung (line 47)
```

```
Bitte bearbeite Schritt 4
Ersetze das "NaN" in der Zeile
    berechne_h = @(xE,yE,zE) NaN;
durch die richtige Formel.
```

Auch hier sollen die Platzhalter `NaN` in den Umrechnungsformeln durch die richtigen Formeln ersetzt werden. Die Überprüfung geschieht dabei in der Reihenfolge `h` vor `phi` vor `lambda`, jedoch kann diese im Hauptprogramm beliebig verändert werden. Die Überprüfungsfunktion erkennt fehlerhafte Lösungen und gibt bei folgenden Fehlern konkrete Hinweise:

- Wenn bei der Berechnung der Höhe der Erdradius nicht gleich dem Äquatorradius von 6378137.0 Meter oder dem mittleren Erdradius von 6371000.8 Meter angenommen wurde.
- Wenn die Winkel für die geographische Breite ϕ und Länge λ im Bogenmaß und nicht im Gradmaß berechnet wurden.
- Wenn bei der Berechnung der geographischen Breite vorausgesetzt wurde, dass sich der Empfänger auf der Oberfläche der Erdkugel befindet. Diese Annahme ist unzulässig, da die Erde keine perfekte Kugel ist und es sowohl Berge als auch Täler gibt.

Da die Empfängerkoordinate $x_E > 0$ ist, lauten die korrekten Umrechnungsformeln nach Formel (6) auf Seite 15:

```
%   Aufstellen der Umrechnungsformeln
berechne_h      = @(xE,yE,zE) sqrt(xE^2+yE^2+zE^2)-6378137.0;
berechne_phi    = @(xE,yE,zE) atan(zE/sqrt(xE^2+yE^2))*180/pi;
berechne_lambda = @(xE,yE,zE) atan(yE/xE)*180/pi;
```

Im Allgemeinen sollte jedoch bei der Berechnung von `lambda` besser der atan2 verwendet werden. Bei der Berechnung von `phi` ist das nicht notwendig, da MATLAB die Norm IEEE 754 für die Rechnung mit Gleitkommazahlen berücksichtigt. Innerhalb dieser Norm gelten die Gleichungen atan(1/0) = $\pi/2$ und atan(-1/0) = -$\pi/2$. Deshalb kann bei der Berechnung von `phi` auf den `atan2` verzichtet werden. Die Formeln (10) auf Seite 16 sind innerhalb der Norm IEEE 754 somit immer anwendbar:

```
%   Aufstellen der Umrechnungsformeln
berechne_h      = @(xE,yE,zE) sqrt(xE^2+yE^2+zE^2)-6378137.0;
berechne_phi    = @(xE,yE,zE) atan(zE/sqrt(xE^2+yE^2))*180/pi;
berechne_lambda = @(xE,yE,zE) atan2(yE,xE)*180/pi;
```

Die Bildschirmausgabe bei der richtigen Lösung ist:

```
Schritt 4  |  Umrechnen in geographische Höhe, Breite und Länge

Die Formel für "h" scheint korrekt zu sein.
Die Formel für "phi" scheint korrekt zu sein.
Die Formel für "lambda" scheint korrekt zu sein.

Geographische Höhe des Empfängers in Meter [h] =

      -18174

Geographische Breite und Länge des Empfängers in Grad [phi lambda] =

      50.343       10.297
```

Didaktisch-methodische Anmerkungen: Für diesen Schritt gibt es zwei Hilfekarten, welche im Anhang auf Seite 97 dargestellt sind. Die erste Hilfekarte zeigt eine deutlichere Darstellung der Winkel als die Skizze auf dem Aufgabenblatt, die zweite Hilfekarte erklärt die Umrechnung vom Bogenmaß ins Gradmaß.

Auch hier werden die geographischen Koordinaten nicht direkt, sondern über drei Funktionen `berechne_h`, `berechne_phi` und `berechne_lambda` ermittelt. Dadurch könnte man die Funktionen auf Korrektheit überprüfen, was besonders bei der Funktion `berechne_lambda` sinnvoll wäre, da die korrekte Lösung mit dem `atan2` und nicht mit dem `atan` formuliert werden muss. Aus den schon genannten Gründen wird auch an dieser Stelle auf eine Überprüfung der Funktionen verzichtet und stattdessen werden nur die Ergebnisse überprüft.

MATLAB verwendet bei den Winkelfunktionen `sin`, `cos`, `tan` und den Umkehrfunktionen `asin`, `acos`, `atan` und `atan2` das Bogenmaß. Deswegen müssen die beiden Winkel `phi` und `lambda` durch eine nachträgliche Multiplikation mit dem Faktor $180/\pi$ in das Gradmaß überführt werden. Alternativ können Winkelberechnungen mit den MATLAB-Funktionen `sind`, `cosd`, `tand`, `asind`, `acosd`, `atand` und `atan2d` direkt im Gradmaß vorgenommen werden. Jedoch werden bewusst keinerlei Hinweise auf diese alternativen Winkelfunktionen gegeben, da die Schülerinnen und Schüler sich mit den beiden Winkelmaßen und den notwendigen Umrechnungen selber auseinandersetzen sollen. Durch die Überprüfungsfunktion werden sie auch auf falsche Winkelmaße hingewiesen.

Die Höhe `h` wird vor der geographischen Breite `phi` überprüft, damit die oben angesprochene Fehlvorstellung (Berge und Täler) bei der Berechnung der geographischen Breite `phi` erkannt werden kann. Denn bei der Überprüfung der Höhe wird bemerkt, wenn die Schülerinnen und Schüler einen falschen Wert für den Erdradius verwenden. Sie werden in diesem Fall darauf hingewiesen, entweder den Äquatorradius 6378137.0 Meter oder den mittleren Erdradius 6371000.8 Meter zu benutzen. Die Verwendung einer dieser beiden Werte für den Radius ist die Voraussetzung dafür, dass die Fehlvorstellung bei der Berechnung von `phi` erkannt werden kann. Trotzdem können die Formeln auch in einer anderen Reihenfolge bearbeitet werden, da die Reihenfolge der Überprüfungen im Hauptprogramm leicht veränderbar ist.

Innerhalb der Überprüfungsfunktion werden für diesen Modellschritt die Testfunktionen `teste_h`, `teste_phi` und `teste_lambda` verwendet. Sie sind im Anhang auf den Seiten 80, 81 und 83 abgedruckt.

4.4.6. Anzeigen der Empfängerposition

Die geographische Breite und Länge werden im letzten Abschnitt auf einer Weltkarte angezeigt. Das geschieht über folgenden Programmabschnitt:

```
%% Anzeigen der Empfängerposition
display('Anzeigen der Empfängerposition')

%  Auswählen des Kartenausschnitts
Kartenausschnitt_manuell = true;
Zentrum = '50.5,6.8';
Zoom = '8';

%  Erzeugen der URL
url = sprintf('https://www.google.de/maps/place/%f,%f',phi,lambda);
if Kartenausschnitt_manuell
    url = [url '/@' Zentrum ',' Zoom 'z'];
end
```

```
%   Anzeigen der URL im Browser
web(url,'-browser')
display(url,'URL für die Anzeige im Webbrowser')
clearvars Kartenausschnitt_manuell Zentrum Zoom url
```

Die Anzeige der Empfängerposition erfolgt hier mit dem Online-Kartendienst Google Maps. Alternativ könnte auch die freie Weltkarte von OpenStreetMap verwendet werden. Durch den Schalter `Kartenausschnitt_manuell` lässt sich einstellen, ob der Ausschnitt der Weltkarte manuell oder automatisch gewählt werden soll. Bei manueller Wahl muss noch das Zentrum und die Vergrößerungsstufe mit den beiden Variablen `Zentrum` und `Zoom` eingestellt werden. Unabhängig davon landet man 300 Kilometer östlich von Aachen in einem Waldgebiet in der Nähe von Hollstadt. Der Empfänger soll sich dort 18 Kilometer tief im Erdboden befinden.

4.5. Die Modellverbesserungen

In der zweiten Phase der Modellverbesserungen wird die Überprüfungsfunktion nicht mehr benötigt. Stattdessen können die Schülerinnen und Schüler ihre Lösungen anhand von Vergleichswerten selbst überprüfen. Zu Beginn dieser Arbeitsphase muss deswegen die Überprüfung der Lösungen durch die Überprüfungsfunktion deaktiviert werden. Dies geschieht dadurch, dass am Anfang des Hauptprogramms die Zeile `Phase = 'Modellschritt'` durch die Zeile `Phase = 'Modellverbesserung'` ersetzt wird oder dadurch, dass alle Aufrufe der Funktion `ueberpruefe_Loesung` aus dem Hauptprogramm entfernt werden. Die Aufforderung, „Modellschritt“ durch „Modellverbesserung“ zu ersetzen, ist auf dem Arbeitsblatt zu den Modellverbesserungen sichtbar vermerkt. Das Arbeitsblatt ist im Anhang auf den Seiten 100 und 101 aufgeführt.

4.5.1. Fehler der Satellitenuhren

Die Berechnung der Satellitenuhrfehler aus den Ephemeriden ist zu aufwendig um sie im Zeitrahmen des Modellierungstages selber programmieren zu lassen. Die Funktion `Delta_t = berechne_Satellitenuhrfehler(t,Ephemeriden)` ist in der Funktionsdatei `berechne_Satellitenuhrfehler.m` schon vorhanden und kann von den Schülerinnen und Schülern eingesehen und verwendet werden. Sie berechnet zu einem vorgegebenen Zeitvektor `t` die Satellitenuhrfehler `Delta_t` aus der Ephemeridenmatrix gemäß Gleichung (12) auf Seite 17. Die Funktionsdatei ist im Anhang auf Seite 71 abgedruckt. Die Berechnung der Satellitenuhrfehler wird am besten zu Beginn des ersten Schrittes vorgenommen. Wie im Abschnitt 3.3.1 erklärt wird, müssen diese Fehler zu den wahren Sendezeiten der Nachrichten berechnet werden. Da die Variable `tS` jedoch die falschen Sendezeiten enthält, müssen die Satellitenuhrfehler wie in Formel (17) auf Seite 17 iterativ berechnet werden. Ein Iterationsschritt ist dabei wegen der schnellen Konvergenz genug:

```
%   Berechnung der Satellitenuhrfehler (sehr gute Genauigkeit)
Delta_tS = berechne_Satellitenuhrfehler(tS,Ephemeriden);
Delta_tS = berechne_Satellitenuhrfehler(tS-Delta_tS,Ephemeriden);
```

Der Vektor der Satellitenuhrfehler zu den wahren Sendezeit wird hier einfach mit `Delta_tS` bezeichnet. Die wahren Sendezeiten berechnen sich nach Formel (19) auf Seite 18 zu `tS-Delta_tS`. Die Schülerinnen und Schüler machen glücklicherweise nur einen kleinen Fehler, wenn sie bei der Berechnung der Satellitenuhrfehler die Iteration

vergessen und diese Fehler zu den falschen Sendezeiten `tS` berechnen. Nach Formel (18) auf Seite 18 ist die folgende Lösung in der Praxis genau genug:

```
%   Berechnung der Satellitenuhrfehler (ausreichende Genauigkeit)
Delta_tS = berechne_Satellitenuhrfehler(tS,Ephemeriden);
```

Diese müssen sowohl bei der Berechnung der Satellitenkoordinaten im ersten Schritt als auch bei der Berechnung der Pseudoentfernungen im zweiten Schritt verwendet werden:

```
%   Berechnen der Satellitenkoordinaten
[xS,yS,zS] = berechne_Satellitenkoordinaten(tS-Delta_tS,Ephemeriden);
```

```
%   Berechnen der Pseudoentfernungen
dS = berechne_Pseudoentfernungen(tS-Delta_tS,tE);
```

Für die erste Modellverbesserung sind somit folgende Korrekturen am Hauptprogramm notwendig. Die Bildschirmausgabe der Satellitenuhrfehler wird dabei nicht von den Schülerinnen und Schüler gefordert:

```
%% Schritt 1  |  Berechnen der Satellitenkoordinaten
display('Schritt 1  |  Berechnen der Satellitenkoordinaten')

%   Berechnung der Satellitenuhrfehler
Delta_tS = berechne_Satellitenuhrfehler(tS,Ephemeriden);
Delta_tS = berechne_Satellitenuhrfehler(tS-Delta_tS,Ephemeriden);

%   Berechnen der Satellitenkoordinaten
[xS,yS,zS] = berechne_Satellitenkoordinaten(tS-Delta_tS,Ephemeriden);

%   Anzeigen der Ergebnisse
format shortG
display(Delta_tS, 'Fehler der Satellitenuhren in Sekunden [Delta_tS]')
format bank
display([xS(1) yS(1) zS(1)], ...
    'Koordinaten des ersten Satelliten in Meter [xS(1) yS(1) zS(1)]')
display([xS(2) yS(2) zS(2)], ...
    'Koordinaten des zweiten Satelliten in Meter [xS(2) yS(2) zS(2)]')
display([xS(3) yS(3) zS(3)], ...
    'Koordinaten des dritten Satelliten in Meter [xS(3) yS(3) zS(3)]')

%% Schritt 2  |  Berechnen der Pseudoentfernungen
display('Schritt 2  |  Berechnen der Pseudoentfernungen')

%   Aufstellen der Berechnungsformel
berechne_Pseudoentfernungen = @(tS,tE) (tE-tS)*299792458;

%   Berechnen der Pseudoentfernungen
dS = berechne_Pseudoentfernungen(tS-Delta_tS,tE);

%   Anzeigen der Ergebnisse
format bank
display([dS(1) dS(2) dS(3)], ...
    'Pseudoentfernungen zu den Satelliten in Meter [dS(1) dS(2) dS(3)]')
```

Didaktisch-methodische Anmerkungen: Zu dieser Modellverbesserung gibt es drei Hilfekarten, welche im Anhang auf Seite 102 eingesehen werden können. Die eigentliche Schwierigkeit bei dieser Modellverbesserung ist das Erkennen, dass die Fehler der Satellitenuhren nicht konstant, sondern zeitabhängig sind und zur wahren Sendezeit

und damit notgedrungen iterativ berechnet werden müssen. Da jedoch die Berechnung der Satellitenuhrfehler zu den falschen Sendezeiten `tS` schon zu einem ausreichend präzisem Ergebnis führt, ist diese Modellverbesserung sehr einfach durchzuführen. Deshalb wird auf dem Aufgabenblatt bewusst kein Hinweis darauf gegeben, an welcher Stelle im Hauptprogramm die Korrektur der Satellitenuhrfehler durchgeführt werden soll.

In der Vergangenheit zeigte sich, dass einige Schülerinnen und Schüler Schwierigkeiten mit der Syntax des Funktionsaufrufs zur Berechnung der Satellitenuhrfehler hatten, obwohl die Syntax in der Funktionsdatei selbst erklärt wird (siehe den Programmausdruck auf Seite 71 im Anhang). Deshalb wurde entschieden, die Berechnung der Satellitenuhrfehler schon fest in das Hauptprogramm mit einzubauen und die Zeile `Delta_tS = berechne_Satellitenuhrfehler(tS,Ephemeriden)` wurde im ersten Schritt ergänzt:

```
%% Schritt 1 | Berechnen der Satellitenkoordinaten
display('Schritt 1 | Berechnen der Satellitenkoordinaten')

Delta_tS = berechne_Satellitenuhrfehler(tS,Ephemeriden);

%  Berechnen der Satellitenkoordinaten
[xS,yS,zS] = berechne_Satellitenkoordinaten(NaN,Ephemeriden);

%  Anzeigen der Ergebnisse
format bank
display([xS(1) yS(1) zS(1)], ...
    'Koordinaten des ersten Satelliten in Meter [xS(1) yS(1) zS(1)]')
display([xS(2) yS(2) zS(2)], ...
    'Koordinaten des zweiten Satelliten in Meter [xS(2) yS(2) zS(2)]')
display([xS(3) yS(3) zS(3)], ...
    'Koordinaten des dritten Satelliten in Meter [xS(3) yS(3) zS(3)]')
```

Die Existenz der Zeile `Delta_tS = berechne_Satellitenuhrfehler(tS,Ephemeriden)` im Hauptprogramm führt jedoch direkt zu mehreren Problemen:

- Dadurch, dass die Satellitenuhrfehler schon vorab zu dem Zeitpunkt `tS` berechnet werden wird auch schon die Lösung der ersten Aufgabe verraten. In dieser Aufgabe sollen die Schülerinnen und Schüler erstens selbst feststellen, dass die Satellitenkoordinaten zu den Sendezeiten benötigt werden und zweitens herausfinden, wie sie in MATLAB die Sendezeiten korrekt in den Funktionsaufruf einsetzen müssen (vgl. dazu Abschnitt 4.4.2 ab Seite 28).

- Stattdessen kann Verwirrung erzeugt werden: Durch die Überlegung, dass die Sendezeitpunkte benötigt werden, entsteht die Problematik, ob diese Zeitpunkte durch `tS`, durch `tS-Delta_tS` oder durch `tS+Delta_tS` angegeben werden müssten. Solche Gedanken wären zu diesem Zeitpunkt irreführend, da sie erst später bei den Modellverbesserungen zum Thema werden.

- Die erste Modellverbesserung wird durch die Existenz dieser Zeile im Hauptprogramm schon in die Phase der Modellschritte vorverlegt.

- Schließlich müsste durch die Existenz dieser Zeile im Hauptprogramm die Überprüfungsfunktion so angepasst werden, dass sie wahren Sendezeiten `tS-Delta_tS` auch als korrekt erkennt und mit einem entsprechenden Kommentar versieht. Bisher werden die wahren Sendezeiten `tS-Delta_tS` von der Überprüfungsfunktion als falsch angesehen, da in der Phase der Modellschritte keine Modellverbesserungen berücksichtigt werden.

Fazit: Die Berechnung der Satellitenuhrfehler darf nicht im Hauptprogramm in der Phase der Modellschritte vorweggenommen werden, sondern muss von den Schülerinnen und Schülern erst während der ersten Modellverbesserung an die richtige Stelle eingefügt werden. Um den Schwierigkeiten mit der Syntax des Funktionsaufrufs zu begegnen, wurde das Aufgabenblatt überarbeitet und die Zeile `Delta_t = berechne_Satellitenuhrfehler(t,Ephemeriden)` im Aufgabentext ergänzt (siehe dazu das Arbeitsblatt zu den Modellverbesserungen im Anhang auf Seite 100). Um nicht alles vorwegzunehmen, wurde bewusst nicht die Zeile `Delta_tS = berechne_Satellitenuhrfehler(tS,Ephemeriden)` in das Aufgabenblatt gestellt.

4.5.2. Fehler der Empfängeruhr

In dieser Modellverbesserung ist der Arbeitsauftrag an die Schülerinnen und Schüler präziser. Sie sollen im dritten Schritt das Gleichungssystem so verbessern, dass der unbekannte Fehler der Empfängeruhr `Delta_tE` in den Gleichungen berücksichtigt wird. Da im Gleichungssystem nun vier Unbekannte vorkommen, müssen auch vier Gleichungen von vier Satelliten verwendet werden. Dazu muss die Satellitenauswahl beim Einlesen der Satellitendaten auf vier Satelliten erweitert werden:

```
%   Auswählen der Satelliten
Satellitenauswahl = [1 2 3 4];
```

Das korrigierte Gleichungssystem lautet nach Formel (20) auf Seite 18:

```
%   Aufstellen des Gleichungssystems
c = 299792458;
Gleichungssystem = @(xE,yE,zE,Delta_tE) ...
    [sqrt((xS(1)-xE)^2+(yS(1)-yE)^2+(zS(1)-zE)^2)-(dS(1)-c*Delta_tE)
     sqrt((xS(2)-xE)^2+(yS(2)-yE)^2+(zS(2)-zE)^2)-(dS(2)-c*Delta_tE)
     sqrt((xS(3)-xE)^2+(yS(3)-yE)^2+(zS(3)-zE)^2)-(dS(3)-c*Delta_tE)
     sqrt((xS(4)-xE)^2+(yS(4)-yE)^2+(zS(4)-zE)^2)-(dS(4)-c*Delta_tE)];
```

Die Multiplikation mit der Lichtgeschwindigkeit $c = 299792458\,\mathrm{m/s}$ im Gleichungssystem führt jedoch zu einem Genauigkeitsverlust bei der numerischen Lösung. Deshalb sollte das Gleichungssystem nicht mit dem Empfängeruhrfehler `Delta_tE`, sondern mit dem Fehler der Pseudoentfernungen `Delta_dS` gemäß Gleichung (21) auf Seite 18 dargestellt werden:

```
%   Aufstellen des Gleichungssystems
Gleichungssystem = @(xE,yE,zE,Delta_dS) ...
    [sqrt((xS(1)-xE)^2+(yS(1)-yE)^2+(zS(1)-zE)^2)-(dS(1)-Delta_dS)
     sqrt((xS(2)-xE)^2+(yS(2)-yE)^2+(zS(2)-zE)^2)-(dS(2)-Delta_dS)
     sqrt((xS(3)-xE)^2+(yS(3)-yE)^2+(zS(3)-zE)^2)-(dS(3)-Delta_dS)
     sqrt((xS(4)-xE)^2+(yS(4)-yE)^2+(zS(4)-zE)^2)-(dS(4)-Delta_dS)];
```

Natürlich müssen auch die Funktion und der Startpunkt für `fsolve` angepasst werden:

```
%   Definiere Funktion und Startpunkt für fsolve
Funktion = @(X) Gleichungssystem(X(1),X(2),X(3),X(4));
Startpunkt = [0 0 0 0];
```

Für die zweite Modellverbesserung sind insgesamt folgende Korrekturen am Hauptprogramm notwendig. In dieser Lösung sind zusätzlich die Bildschirmausgaben angepasst worden, was von den Schülerinnen und Schülern nicht gefordert wird:

```
%% Einlesen der Satellitendaten
display('Einlesen der Satellitendaten')
```

```
%   Einlesen der Satellitendaten falls nicht vorhanden
if ~exist('Satellitendaten','var')
    Dateien = {'114255.obs','114255.nav'};
    Satellitendaten = einlesen_Satellitendaten(Dateien);
end

%   Auswählen eines Datensatzes
Datensatznummer = 1;
Datensatz = Satellitendaten(Datensatznummer);

%   Satelliten, deren Nachrichten empfangen wurden
Satelliten = Datensatz.Satelliten;

%   Auswählen der Satelliten
Satellitenauswahl = [1 2 3 4];

%   Auslesen der ausgewählten Satellitendaten
tE = Datensatz.Empfangszeiten(Satellitenauswahl);
tS = Datensatz.Sendezeiten(Satellitenauswahl);
Ephemeriden = Datensatz.Ephemeriden(:,Satellitenauswahl);

%   Anzeigen der Ergebnisse
format shortG
display(Datensatz,sprintf('Datensatz Nummer %d',Datensatznummer))
display(Satelliten, ...
    'Satelliten, deren Nachrichten empfangen wurden [Satelliten]')
display(Satellitenauswahl, ...
    'Nummern der ausgewählten Satelliten [Satellitenauswahl]')
format bank
display(tE, ...
    'Empfangszeiten der Nachrichten in Sekunden seit Wochenbeginn [tE]')
display(tS, ...
    'Sendezeiten der Nachrichten in Sekunden seit Wochenbeginn [tS]')
display(['Die Positionsdaten der Satelliten werden nicht angezeigt ', ...
    '[Ephemeriden]'])
clearvars Dateien Datensatznummer

%% Schritt 1  |  Berechnen der Satellitenkoordinaten
display('Schritt 1  |  Berechnen der Satellitenkoordinaten')

%   Berechnen der Satellitenkoordinaten
[xS,yS,zS] = berechne_Satellitenkoordinaten(tS,Ephemeriden);

%   Anzeigen der Ergebnisse
format bank
display([xS(1) yS(1) zS(1)], ...
    'Koordinaten des ersten Satelliten in Meter [xS(1) yS(1) zS(1)]')
display([xS(2) yS(2) zS(2)], ...
    'Koordinaten des zweiten Satelliten in Meter [xS(2) yS(2) zS(2)]')
display([xS(3) yS(3) zS(3)], ...
    'Koordinaten des dritten Satelliten in Meter [xS(3) yS(3) zS(3)]')
display([xS(4) yS(4) zS(4)], ...
    'Koordinaten des vierten Satelliten in Meter [xS(4) yS(4) zS(4)]')

%% Schritt 2  |  Berechnen der Pseudoentfernungen
display('Schritt 2  |  Berechnen der Pseudoentfernungen')

%   Aufstellen der Berechnungsformel
berechne_Pseudoentfernungen = @(tS,tE) (tE-tS)*299792458;

%   Berechnen der Pseudoentfernungen
```

```
dS = berechne_Pseudoentfernungen(tS,tE);

%   Anzeigen der Ergebnisse
format bank
display([dS(1) dS(2) dS(3) dS(4)], ['Pseudoentfernungen zu den ', ...
    'Satelliten in Meter [dS(1) dS(2) dS(3) dS(4)]'])

%% Schritt 3 | Berechnen der Empfängerkoordinaten
display('Schritt 3 | Berechnen der Empfängerkoordinaten')

%   Aufstellen des Gleichungssystems
Gleichungssystem = @(xE,yE,zE,Delta_dS) ...
   [sqrt((xS(1)-xE)^2+(yS(1)-yE)^2+(zS(1)-zE)^2)-(dS(1)-Delta_dS)
    sqrt((xS(2)-xE)^2+(yS(2)-yE)^2+(zS(2)-zE)^2)-(dS(2)-Delta_dS)
    sqrt((xS(3)-xE)^2+(yS(3)-yE)^2+(zS(3)-zE)^2)-(dS(3)-Delta_dS)
    sqrt((xS(4)-xE)^2+(yS(4)-yE)^2+(zS(4)-zE)^2)-(dS(4)-Delta_dS)];

%   Definiere Funktion und Startpunkt für fsolve
Funktion = @(X) Gleichungssystem(X(1),X(2),X(3),X(4));
Startpunkt = [0 0 0 0];

%   Berechnen einer Lösung in der Nähe des Startpunktes
X = fsolve(Funktion,Startpunkt);

%   Speichern der berechneten Lösung
xE = X(1);
yE = X(2);
zE = X(3);
Delta_tE = X(4)/299792458;

%   Anzeigen der Ergebnisse
format shortG
display(Delta_tE,'Fehler der Empfängeruhr in Sekunden [Delta_tE]')
format bank
display([xE yE zE],'Koordinaten des Empfängers in Meter [xE yE zE]')
clearvars Funktion Startpunkt X
```

Didaktisch-methodische Anmerkungen: Zu dieser Modellverbesserung gibt es zwei Hilfekarten, welche im Anhang auf Seite 102 eingesehen werden können. Die eigentliche Schwierigkeit bei dieser Aufgabe liegt darin, zu erkennen, dass alle Pseudoentfernungen um die Strecke $c \cdot \Delta t_E$ korrigiert werden müssen, wenn der Empfängeruhrfehler gleich Δt_E ist. Ist Δt_E positiv, so geht die Empfängeruhr vor und alle Pseudoentfernungen sind zu lang. Ist Δt_E negativ, so geht die Empfängeruhr nach und alle Pseudoentfernungen sind zu kurz. Die korrigierten Pseudoentfernungen sind somit gegeben durch $d_S - c \cdot \Delta t_E$. Diese Überlegungen werden auf der zweiten Hilfekarte angedeutet. Danach ist die Korrektur der Pseudoentfernungen kein Problem mehr. Wie man in der Lösung sehr gut erkennen kann, sind bei der Verwendung von vier statt drei Satelliten keine größeren Änderungen im Programmtext nötig. Dieses liegt an der konsequenten Verwendung von Vektoren im Hauptprogramm und an den für die Berechnung mit beliebig großen Vektoren sorgfältig angepassten Funktionen. Nur bei den Bildschirmausgaben in den ersten beiden Schritten wurde bewusst auf die Ausgabe der kompletten Vektoren verzichtet, wie in den didaktischen Anmerkungen im Abschnitt 4.4.2 auf Seite 30 erklärt wird.

Ob das Gleichungssystem mit dem Empfängeruhrfehler `Delta_tE` oder mit dem Fehler der Pseudoentfernung `Delta_dS` formuliert wird, ist für den verwendeten Datensatz der Satellitendaten unerheblich. Bei anderen getesteten Satellitendaten gab

es in der Vergangenheit jedoch signifikante Unterschiede, wobei mit `Delta_dS` immer das genauere Ergebnis erreicht wurde. Deswegen sollte im Gleichungssystem besser der Fehler der Pseudoentfernungen `Delta_dS` verwendet werden.

4.5.3. Die Form der Erde

Bei dieser Modellverbesserung sollen sich die Schülerinnen und Schüler über folgende Begriffe im Internet informieren (vergleiche das Arbeitsblatt zu den Modellverbesserungen im Anhang auf Seite 100):

- Referenzellipsoid
- WGS 84
- geodätische Koordinaten
- ECEF-System

Dabei stößt man z. B. auf der deutschen Wikipedia-Seite unter dem Stichwort *Referenzellipsoid* auf die Gleichungen (24) von der Seite 19. Diese Gleichungen müssen für `fsolve` zu den Gleichungen (28) auf Seite 20 umgeformt werden. Zuerst werden die geographischen Koordinaten wie im vierten Modellschritt berechnet. Sie werden später als Startwerte für `fsolve` benötigt werden:

```
%   Berechnen der geographischen Koordinaten
h      = berechne_h(xE,yE,zE);
phi    = berechne_phi(xE,yE,zE);
lambda = berechne_lambda(xE,yE,zE);
```

Dann wird die numerischen Exzentrizität ε aus den beiden Halbachsen des Referenzellipsoiden WGS 84 errechnet:

```
%   Parameter des Referenzellipsoiden WGS 84
a = 6378137.0;
b = 6356752.3142;
epsilon = sqrt(a^2-b^2)/a;
```

Danach wird das Gleichungssystem aus den Funktionen (29) auf Seite 20 aufgestellt:

```
%   Aufstellen der Gleichungen
Gleichungen = @(h,phi,lambda) ...
   [(a/sqrt(1-epsilon^2*(sin(phi)^2))+h)*cos(phi)*cos(lambda)-xE
    (a/sqrt(1-epsilon^2*(sin(phi)^2))+h)*cos(phi)*sin(lambda)-yE
    (a*(1-epsilon^2)/sqrt(1-epsilon^2*(sin(phi)^2))+h)*sin(phi)-zE];
```

Anschließend müssen Funktion und Startpunkt für `fsolve` definiert werden. Als Startwerte für das Iterationsverfahren sollten die alten Lösungen aus vierten Modellschritt gewählt werden. Hier kann nicht einfach der Nullvektor `[0 0 0]` verwendet werden. da `fsolve` sonst nicht die richtige Lösung für `h`, `phi` und `lambda` finden wird. Die Winkel `phi` und `lambda` müssen dabei in das Bogenmaß umgerechnet werden, da die Winkelfunktionen in den Gleichungen die Winkel im Bogenmaß voraussetzen.

```
%   Definiere Funktion und Startpunkt für fsolve
Funktion = @(X) Gleichungen(X(1),X(2),X(3));
Startpunkt = [h phi*pi/180 lambda*pi/180];
```

Die Lösung kann einfach mit `fsolve` berechnet werden:

```
%   Berechnen einer Lösung in der Nähe des Startpunktes
X = fsolve(Funktion,Startpunkt);
```

Schließlich muss die berechnete Lösung noch in die Variablen `h`, `phi` und `lambda` abgespeichert werden:

```
%   Speichern der berechneten Lösung
h      = X(1);
phi    = X(2)*180/pi;
lambda = X(3)*180/pi;
```

Für die dritte Modellverbesserung sind somit folgende Korrekturen am Hauptprogramm notwendig. Das Löschen der nicht mehr benötigten Variablen wird dabei von den Schülerinnen und Schülern nicht gefordert:

```
%% Schritt 4 | Umrechnen in geographische Höhe, Breite und Länge
display('Schritt 4 | Umrechnen in geographische Höhe, Breite und Länge')

%   Aufstellen der Umrechnungsformeln
berechne_h      = @(xE,yE,zE) sqrt(xE^2+yE^2+zE^2)-6378137.0;
berechne_phi    = @(xE,yE,zE) atan(zE/sqrt(xE^2+yE^2))*180/pi;
berechne_lambda = @(xE,yE,zE) atan2(yE,xE)*180/pi;

%   Berechnen der geographischen Koordinaten
h      = berechne_h(xE,yE,zE);
phi    = berechne_phi(xE,yE,zE);
lambda = berechne_lambda(xE,yE,zE);

%   Parameter des Referenzellipsoiden WGS 84
a = 6378137.0;
b = 6356752.3142;
epsilon = sqrt(a^2-b^2)/a;

%   Aufstellen der Gleichungen
Gleichungen = @(h,phi,lambda) ...
    [(a/sqrt(1-epsilon^2*(sin(phi)^2))+h)*cos(phi)*cos(lambda)-xE
     (a/sqrt(1-epsilon^2*(sin(phi)^2))+h)*cos(phi)*sin(lambda)-yE
     (a*(1-epsilon^2)/sqrt(1-epsilon^2*(sin(phi)^2))+h)*sin(phi)-zE];

%   Definiere Funktion und Startpunkt für fsolve
Funktion = @(X) Gleichungen(X(1),X(2),X(3));
Startpunkt = [h phi*pi/180 lambda*pi/180];

%   Berechnen einer Lösung in der Nähe des Startpunktes
X = fsolve(Funktion,Startpunkt);

%   Speichern der berechneten Lösung
h      = X(1);
phi    = X(2)*180/pi;
lambda = X(3)*180/pi;

%   Anzeigen der Ergebnisse
format shortG
display(h,'Geographische Höhe des Empfängers in Meter [h]')
display([phi,lambda], ...
    'Geographische Breite und Länge des Empfängers in Grad [phi lambda]')
clearvars Funktion Startpunkt X
```

Didaktisch-methodische Anmerkungen: Würde man die Gleichungen (24) und (25) auf Seite 19 von den Schülerinnen und Schülern herleiten lassen, wäre diese Modell-

verbesserung unverhältnismäßig schwierig. Außerdem ist eine vollständige Herleitung unmöglich im Rahmen eines Modellierungstages zu schaffen. Deshalb sollen die Schülerinnen und Schüler nur auf Internetsuche gehen mit der Idee, `fsolve` für die Lösung der Gleichungen zu verwenden. Falls kein Internet für die Suche nach den Schlüsselbegriffen zur Verfügung steht oder die richtigen Umrechnungsformeln nicht gefunden werden, kann eine Hilfekarte angefordert werden, auf der die Gleichungen (24) und (25) sowie die Parameter des Referenzellipsoiden WGS 84 angegeben sind (vergleiche die Hilfekarten zu den Modellverbesserungen im Anhang auf Seite 103).

Wesentlich für die Verwendung von `fsolve` ist hier ein möglichst guter Startpunkt. Deshalb sollten hier die „alten“ Lösungen für `h`, `phi` und `lambda` verwendet werden. Die Winkel müssen dafür in das Bogenmaß zurückgerechnet werden, da die Gleichungen mit den Winkelfunktionen `sin` und `cos` formuliert sind. Alternativ könnte man bei den Gleichungen auch die für das Gradmaß ausgelegten Winkelfunktionen `sind` und `cosd` verwenden, um so die Umrechnung der Startwerte und der Lösungen einzusparen.

4.5.4. Verwendung aller Satelliten

In dieser Modellverbesserung sollen die Schülerinnen und Schüler recherchieren, wie mit überbestimmten Gleichungssystemen umgegangen werden kann. Durch eine Internetsuche sollen sie auf das Thema *Ausgleichsrechnung* und die *Methode der kleinsten Quadrate* aufmerksam werden. Dazu ist auch die Warnung hilfreich, die `fsolve` ausgibt, wenn das Gleichungssystem mehr Gleichungen als Unbekannte hat:

```
Warning: Trust-region-dogleg algorithm of FSOLVE cannot handle
non-square systems; using Levenberg-Marquardt algorithm instead.
```

Durch eine Internetsuche nach den Begriffen „Matlab“ und „Levenberg-Marquardt algorithm“ oder durch eine MATLAB-interne Suche nach „Levenberg-Marquardt algorithm“ wird man sehr schnell auf das MATLAB-Programm `lsqnonlin` geführt, welches eine Ausgleichsrechnung nach der Methode der kleinsten Quadrate für nichtlineare Gleichungssysteme durchführt. Da jetzt alle empfangenen Satelliten verwendet werden sollen, muss zuerst die Auswahl der Satelliten im ersten Abschnitt angepasst werden. Für acht empfangene Satelliten lautet die Änderung wie folgt:

```
%   Auswählen der Satelliten (für acht Satelliten)
Satellitenauswahl = [1 2 3 4 5 6 7 8];
```

Eine beliebige Anzahl empfangener Satelliten lässt sich auch allgemeiner darstellen, da in der Variable `Satelliten` alle empfangenen Satelliten als Vektor abgespeichert sind:

```
%   Auswählen der Satelliten (für beliebig viele Satelliten)
Satellitenauswahl = Satelliten;
```

Danach muss das Gleichungssystem so angepasst werden, dass alle Satelliten verwendet werden. Das kann speziell für beispielsweise acht Satelliten geschehen:

```
%   Aufstellen des Gleichungssystems (für acht Satelliten)
Gleichungssystem = @(xE,yE,zE) ...
    [sqrt((xS(1)-xE)^2+(yS(1)-yE)^2+(zS(1)-zE)^2)-dS(1)
     sqrt((xS(2)-xE)^2+(yS(2)-yE)^2+(zS(2)-zE)^2)-dS(2)
     sqrt((xS(3)-xE)^2+(yS(3)-yE)^2+(zS(3)-zE)^2)-dS(3)
     sqrt((xS(4)-xE)^2+(yS(4)-yE)^2+(zS(4)-zE)^2)-dS(4)
     sqrt((xS(5)-xE)^2+(yS(5)-yE)^2+(zS(5)-zE)^2)-dS(5)
     sqrt((xS(6)-xE)^2+(yS(6)-yE)^2+(zS(6)-zE)^2)-dS(6)
     sqrt((xS(7)-xE)^2+(yS(7)-yE)^2+(zS(7)-zE)^2)-dS(7)
     sqrt((xS(8)-xE)^2+(yS(8)-yE)^2+(zS(8)-zE)^2)-dS(8)];
```

Durch Verwendung der Vektoren `xS`, `yS`, `zS` und `dS` kann das Gleichungssystem auch für eine beliebige Anzahl von Satelliten dargestellt werden:

```
%   Aufstellen des Gleichungssystems (für beliebig viele Satelliten)
Gleichungssystem = @(xE,yE,zE) ...
    sqrt((xS-xE).^2+(yS-yE).^2+(zS-zE).^2)-dS;
```

Dabei bewirkt der Punkt vor dem Potenzoperator eine komponentenweise Berechnung der Quadrate. Schließlich ist der Aufruf von `fsolve` durch einen Aufruf von `lsqnonlin` zu ersetzen:

```
%   Berechnen einer Lösung in der Nähe des Startpunktes
X = lsqnonlin(Funktion,Startpunkt);
```

Letztlich sollten noch die Bildschirmausgaben für eine beliebige Zahl verwendeter Satelliten angepasst werden, obwohl das von den Schülerinnen und Schülern nicht unbedingt erwartet wird. Für die vierte Modellverbesserung sind insgesamt folgende Korrekturen am Hauptprogramm notwendig:

```
%% Einlesen der Satellitendaten
display('Einlesen der Satellitendaten')

%   Einlesen der Satellitendaten falls nicht vorhanden
if ~exist('Satellitendaten','var')
    Dateien = {'114255.obs','114255.nav'};
    Satellitendaten = einlesen_Satellitendaten(Dateien);
end

%   Auswählen eines Datensatzes
Datensatznummer = 1;
Datensatz = Satellitendaten(Datensatznummer);

%   Satelliten, deren Nachrichten empfangen wurden
Satelliten = Datensatz.Satelliten;

%   Auswählen der Satelliten
Satellitenauswahl = Satelliten;

%   Auslesen der ausgewählten Satellitendaten
tE = Datensatz.Empfangszeiten(Satellitenauswahl);
tS = Datensatz.Sendezeiten(Satellitenauswahl);
Ephemeriden = Datensatz.Ephemeriden(:,Satellitenauswahl);

%   Anzeigen der Ergebnisse
format shortG
display(Datensatz,sprintf('Datensatz Nummer %d',Datensatznummer))
display(Satelliten, ...
    'Satelliten, deren Nachrichten empfangen wurden [Satelliten]')
display(Satellitenauswahl, ...
    'Nummern der ausgewählten Satelliten [Satellitenauswahl]')
format bank
display(['tE.' tS.'], ['Empfangszeiten und Sendezeiten der ', ...
    'Nachrichten in Sekunden seit Wochenbeginn [tE tS]'])
display(['Die Positionsdaten der Satelliten werden nicht angezeigt ', ...
    '[Ephemeriden]'])
clearvars Dateien Datensatznummer

%% Schritt 1  |  Berechnen der Satellitenkoordinaten
display('Schritt 1  |  Berechnen der Satellitenkoordinaten')

%   Berechnen der Satellitenkoordinaten
```

```
[xS,yS,zS] = berechne_Satellitenkoordinaten(tS,Ephemeriden);

%   Anzeigen der Ergebnisse
format bank
display([xS.' yS.' zS.'], ...
    'Koordinaten der Satelliten in Meter [xS yS zS]')

%% Schritt 2  |  Berechnen der Pseudoentfernungen
display('Schritt 2  |  Berechnen der Pseudoentfernungen')

%   Aufstellen der Berechnungsformel
berechne_Pseudoentfernungen = @(tS,tE) (tE-tS)*299792458;

%   Berechnen der Pseudoentfernungen
dS = berechne_Pseudoentfernungen(tS,tE);

%   Anzeigen der Ergebnisse
format bank
display(dS.','Pseudoentfernungen zu den Satelliten in Meter [dS]')

%% Schritt 3  |  Berechnen der Empfängerkoordinaten
display('Schritt 3  |  Berechnen der Empfängerkoordinaten')

%   Aufstellen des Gleichungssystems
Gleichungssystem = @(xE,yE,zE) ...
    sqrt((xS-xE).^2+(yS-yE).^2+(zS-zE).^2)-dS;

%   Definiere Funktion und Startpunkt für fsolve
Funktion = @(X) Gleichungssystem(X(1),X(2),X(3));
Startpunkt = [0 0 0];

%   Berechnen einer Lösung in der Nähe des Startpunktes
X = lsqnonlin(Funktion,Startpunkt);

%   Speichern der berechneten Lösung
xE = X(1);
yE = X(2);
zE = X(3);

%   Anzeigen der Ergebnisse
format bank
display([xE yE zE],'Koordinaten des Empfängers in Meter [xE yE zE]')
clearvars Funktion Startpunkt X
```

Didaktisch-methodische Anmerkungen: Bei der vierten Modellverbesserung geht es hauptsächlich um die Erkenntnis, wie mit überbestimmten Gleichungssystemen umgegangen werden sollte. Die Verwendung von fsolve führt in so einem Fall zu einer Warnung, die Anlass gibt, weitere Nachforschungen anzustellen. Natürlich könnte man auch die Warnung ignorieren und fsolve auch in diesem Fall verwenden, denn fsolve erkennt, wenn eine exakte Lösung nicht möglich ist und wechselt den Algorithmus, um für das nicht-lösbare Gleichungssystem trotzdem eine „Lösung“ zu finden. Hier ergibt sich jedoch auch die Chance zu lernen, was man in so einem Fall unter einer „Lösung“ überhaupt verstehen kann. In diesem Sinne geht es bei dieser Modellverbesserung hauptsächlich darum, die Ausgleichsrechnung und die Methode der kleinsten Quadrate kennenzulernen. Tatsächlich verwendet auch fsolve zur Nullstellensuche nicht das Newton-Verfahren, sondern entweder ein Trust-Region-Verfahren oder den Levenberg-Marquardt Algorithmus. Damit berechnet fsolve in Wirklichkeit keine Nullstelle, son-

dern genau wie lsqnonlin eine Minimalstelle nach der Methode der kleinsten Quadrate. Jedoch sieht fsolve im Gegensatz zu lsqnonlin eine gefundene Minimalstelle nur dann als Lösung an, wenn das zugehörige Minimum den Wert Null hat.

4.5.5. Die Kombination aller vier Modellverbesserungen

Die einzelnen Modellverbesserungen lassen sich problemlos miteinander kombinieren. Sie sind weitestgehend unabhängig voneinander, nur die zweite und vierte Modellverbesserung finden im selben Gleichungssystem statt. Durch die Kombination der vier Modellverbesserungen findet man jetzt die korrekte Position des Empfängers in Aachen mit einer Abweichung von weniger als 20 Metern. Das Hauptprogramm mit allen vier Modellverbesserungen lautet:

```
%% Einlesen der Satellitendaten
display('Einlesen der Satellitendaten')

%  Einlesen der Satellitendaten falls nicht vorhanden
if ~exist('Satellitendaten','var')
    Dateien = {'114255.obs','114255.nav'};
    Satellitendaten = einlesen_Satellitendaten(Dateien);
end

%  Auswählen eines Datensatzes
Datensatznummer = 1;
Datensatz = Satellitendaten(Datensatznummer);

%  Satelliten, deren Nachrichten empfangen wurden
Satelliten = Datensatz.Satelliten;

%  Auswählen der Satelliten
Satellitenauswahl = Satelliten;

%  Auslesen der ausgewählten Satellitendaten
tE = Datensatz.Empfangszeiten(Satellitenauswahl);
tS = Datensatz.Sendezeiten(Satellitenauswahl);
Ephemeriden = Datensatz.Ephemeriden(:,Satellitenauswahl);

%  Anzeigen der Ergebnisse
format shortG
display(Datensatz,sprintf('Datensatz Nummer %d',Datensatznummer))
display(Satelliten, ...
    'Satelliten, deren Nachrichten empfangen wurden [Satelliten]')
display(Satellitenauswahl, ...
    'Nummern der ausgewählten Satelliten [Satellitenauswahl]')
format bank
display([tE.' tS.'], ['Empfangszeiten und Sendezeiten der ', ...
    'Nachrichten in Sekunden seit Wochenbeginn [tE tS]'])
display(['Die Positionsdaten der Satelliten werden nicht angezeigt ', ...
    '[Ephemeriden]'])
clearvars Dateien Datensatznummer

%% Schritt 1  |  Berechnen der Satellitenkoordinaten
display('Schritt 1  |  Berechnen der Satellitenkoordinaten')

%  Berechnung der Satellitenuhrfehler
Delta_tS = berechne_Satellitenuhrfehler(tS,Ephemeriden);
Delta_tS = berechne_Satellitenuhrfehler(tS-Delta_tS,Ephemeriden);

%  Berechnen der Satellitenkoordinaten
[xS,yS,zS] = berechne_Satellitenkoordinaten(tS-Delta_tS,Ephemeriden);
```

```
%   Anzeigen der Ergebnisse
format shortG
display(Delta_tS, 'Fehler der Satellitenuhren in Sekunden [Delta_tS]')
format bank
display([xS.' yS.' zS.'], ...
    'Koordinaten der Satelliten in Meter [xS yS zS]')

%% Schritt 2  |  Berechnen der Pseudoentfernungen
display('Schritt 2  |  Berechnen der Pseudoentfernungen')

%   Aufstellen der Berechnungsformel
berechne_Pseudoentfernungen = @(tS,tE) (tE-tS)*299792458;

%   Berechnen der Pseudoentfernungen
dS = berechne_Pseudoentfernungen(tS-Delta_tS,tE);

%   Anzeigen der Ergebnisse
format bank
display(dS.','Pseudoentfernungen zu den Satelliten in Meter [dS]')

%% Schritt 3  |  Berechnen der Empfängerkoordinaten
display('Schritt 3  |  Berechnen der Empfängerkoordinaten')

%   Aufstellen des Gleichungssystems
Gleichungssystem = @(xE,yE,zE,Delta_dS) ...
    sqrt((xS-xE).^2+(yS-yE).^2+(zS-zE).^2)-(dS-Delta_dS);

%   Definiere Funktion und Startpunkt für fsolve
Funktion = @(X) Gleichungssystem(X(1),X(2),X(3),X(4));
Startpunkt = [0 0 0 0];

%   Berechnen einer Lösung in der Nähe des Startpunktes
X = lsqnonlin(Funktion,Startpunkt);

%   Speichern der berechneten Lösung
xE = X(1);
yE = X(2);
zE = X(3);
Delta_tE = X(4)/299792458;

%   Anzeigen der Ergebnisse
format shortG
display(Delta_tE,'Fehler der Empfängeruhr in Sekunden [Delta_tE]')
format bank
display([xE yE zE],'Koordinaten des Empfängers in Meter [xE yE zE]')
clearvars Funktion Startpunkt X

%% Schritt 4  |  Umrechnen in geographische Höhe, Breite und Länge
display('Schritt 4  |  Umrechnen in geographische Höhe, Breite und Länge')

%   Aufstellen der Umrechnungsformeln
berechne_h      = @(xE,yE,zE) sqrt(xE^2+yE^2+zE^2)-6378137.0;
berechne_phi    = @(xE,yE,zE) atan(zE/sqrt(xE^2+yE^2))*180/pi;
berechne_lambda = @(xE,yE,zE) atan2(yE,xE)*180/pi;

%   Berechnen der geographischen Koordinaten
h      = berechne_h(xE,yE,zE);
phi    = berechne_phi(xE,yE,zE);
lambda = berechne_lambda(xE,yE,zE);
```

```
%   Parameter des Referenzellipsoiden WGS 84
a = 6378137.0;
b = 6356752.3142;
epsilon = sqrt(a^2-b^2)/a;

%   Aufstellen der Gleichungen
Gleichungen = @(h,phi,lambda) ...
    [(a/sqrt(1-epsilon^2*(sin(phi)^2))+h)*cos(phi)*cos(lambda)-xE
     (a/sqrt(1-epsilon^2*(sin(phi)^2))+h)*cos(phi)*sin(lambda)-yE
     (a*(1-epsilon^2)/sqrt(1-epsilon^2*(sin(phi)^2))+h)*sin(phi)-zE];

%   Definiere Funktion und Startpunkt für fsolve
Funktion = @(X) Gleichungen(X(1),X(2),X(3));
Startpunkt = [h phi*pi/180 lambda*pi/180];

%   Berechnen einer Lösung in der Nähe des Startpunktes
X = fsolve(Funktion,Startpunkt);

%   Speichern der berechneten Lösung
h      = X(1);
phi    = X(2)*180/pi;
lambda = X(3)*180/pi;

%   Anzeigen der Ergebnisse
format shortG
display(h,'Geographische Höhe des Empfängers in Meter [h]')
display([phi,lambda], ...
    'Geographische Breite und Länge des Empfängers in Grad [phi lambda]')
clearvars Funktion Startpunkt X
```

Didaktisch-methodische Anmerkungen: Es ist sehr reizvoll, die Folge der Verbesserungen zu betrachten, wenn sie nacheinander durchgeführt werden. Dabei rutscht die berechnete Empfängerposition immer näher an Aachen heran. Die geringsten Auswirkungen scheint dabei die vierte Modellverbesserung zu haben. Jedoch kann man nach den ersten drei Verbesserungen durch Änderung der Satellitenauswahl genau erkennen, dass die berechnete Empfängerposition von den ausgewählten Satelliten abhängig ist. Dabei sind deutliche Sprünge der Empfängerposition feststellbar, und die einzelnen Lösungen haben Ungenauigkeiten von etwa 50 Meter. Erst durch die vierte Modellverbesserung erhält man eine eindeutige Lösung mit einer Ungenauigkeit von unter 20 Metern.

Die Fehler bei der Höhenbestimmung des Empfängers sind allerdings deutlich größer als die durchschnittlichen Positionsfehler. Diese sogenannten Geoidundulationen hängen mit der unregelmäßigen Oberfläche der Erde zusammen und können bei einem Bezugsellipsoiden bis zu 100 Meter betragen.

4.5.6. Gewichtung der Satelliten

Die fünfte Modellverbesserung ist als Zusatzaufgabe gedacht und sollte erst nach den ersten vier Modellverbesserungen bearbeitet werden. Tatsächlich stellt es eine Erweiterung der vierten Modellverbesserung dar und kann deshalb nicht unabhängig von ihr gelöst werden.

Für die fünfte Modellverbesserung müssen die Signalstärken als Gewichtsfaktoren bei der Summenbildung berücksichtigt werden. Dazu müssen im ersten Abschnitt die Signalstärken aus dem Datensatz ausgelesen und in den Vektor G als Gewichtsfaktoren abgespeichert werden:

```
%   Auslesen der ausgewählten Satellitendaten
tE = Datensatz.Empfangszeiten(Satellitenauswahl);
tS = Datensatz.Sendezeiten(Satellitenauswahl);
G = Datensatz.Signalstaerken(Satellitenauswahl);
Ephemeriden = Datensatz.Ephemeriden(:,Satellitenauswahl);
```

Danach ist nur noch das Gleichungssystem bei der Berechnung der Empfängerkoordinaten anzupassen. Nach Gleichung (34) auf Seite 23 müssen die Funktionen aus Gleichung (32) auf Seite 22 nur mit den Quadratwurzeln der Gewichtsfaktoren multipliziert werden. Diese Multiplikationen können durch die Verwendung des Vektors G kompakt dargestellt werden:

```
%   Aufstellen des Gleichungssystems
Gleichungssystem = @(xE,yE,zE,Delta_dS) ...
    sqrt(G).*(sqrt((xS-xE).^2+(yS-yE).^2+(zS-zE).^2)-(dS-Delta_dS));
```

Der Punkt vor dem Multiplikationsoperator bewirkt eine komponentenweise Berechnung der Produkte. Insgesamt sind folgende Korrekturen am Hauptprogramm notwendig. Dabei wird die Bildschirmausgabe der Signalstärken nicht von den Schülerinnen und Schülern gefordert. Jedoch ist die Formatänderung der Ausgabe im vierten Schritt von `shortG` auf `longG` notwendig, da die Veränderungen in den geographischen Koordinaten sonst nicht wahrgenommen werden können.

```
%% Einlesen der Satellitendaten
display('Einlesen der Satellitendaten')

%   Einlesen der Satellitendaten falls nicht vorhanden
if ~exist('Satellitendaten','var')
    Dateien = {'114255.obs','114255.nav'};
    Satellitendaten = einlesen_Satellitendaten(Dateien);
end

%   Auswählen eines Datensatzes
Datensatznummer = 1;
Datensatz = Satellitendaten(Datensatznummer);

%   Satelliten, deren Nachrichten empfangen wurden
Satelliten = Datensatz.Satelliten;

%   Auswählen der Satelliten
Satellitenauswahl = Satelliten;

%   Auslesen der ausgewählten Satellitendaten
tE = Datensatz.Empfangszeiten(Satellitenauswahl);
tS = Datensatz.Sendezeiten(Satellitenauswahl);
G = Datensatz.Signalstaerken(Satellitenauswahl);
Ephemeriden = Datensatz.Ephemeriden(:,Satellitenauswahl);

%   Anzeigen der Ergebnisse
format shortG
display(Datensatz,sprintf('Datensatz Nummer %d',Datensatznummer))
display(Satelliten, ...
    'Satelliten, deren Nachrichten empfangen wurden [Satelliten]')
display(Satellitenauswahl, ...
    'Nummern der ausgewählten Satelliten [Satellitenauswahl]')
format bank
display([tE.' tS.'], ['Empfangszeiten und Sendezeiten der ', ...
    'Nachrichten in Sekunden seit Wochenbeginn [tE tS]'])
format shortG
```

```
display(G,'Signalstärken der ausgewählten Nachrichten in dBHz [G]')
display(['Die Positionsdaten der Satelliten werden nicht angezeigt ', ...
    '[Ephemeriden]'])
clearvars Dateien Datensatznummer

%% Schritt 1 | Berechnen der Satellitenkoordinaten
display('Schritt 1 | Berechnen der Satellitenkoordinaten')

% Berechnung der Satellitenuhrfehler
Delta_tS = berechne_Satellitenuhrfehler(tS,Ephemeriden);
Delta_tS = berechne_Satellitenuhrfehler(tS-Delta_tS,Ephemeriden);

% Berechnen der Satellitenkoordinaten
[xS,yS,zS] = berechne_Satellitenkoordinaten(tS-Delta_tS,Ephemeriden);

% Anzeigen der Ergebnisse
format shortG
display(Delta_tS, 'Fehler der Satellitenuhren in Sekunden [Delta_tS]')
% Anzeigen der Ergebnisse
format bank
display([xS.' yS.' zS.'], ...
    'Koordinaten der Satelliten in Meter [xS yS zS]')

%% Schritt 2 | Berechnen der Pseudoentfernungen
display('Schritt 2 | Berechnen der Pseudoentfernungen')

% Aufstellen der Berechnungsformel
berechne_Pseudoentfernungen = @(tS,tE) (tE-tS)*299792458;

% Berechnen der Pseudoentfernungen
dS = berechne_Pseudoentfernungen(tS-Delta_tS,tE);

% Anzeigen der Ergebnisse
format bank
display(dS.','Pseudoentfernungen zu den Satelliten in Meter [dS]')

%% Schritt 3 | Berechnen der Empfängerkoordinaten
display('Schritt 3 | Berechnen der Empfängerkoordinaten')

% Aufstellen des Gleichungssystems
Gleichungssystem = @(xE,yE,zE,Delta_dS) ...
    sqrt(G).*(sqrt((xS-xE).^2+(yS-yE).^2+(zS-zE).^2)-(dS-Delta_dS));

% Definiere Funktion und Startpunkt für fsolve
Funktion = @(X) Gleichungssystem(X(1),X(2),X(3),X(4));
Startpunkt = [0 0 0 0];

% Berechnen einer Lösung in der Nähe des Startpunktes
X = lsqnonlin(Funktion,Startpunkt);

% Speichern der berechneten Lösung
xE = X(1);
yE = X(2);
zE = X(3);
Delta_tE = X(4)/299792458;

% Anzeigen der Ergebnisse
format shortG
display(Delta_tE,'Fehler der Empfängeruhr in Sekunden [Delta_tE]')
format bank
display([xE yE zE],'Koordinaten des Empfängers in Meter [xE yE zE]')
```

```
clearvars Funktion Startpunkt X

%% Schritt 4 | Umrechnen in geographische Höhe, Breite und Länge
display('Schritt 4 | Umrechnen in geographische Höhe, Breite und Länge')

%   Aufstellen der Umrechnungsformeln
berechne_h      = @(xE,yE,zE) sqrt(xE^2+yE^2+zE^2)-6378137.0;
berechne_phi    = @(xE,yE,zE) atan(zE/sqrt(xE^2+yE^2))*180/pi;
berechne_lambda = @(xE,yE,zE) atan2(yE,xE)*180/pi;

%   Berechnen der geographischen Koordinaten
h      = berechne_h(xE,yE,zE);
phi    = berechne_phi(xE,yE,zE);
lambda = berechne_lambda(xE,yE,zE);

%   Parameter des Referenzellipsoiden WGS 84
a = 6378137.0;
b = 6356752.3142;
epsilon = sqrt(a^2-b^2)/a;

%   Aufstellen der Gleichungen
Gleichungen = @(h,phi,lambda) ...
   [(a/sqrt(1-epsilon^2*(sin(phi)^2))+h)*cos(phi)*cos(lambda)-xE
    (a/sqrt(1-epsilon^2*(sin(phi)^2))+h)*cos(phi)*sin(lambda)-yE
    (a*(1-epsilon^2)/sqrt(1-epsilon^2*(sin(phi)^2))+h)*sin(phi)-zE];

%   Definiere Funktion und Startpunkt für fsolve
Funktion = @(X) Gleichungen(X(1),X(2),X(3));
Startpunkt = [h phi*pi/180 lambda*pi/180];

%   Berechnen einer Lösung in der Nähe des Startpunktes
X = fsolve(Funktion,Startpunkt);

%   Speichern der berechneten Lösung
h      = X(1);
phi    = X(2)*180/pi;
lambda = X(3)*180/pi;

%   Anzeigen der Ergebnisse
format longG
display(h,'Geographische Höhe des Empfängers in Meter [h]')
display([phi,lambda], ...
    'Geographische Breite und Länge des Empfängers in Grad [phi lambda]')
clearvars Funktion Startpunkt X
```

Didaktisch-methodische Anmerkungen: Die fünfte Modellverbesserung kommt nur bei besonders leistungsstarken Schülerinnen und Schülern zum Einsatz. Sollte tatsächlich eine Gruppe die übrigen vier Modellverbesserungen alle selbstständig durchgeführt haben, kann sie sich in der verbleibenden Zeit noch mit der fünften Modellverbesserung auseinandersetzen. Die Schwierigkeit dieser Aufgabe liegt in der gedanklichen Hürde, die gewichtete Summe $g_1 \cdot f_1^2(x) + g_2 \cdot f_2^2(x) + \ \ldots \ + g_n \cdot f_n^2(x)$ auf eine einfache Summe $h_1^2(x) + h_2^2(x) + \ \ldots \ + h_n^2(x)$ zurückzuführen. Für diese Zusatzaufgabe gibt es keine Hilfekarten. Hier müssen Andeutungen der Betreuer genügen, dass eine Lösung mit den bisher verwendeten Mitteln und Methoden auch möglich ist.

4.6. Die Musterlösungen für die Betreuer

Für die Betreuer wurden für die Modellschritte und Modellverbesserungen kurze Musterlösungen erstellt, welche im Anhang aufgeführt sind. Die Lösung zu den Modellschritten ist auf den Seiten 98 und 99 abgedruckt, die Lösung zu den Modellverbesserungen findet sich auf den Seiten 104 bis 109. Neben den gedruckten Lösungen haben die Betreuer auch Zugriff auf die beiden MATLAB-Programme `gps_master.m` und `gps_solution.m`, in denen ebenfalls die Musterlösungen enthalten sind.

Das Progamm `gps_master.m` enthält die Lösungen der Modellschritte und ist im Anhang ab Seite 85 abgedruckt. Durch die Variable `Modellschritt` kann ausgewählt werden, an welcher Stelle im Hauptprogramm die Überprüfungsfunktion den Programmablauf abbrechen soll. Dadurch kann das Programm `gps_master.m` auch genutzt werden, um Änderungen an der Überprüfungsfunktion komfortabel testen zu können.

Das Programm `gps_solution.m` enthält die Lösungen aller Modellverbesserungen und ist im Anhang ab Seite 89 abgedruckt. Über die Variable `Modellverbesserung` kann eine beliebige Kombination der fünf Modellverbesserungen ausgewählt werden. Durch eine leere Liste werden alle Verbesserungen abgeschaltet, und jede Zahl von 1 bis 5 in dieser Liste schaltet die entsprechende Modellverbesserung ein. Mit der Liste `[1 2 3 4 5]` werden alle Verbesserungen aktiviert. Dabei sind die Modellverbesserungen beliebig miteinander kombinierbar, so dass alle $2^5 = 32$ möglichen Kombinationen durchgespielt werden können, obwohl die fünfte Modellverbesserung nur im Zusammenspiel mit der vierten als sinnvoll angesehen werden kann. So ist dieses Programm eine kompakte Übersicht über alle Modellverbesserungen und kann am Ende eines Modellierungstages dazu verwendet werden, die Effekte der einzelnen Verbesserungen komfortabel vorführen zu können. Außerdem ist es von Nutzen, wenn für andere Satellitendaten eine Vergleichsliste mit Zielkoordinaten erstellt werden muss (vgl. dazu das Arbeitsblatt zu den Modellverbesserungen im Anhang auf Seite 101).

5. Evaluation

Der Projekttag wurde in der Praxis mehrfach erprobt und durch die teilnehmenden Schülerinnen und Schüler anschließend evaluiert. Der Evaluationsbogen ist im Anhang auf den Seiten 110 und 111 abgedruckt. Diese Auswertung bezieht sich auf folgende vier Projekttage:

- Gesamtschule Langerwehe am 16. Januar 2015
- Pius Gymnasium am 28. Mai 2015
- Anne-Frank Gymnasium am 5. Juni 2015
- Gesamtschule Aachen Brand am 19. Juni 2015

Es wurden insgesamt 82 Schülerinnen und Schüler befragt. Die Ergebnisse der Evaluation sind in Tabelle 3 dargestellt.

	Trifft gar nicht zu (−−)	**Trifft eher nicht zu** (−)	**Trifft zum Teil zu** (+)	**Trifft voll zu** (++)
Durch den Workshop habe ich mathematisches Modellieren besser begriffen.	1,2 %	9,8 %	40,2 %	48,8 %
Der Vortrag über Modellierung war hilfreich.	0 %	3,7 %	64,6 %	31,7 %
Der einführende Kurzfilm war hilfreich.	0 %	9,8 %	32,9 %	57,3 %
Die Einführung in MATLAB war hilfreich.	4,9 %	19,5 %	51,2 %	23,2 %
Der Umgang mit MATLAB fiel mir schwer.	9,8 %	29,3 %	43,9 %	15,8 %
Die Aufgaben waren zu einfach.	43,9 %	46,3 %	9,8 %	0 %
Die Aufgaben waren zu schwierig.	3,6 %	29,3 %	61,0 %	6,1 %
Die Hilfekarten waren hilfreich.	1,2 %	17,1 %	50,0 %	29,3 %
Das Hilfesystem in MATLAB war hilfreich.	11,0 %	26,8 %	25,6 %	19,5 %

Tabelle 3: Ergebnisse der Evaluation des Projekttages

5.1. Auswertung der Evaluation

- Das Ziel, mathematisches Modellieren an einem praxisnahen Beispiel verständlicher zu machen wird durch den Projekttag fast immer erreicht. Nur 11 % der Schülerinnen und Schüler geben an, dass dieses Ziel bei ihnen nicht erreicht worden sei. 49 % hingegen sagen, dass dieses Ziel voll und ganz erreicht worden ist. Dazu trägt sicherlich auch der einführende Vortrag über den Modellierungskreislauf bei. Dieser wird gerade mal von 4 % der Schülerinnen und Schüler als wenig hilfreich angesehen. Dennoch können hier sicherlich noch Änderungen eingeplant werden, da die Mehrzahl der Schülerinnen und Schüler ihn nur teilweise als hilfreich empfand.

- Der einführende Kurzfilm hingegen hat sich auf jeden Fall bewährt und sollte auch weiterhin vorgeführt werden. Der größte Teil der Schülerinnen und Schüler fanden ihn besonders hilfreich und nur 10 % konnten sich nicht dafür begeistern.

- 60 % der Schülerinnen und Schüler hatten Schwierigkeiten beim Umgang mit MATLAB, und über 75 % fanden die Einführung in MATLAB wenig oder nur zum Teil hilfreich. Somit sollte die Einführung in MATLAB weiter überarbeitet und angepasst werden.

- Der Schwierigkeitsgrad der Aufgaben kann als hoch angesehen werden. 90 % der Schülerinnen und Schüler halten die Aufgaben nicht für zu einfach, 33 % halten sie nicht für zu schwierig. 61 % hielten einzelne Aufgaben für zu schwierig, und nur 6 % sehen alle Aufgaben als zu schwierig an. Der hohe Schwierigkeitsgrad der Aufgaben lässt sich sicherlich durch Verbesserungen bei den Hilfekarten noch etwas absenken. Denn über 68 % der Schülerinnen und Schüler sehen Verbesserungsmöglichkeiten bei den Hilfekarten, nur für 29 % waren sie allesamt hilfreich.

- Die Frage nach der Nützlichkeit der von der Überprüfungsfunktion ausgegebenen Hilfetexte wurde leider missverständlich formuliert, denn nur 83 % der Schülerinnen und Schüler haben hier eine Antwort gegeben. Außerdem fällt die Verteilung der Antworten sehr gleichmäßig aus, obwohl bei Befragungen während einiger Projekttage sich fast alle Schülerinnen und Schüler positiv zu der Überprüfungsfunktion geäußert haben. Hier ist bestimmt nicht klar gewesen, was in dieser Frage mit dem „Hilfesystem in MATLAB“ gemeint ist, da es neben der Hilfe durch die Überprüfungsfunktion auch noch eine MATLAB-interne Hilfeseite gibt. Deswegen werden die Antworten hier nicht weiter ausgewertet.

6. Fazit und Ausblick

Die Ergebnisse der Evaluation zeigen, dass der Projekttag gut funktioniert und bei vielen Schülerinnen und Schülern das Verständnis für das mathematische Modellieren erweitern kann. Verbesserungsmöglichkeiten gibt es jedoch mindestens an folgenden Stellen:

- Der Modellierungsvortrag sollte überarbeitet werden
- Die Einführung in MATLAB sollte besser für diesen Tag angepasst werden
- Die Hilfekarten sollten überarbeitet oder erweitert werden

Desweiteren ist im Fragebogen der letzte Eintrag über das Hilfesystem in MATLAB zu verändern. Es sollte besser nach der Nützlichkeit der Überprüfungsfunktion gefragt werden, um Missverständnissen vorzubeugen. Als zukünftige Erweiterungen des Projekttages sind denkbar:

- Eine eigene Implementierung des Newton-Verfahrens für besonders interessierte Schülerinnen und Schüler
- Die Entwicklung und Implementierung von Korrekturmodellen für die Troposphäre und die Ionosphäre. Eine beispielhafte Implementierung einer Troposhärenkorrektur nach K. Borre befindet sich im Projektverzeichnis (SVN) im Ordner `gps/temp/future`.
- Anstelle einer Ausgleichsrechnung mit allen Satelliten können in der vierten Modellverbesserung auch die vier Satelliten mit der kleinsten DOP (Dilution of Precision) ausgewählt werden (vgl. [11], Zogg, 2009, S. 90 ff.).

Trotz des hohen Anforderungsprofils verliefen die Projekttage insgesamt sehr zufriedenstellend und stärkten die Selbstständigkeit und weitere Kompetenzen der Schülerinnen und Schüler, möglicherweise nicht zuletzt durch den außerschulischen Lernort, der hohe Arbeitsplatznähe bzw. Studienplatznähe verkörpert.

7. Literaturverzeichnis

[1] V. Birkenbihl. *Stichwort Schule: Trotz Schule lernen!* Heidelberg: mvg, 2004.

[2] K. Borre. „The GPS Easy Suite – Matlab code for the GPS newcomer“. In: *GPS solutions* 7.1 (2003), S. 47–51. DOI: 10.1007/s10291-003-0049-3.

[3] Global Positioning System Directorate. *Interface Specification IS-GPS-200. Revision H.* Navstar GPS Space Segment / Navigation User Interfaces. 2013. URL: http://www.gps.gov/technical/icwg/IS-GPS-200H.pdf (besucht am 12. 08. 2015).

[4] W. Gurtner und L. Estey. *RINEX. The Receiver Independent Exchange Format.* Version 3.01. 2009. URL: ftp://ftp.igs.org/pub/data/format/rinex301.pdf (besucht am 12. 08. 2015).

[5] D. Leiss und N. Tropper. *Umgang mit Heterogenität im Mathematikunterricht. Adaptives Lehrerhandeln beim Modellieren.* Mathematik im Fokus. Berlin, Heidelberg: Springer, 2014.

[6] T. Leuders, Hrsg. *Mathematik-Didaktik.* Berlin: Cornelsen Scriptor, 2011. ISBN: 978-3-589-21695-6.

[7] Ministerium für Schule und Weiterbildung des Landes Nordrhein-Westfalen. *Kernlehrplan für das Gymnasium – Sekundarstufe I. Mathematik.* 2007. URL: http://www.schulentwicklung.nrw.de/lehrplaene/upload/lehrplaene_download/gymnasium_g8/gym8_mathematik.pdf (besucht am 12. 08. 2015).

[8] Ministerium für Schule und Weiterbildung des Landes Nordrhein-Westfalen. *Kernlehrplan für die Gesamtschule – Sekundarstufe I. Mathematik.* 2004. URL: http://www.schulentwicklung.nrw.de/lehrplaene/upload/lehrplaene_download/gesamtschule/gs_mathematik.pdf (besucht am 12. 08. 2015).

[9] Ministerium für Schule und Weiterbildung des Landes Nordrhein-Westfalen. *Kernlehrplan für die Sekundarstufe II Gymnasium / Gesamtschule. Mathematik.* 2013. URL: http://www.schulentwicklung.nrw.de/lehrplaene/upload/klp_SII/m/GOSt_Mathematik_Endfassung.pdf (besucht am 12. 08. 2015).

[10] T. Takasu. *Research and Development of Precise Positioning Technology by Global Navigation Satellite System.* 2005–2015. URL: http://gpspp.sakura.ne.jp/indexe.html (besucht am 12. 08. 2015).

[11] J.-M. Zogg. *GPS. Essentials of Satellite Navigation.* Compendium. 2009. URL: https://www.u-blox.com/sites/default/files/products/documents/GPS-Compendium_Book_(GPS-X-02007).pdf (besucht am 12. 08. 2015).

8. Abbildungsverzeichnis

9. Tabellenverzeichnis

A. Programmausdrucke

A.1. Das Hauptprogramm gps.m

```
%   Wie funktioniert eigentlich... GPS
%       ... uns was hat das mit Mathe zu tun?

%   CAMMP - COMPUTATIONAL AND MATHEMATICAL MODELING PROGRAM
%   RWTH AACHEN UNIVERSITY 2015

%   Starte das Skript durch Eingabe von >> gps << im Command Window

clc
format
clearvars -except Satellitendaten

Phase = 'Modellschritt';

%% Einlesen der Satellitendaten
display(sprintf('\n<strong>Einlesen der Satellitendaten</strong>\n'))

%   Einlesen der Satellitendaten falls nicht vorhanden
if ~exist('Satellitendaten','var')
    Dateien = {'114255.obs','114255.nav'};
    Satellitendaten = einlesen_Satellitendaten(Dateien);
end

%   Auswählen eines Datensatzes
Datensatznummer = 1;
Datensatz = Satellitendaten(Datensatznummer);

%   Satelliten, deren Nachrichten empfangen wurden
Satelliten = Datensatz.Satelliten;

%   Auswählen der Satelliten
Satellitenauswahl = [1 2 3];

%   Auslesen der ausgewählten Satellitendaten
tE = Datensatz.Empfangszeiten(Satellitenauswahl);
tS = Datensatz.Sendezeiten(Satellitenauswahl);
Ephemeriden = Datensatz.Ephemeriden(:,Satellitenauswahl);

%   Anzeigen der Ergebnisse
format shortG
display(Datensatz,sprintf('Datensatz Nummer %d',Datensatznummer))
display(Satelliten, ...
    'Satelliten, deren Nachrichten empfangen wurden [Satelliten]')
display(Satellitenauswahl, ...
    'Nummern der ausgewählten Satelliten [Satellitenauswahl]')
format bank
display(tE, ...
    'Empfangszeiten der Nachrichten in Sekunden seit Wochenbeginn [tE]')
display(tS, ...
    'Sendezeiten der Nachrichten in Sekunden seit Wochenbeginn [tS]')
display(sprintf(['\nDie Positionsdaten der Satelliten werden nicht ' ...
    'angezeigt [Ephemeriden]\n']))
clearvars Dateien Datensatznummer

%% Schritt 1 | Berechnen der Satellitenkoordinaten
display(sprintf(['\n<strong>Schritt 1 | Berechnen der ' ...
    'Satellitenkoordinaten</strong>\n']))
```

```
%   Berechnen der Satellitenkoordinaten
[xS,yS,zS] = berechne_Satellitenkoordinaten(NaN,Ephemeriden);

%   Überprüfen der Satellitenkoordinaten
ueberpruefe_Loesung(Phase,'Schritt 1', ...
    xS,yS,zS,Datensatz,Satellitenauswahl);

%   Anzeigen der Ergebnisse
format bank
display([xS(1) yS(1) zS(1)], ...
    'Koordinaten des ersten Satelliten in Meter [xS(1) yS(1) zS(1)]')
display([xS(2) yS(2) zS(2)], ...
    'Koordinaten des zweiten Satelliten in Meter [xS(2) yS(2) zS(2)]')
display([xS(3) yS(3) zS(3)], ...
    'Koordinaten des dritten Satelliten in Meter [xS(3) yS(3) zS(3)]')

%% Schritt 2 | Berechnen der Pseudoentfernungen
display(sprintf(['\n<strong>Schritt 2 | Berechnen der ' ...
    'Pseudoentfernungen</strong>\n']))

%   Aufstellen der Berechnungsformel
berechne_Pseudoentfernungen = @(tS,tE) NaN;

%   Berechnen der Pseudoentfernungen
dS = berechne_Pseudoentfernungen(tS,tE);

%   Überprüfen der Pseudoentfernungen
ueberpruefe_Loesung(Phase,'Schritt 2',dS,Datensatz,Satellitenauswahl);

%   Anzeigen der Ergebnisse
format bank
display([dS(1) dS(2) dS(3)], ...
    'Pseudoentfernungen zu den Satelliten in Meter [dS(1) dS(2) dS(3)]')

%% Schritt 3 | Berechnen der Empfängerkoordinaten
display(sprintf(['\n<strong>Schritt 3 | Berechnen der ' ...
    'Empfängerkoordinaten</strong>\n']))

%   Aufstellen des Gleichungssystems
Gleichungssystem = @(xE,yE,zE) [NaN
                                NaN
                                NaN];

%   Überprüfen des Gleichungssystems
ueberpruefe_Loesung(Phase,'Schritt 3',Gleichungssystem,xS,yS,zS,dS);

%   Definiere Funktion und Startpunkt für fsolve
Funktion = @(X) Gleichungssystem(X(1),X(2),X(3));
Startpunkt = [0 0 0];

%   Berechnen einer Lösung in der Nähe des Startpunktes
X = fsolve(Funktion,Startpunkt);

%   Speichern der berechneten Lösung
xE = X(1);
yE = X(2);
zE = X(3);

%   Überprüfen der Empfängerkoordinaten
ueberpruefe_Loesung(Phase,'Schritt 3',xE,yE,zE,xS,yS,zS,dS);
```

```
%   Anzeigen der Ergebnisse
format bank
display([xE yE zE],'Koordinaten des Empfängers in Meter [xE yE zE]')
clearvars Funktion Startpunkt X

%% Schritt 4 | Umrechnen in geographische Höhe, Breite und Länge
display(sprintf(['\n<strong>Schritt 4 | Umrechnen in ' ...
    'geographische Höhe, Breite und Länge</strong>\n']))

%   Aufstellen der Umrechnungsformeln
berechne_h      = @(xE,yE,zE) NaN;
berechne_phi    = @(xE,yE,zE) NaN;
berechne_lambda = @(xE,yE,zE) NaN;

%   Berechnen der geographischen Koordinaten
h      = berechne_h(xE,yE,zE);
phi    = berechne_phi(xE,yE,zE);
lambda = berechne_lambda(xE,yE,zE);

%   Überprüfen der geographischen Koordinaten
ueberpruefe_Loesung(Phase,'Schritt 4',h,xE,yE,zE);
ueberpruefe_Loesung(Phase,'Schritt 4',phi,xE,yE,zE);
ueberpruefe_Loesung(Phase,'Schritt 4',lambda,xE,yE,zE);

%   Anzeigen der Ergebnisse
format shortG
display(h,'Geographische Höhe des Empfängers in Meter [h]')
display([phi lambda], ...
    'Geographische Breite und Länge des Empfängers in Grad [phi lambda]')

%% Anzeigen der Empfängerposition
display(sprintf('\n<strong>Anzeigen der Empfängerposition</strong>\n'))

%   Auswählen des Kartenausschnitts
Kartenausschnitt_manuell = true;
Zentrum = '50.5,6.8';
Zoom = '8';

%   Erzeugen der URL
url = sprintf('https://www.google.de/maps/place/%f,%f',phi,lambda);
if Kartenausschnitt_manuell
    url = [url '/@' Zentrum ',' Zoom 'z'];
end

%   Anzeigen der URL im Browser
web(url,'-browser')
display(url,'URL für die Anzeige im Webbrowser')
clearvars Kartenausschnitt_manuell Zentrum Zoom url
```

A.2. Die Funktionsdatei einlesen_Satellitendaten.m

```
function Satellitendaten = einlesen_Satellitendaten(Dateien)
%EINLESEN_SATELLITENDATEN liest Einträge aus den Rinexdateien aus
%   Die Dateien werden ausgelesen und strukturiert abgespeichert.

warning('off','GPS:einlesen_Satellitendaten')

[Obs,Nav] = readrnx(Dateien);

Satellitendaten = [];

IDnumbers = unique(Obs(:,7),'stable');

for id = IDnumbers.'

    obs = Obs(Obs(:,7)==id & Obs(:,8)==0,:);

    if isempty(obs)
        warning('GPS:einlesen_Satellitendaten', ...
            'Keine gültigen Daten bei ID-Nummer %d',id)
        continue
    end

    if size(unique(obs(:,1:6),'rows'),1) ~= 1
        warning('GPS:einlesen_Satellitendaten', ...
            'Empfangszeiten der Satellitensignale sind verschieden.');
        continue
    end

    Anzahl_Satelliten = size(obs,1);
    Signalstaerken = NaN(1,Anzahl_Satelliten);
    Wochennummern  = NaN(1,Anzahl_Satelliten);
    Empfangszeiten = NaN(1,Anzahl_Satelliten);
    Sendezeiten    = NaN(1,Anzahl_Satelliten);
    Ephemeriden    = NaN(21,Anzahl_Satelliten);

    n = 0;

    for i = 1 : Anzahl_Satelliten
        % Auslesen der Satellitennummern
        prn = obs(i,9);
        % Berechnen von GPS-Woche und Empfangszeit in der GPS-Woche
        day = datenum(obs(i,1:3))-datenum(1980,1,6);
        week = floor(day/7);
        time = mod(day,7)*86400+obs(i,4:6)*[3600;60;1];
        % Finde korrekte PRN-Nummer (nur wenn SV health == 0)
        nav = Nav(Nav(:,1)==prn & Nav(:,32)==0,:);
        % Prüfe, ob Daten für diese Nummer vorhanden sind
        if isempty(nav)
            warning('GPS:einlesen_Satellitendaten', ...
                'Keine gültigen Daten für Satellit mit PRN %d',prn);
            continue
        end
        % Prüfe, ob es Daten in einem gültigen Zeitintervall gibt
        nav = nav(abs((week-nav(:,29))*604800+(time-nav(:,19))) <= ...
            nav(:,36)/2*3600,:);
        if isempty(nav)
            warning('GPS:einlesen_Satellitendaten', ...
                'Keine gültigen Daten für Satellit mit PRN %d',prn);
            continue
```

```
        end
        % Finde die Ephemeriden, deren Referenzzeit TOE am nächsten ist
        [~,row] = min(abs((week-nav(:,29))*604800+(time-nav(:,19))));
        % Anpassen der GPS-Woche & Berechnen der Empfangs- und Sendezeit
        GPS_Woche = nav(row,29);
        Empfangszeit = time+(week-GPS_Woche)*604800;
        Sendezeit = Empfangszeit-obs(i,10)/299792458;
        % Berechnen der Referenzzeit Toc
        % (obwohl Toc warscheinlich immer gleich Toe ist)
        day = datenum(nav(row,2:4))-datenum(1980,1,6);
        week = floor(day/7);
        time = mod(day,7)*86400+nav(row,5:7)*[3600;60;1];
        toc = time+(week-GPS_Woche)*604800;
        if toc ~= nav(row,19)
            warning('GPS:einlesen_Satellitendaten', ...
                'Toc ist ungleich Toe.');
            % Der Fall Toc ungleich Toe ist in Ordnung.
            % Er ist mir bisher nur nie begegnet.
            % Selbst Kai Borre setzt in seiner GPS Easy Suite
            % einfach Toc gleich Toe.
            % Ich vermute aber, dass es durchaus Fälle gibt,
            % wo es einen Unterschied gibt,
            % denn sonst wäre die Variable Toc überflüssig.
            % Deshalb hier kein Abbruch mit "continue".
        end
        % Kopiere Daten
        n = n+1;
        Wochennummern(n)  = GPS_Woche;
        Empfangszeiten(n) = Empfangszeit;
        Sendezeiten(n)    = Sendezeit;
        Signalstaerken(n) = obs(i,11);
        % Kopiere Ephemeriden
        Ephemeriden(1,n)  = nav(row,14); % M0
        Ephemeriden(2,n)  = nav(row,13); % Delta n
        Ephemeriden(3,n)  = nav(row,16); % e
        Ephemeriden(4,n)  = nav(row,18); % sqrt(A)
        Ephemeriden(5,n)  = nav(row,21); % OMEGA0
        Ephemeriden(6,n)  = nav(row,23); % i0
        Ephemeriden(7,n)  = nav(row,25); % omega
        Ephemeriden(8,n)  = nav(row,26); % OMEGADOT
        Ephemeriden(9,n)  = nav(row,27); % idot
        Ephemeriden(10,n) = nav(row,15); % Cuc
        Ephemeriden(11,n) = nav(row,17); % Cus
        Ephemeriden(12,n) = nav(row,24); % Crc
        Ephemeriden(13,n) = nav(row,12); % Crs
        Ephemeriden(14,n) = nav(row,20); % Cic
        Ephemeriden(15,n) = nav(row,22); % Cis
        Ephemeriden(16,n) = nav(row,19); % Toe
        Ephemeriden(17,n) = nav(row,8);  % af0
        Ephemeriden(18,n) = nav(row,9);  % af1
        Ephemeriden(19,n) = nav(row,10); % af2
        Ephemeriden(20,n) = toc;         % Toc
        Ephemeriden(21,n) = prn;         % prn
    end

    if n<4
        warning('GPS:einlesen_Satellitendaten', ...
            'Zu wenig Daten bei ID %d',id)
        continue
    end
```

```
    Datensatz = struct('Satelliten',1:n,...
        'Wochennummern',Wochennummern(:,1:n),...
        'Empfangszeiten',Empfangszeiten(:,1:n),...
        'Sendezeiten',Sendezeiten(:,1:n),...
        'Signalstaerken',Signalstaerken(:,1:n),...
        'Ephemeriden',Ephemeriden(:,1:n));

    Satellitendaten = [Satellitendaten Datensatz];
end
end


function [obs,nav] = readrnx(files)
% READRNX : read rinex observation data/navigation message files
%
% Read RINEX (Receiver Independent Exchange Format) GPS observation data
% (OBS) and navigation message (NAV) files. Only supports GPS.
%
% Original file by T.TAK5ASU from 2006 modified for CAMMP in 2014
% Origin: http://gpspp.sakura.ne.jp/prog/rtkdemo/readrnx.m
%
% argin  : files = RINEX OBS/NAV file paths (cell array)
%
% argout : obs = observation data sorted by sampling time (nan:no data)
%          obs(n,1:6) : sampling time [year,month,day,hour,min,sec]
%          obs(n,7)   : id number
%          obs(n,8)   : health flag (0 = data is valid)
%          obs(n,9)   : satellite PRN number
%          obs(n,10)  : C1 pseudorange (m)
%          obs(n,11)  : S1 signal strength (dBHz)
%          nav = navigation messages
%          nav(n,1)   : satellite PRN number
%          nav(n,2:7) : Toc [year,month,day,hour,min,sec]
%          nav(n,8:10): SV Clock [bias,drift,drift-rate]
%          nav(n,11): IODE,     nav(n,12): Crs,      nav(n,13): Delta n
%          nav(n,14): M0,       nav(n,15): Cuc,      nav(n,16): e
%          nav(n,17): Cus,      nav(n,18): sqrt(A),  nav(n,19): Toe
%          nav(n,20): Cic,      nav(n,21): OMEGA0,   nav(n,22): Cis
%          nav(n,23): i0,       nav(n,24): Crc,      nav(n,25): omega
%          nav(n,26): OMEGADOT, nav(n,27): idot,     nav(n,28): Codes on L2
%          nav(n,29): GPS Week #,    nav(n,30): L2 P data flag
%          nav(n,31): SV accuracy,   nav(s,32): SV health
%          nav(s,33): TGD,           nav(n,34): IODC
%          nav(n,35): Trans. time,   nav(n,36): fit interval
%          nav(n,37:38): spare

obs=[]; nav=[]; id=1;
for n=1:length(files)
    f=fopen(files{n},'rt');
    if f<0, error(['Fehler beim Öffnen der Datei : ',files{n}]); end
    display(['Lese Rinex Datei ... : ',files{n}]);
    [trnx,tobs]=readrnxh(f);
    if trnx=='O'
        [o, id]=readrnxo(f,tobs,id); obs=[obs;o];
    elseif trnx=='N'
        v=readrnxn(f); nav=[nav;v];
    end
    fclose(f);
end
if ~isempty(obs), obs=sortrows(obs); end
if ~isempty(nav), nav=unique(nav,'rows'); end
```

```
end

% read rinex header ------------------------------------------------------
function [trnx,tobs]=readrnxh(f)
trnx=''; tobs={};
while 1
    s=fgetl(f); if ~ischar(s), break, end
    label=subs(s,61,20);
    if strfind(label,'RINEX VERSION / TYPE')
        trnx=subs(s,21,1);
    elseif strfind(label,'# / TYPES OF OBSERV')
        p=11;
        for i=1:s2n(s,1,6)
            if p>=59, s=fgetl(f); p=11; end
            tobs{i}=subs(s,p,2); p=p+6;
        end
    elseif strfind(label,'END OF HEADER'), break
    end
end
end

% read rinex observation data ---------------------------------------------
function [obs,id]=readrnxo(f,tobs,id)
to={'C1','S1'}; l=length(to); ind=zeros(1,length(tobs));
for n=1:l, i=find(strcmp(to{n},tobs)); if ~isempty(i), ind(i)=n; end
end
obs=zeros(100000,9+l); n=0;
while 1
    s=fgetl(f); if ~ischar(s), break, end
    e=s2e(s,1,26);
    if length(e)>=6
        if e(1)<80, e(1)=2000+e(1); else e(1)=1900+e(1); end
        flag=s2n(s,27,3);
        a=[e,id,flag];
        ns=s2n(s,30,3);
        obs(n+(1:ns),1:8)=repmat(a,ns,1);
        p=33;
        for i=1:ns
            if p>=69, s=fgetl(f); p=33; end
            if any(subs(s,p,1)==' G'), obs(n+i,9)=s2n(s,p+1,2); end
            p=p+3;
        end
        for i=1:ns
            s=fgetl(f); p=1;
            for j=1:length(tobs)
                if p>=80, s=fgetl(f); p=1; end
                if ind(j)>0, obs(n+i,9+ind(j))=s2n(s,p,14); end
                p=p+16;
            end
        end
        n=n+ns; id=id+1;
    end
end
obs=obs(1:n,:); obs=obs(obs(:,9)>0,:);
end

% read rinex navigation message ------------------------------------------
function nav=readrnxn(f)
nav=zeros(2000,38); n=0; m=1;
while 1
    s=fgetl(f); if ~ischar(s), break, end
```

```
    s=strrep(s,'D','E');
    if m==1, prn=s2n(s,1,2); e=s2e(s,3,20);
        if e(1)<80, e(1)=2000+e(1); else e(1)=1900+e(1); end
        a=[prn,e];
    else a=[a,s2n(s,4,19)];
    end
    for p=23:19:61, a=[a,s2n(s,p,19)]; end
    if m<8, m=m+1;
    else n=n+1; nav(n,:)=a; m=1;
    end
end
nav=nav(1:n,1:36);
end

% string to number/epoch/substring ------------------------------------
function a=s2n(s,p,n), a=sscanf(subs(s,p,n),'%f');
    if isempty(a), a=nan; end
end

function e=s2e(s,p,n), e=sscanf(subs(s,p,n),'%d %d %d %d %f',6)';
end

function s=subs(s,p,n), s=s(p:min(p+n-1,length(s)));
end
```

A.3. Die Funktionsdatei berechne_Satellitenkoordinaten.m

```
function [x,y,z] = berechne_Satellitenkoordinaten(t,Ephemeriden)
%BERECHNE_SATELLITENKOORDINATEN berechnet X,Y,Z ECEF Koordinaten
%   Eingabe:
%       - t = Vektor der Zeitpunkte in Sekunden
%       - Ephemeriden = Ephemeridenmatrix
%   Ausgabe:
%       - x,y,z = Koordinaten der Satelliten in Meter
%   Aufruf:
%       [x,y,z] = berechne_Satellitenkoordinaten(t,Ephemeriden)

% Lese Ephemeriden
M0      = Ephemeriden(1,:);
deltan  = Ephemeriden(2,:);
ecc     = Ephemeriden(3,:);
rootA   = Ephemeriden(4,:);
Omega0  = Ephemeriden(5,:);
i0      = Ephemeriden(6,:);
omega   = Ephemeriden(7,:);
Omegadot = Ephemeriden(8,:);
idot    = Ephemeriden(9,:);
cuc     = Ephemeriden(10,:);
cus     = Ephemeriden(11,:);
crc     = Ephemeriden(12,:);
crs     = Ephemeriden(13,:);
cic     = Ephemeriden(14,:);
cis     = Ephemeriden(15,:);
toe     = Ephemeriden(16,:);

% Vorschrift zur Berechnung der Satellitenkoordinaten
mu = 3.986005e14;
Omegae_dot = 7.2921151467e-5;
A = rootA.*rootA;
n0 = sqrt(mu./A.^3);
tk = t-toe;
n = n0+deltan;
M = M0+n.*tk;
E = M;
for i = 1:20
    E_old = E;
    E = M+ecc.*sin(E);
    dE = E-E_old;
    if max(abs(dE)) < 1.e-12
        break;
    end
end
v = atan2(sqrt(1-ecc.^2).*sin(E), cos(E)-ecc);
phi = v+omega;
u = phi                  + cuc.*cos(2*phi)+cus.*sin(2*phi);
r = A.*(1-ecc.*cos(E)) + crc.*cos(2*phi)+crs.*sin(2*phi);
i = i0+idot.*tk          + cic.*cos(2*phi)+cis.*sin(2*phi);
Omega = Omega0+(Omegadot-Omegae_dot).*tk-Omegae_dot*toe;
x1 = cos(u).*r;
y1 = sin(u).*r;
x = x1.*cos(Omega)-y1.*cos(i).*sin(Omega);
y = x1.*sin(Omega)+y1.*cos(i).*cos(Omega);
z = y1.*sin(i);
end
```

A.4. Die Funktionsdatei berechne_Satellitenuhrfehler.m

```
function Delta_t = berechne_Satellitenuhrfehler(t,Ephemeriden)
%BERECHNE_SATELLITENUHRFEHLER Berechnet Zeitfehler der Satellitenuhren
%   Eingabe:
%       - t = Vektor der Zeitpunkte in Sekunden
%       - Ephemeriden = Ephemeridenmatrix
%   Ausgabe:
%       - Delta_t = Vektor der Uhrfehler in Sekunden
%   Aufruf:
%       Delta_t = berechne_Satellitenuhrfehler(t,Ephemeriden)

% Lese Ephemeriden
M0     = Ephemeriden(1,:);
deltan = Ephemeriden(2,:);
ecc    = Ephemeriden(3,:);
rootA  = Ephemeriden(4,:);
toe    = Ephemeriden(16,:);
af0    = Ephemeriden(17,:);
af1    = Ephemeriden(18,:);
af2    = Ephemeriden(19,:);
toc    = Ephemeriden(20,:);

% Vorschrift zur Berechnung der Satellitenuhrfehler
mu = 3.986005e14;
c = 2.99792458e8;
F = -2*sqrt(mu)/c^2;
A = rootA.*rootA;
n0 = sqrt(mu./A.^3);
tk = t-toe;
n = n0+deltan;
M = M0+n.*tk;
E = M;
for i = 1:20
    E_old = E;
    E = M+ecc.*sin(E);
    dE = E-E_old;
    if max(abs(dE)) < 1.e-12
        break;
    end
end
Delta_tr = F*ecc.*rootA.*sin(E);
tk = t-toc;
Delta_t = af0+af1.*tk+af2.*tk.^2+Delta_tr;
end
```

A.5. Die Funktionsdatei ueberpruefe_loesung.m

```
function ueberpruefe_Loesung(Phase,Schritt,varargin)
%UEBERPRUEFE_LOESUNG Überprüft die Lösungen der Aufgaben 1 bis 4
%   Bitte durch Aufruf von >> pcode('ueberpruefe_Loesung') <<
%   zu einer geschützten Datei "ueberpruefe_Loesung.p" machen.
%   Die Schüler dürfen nur die geschützte Datei "ueberpruefe_Loesung.p"
%   erhalten, da aus dem Inhalt der ungeschützten Datei
%   "ueberpruefe_Loesung.m" Rückschlüsse auf die Lösungen möglich sind.

switch Phase
case 'Modellschritt'
    switch Schritt
    case 'Schritt 1'
        [msgString,errString] = ...
            teste_Satellitenkoordinaten(Schritt,varargin{1:end});
    case 'Schritt 2'
        [msgString,errString] = ...
            teste_Pseudoentfernungen(Schritt,varargin{1:end});
    case 'Schritt 3'
        switch inputname(3)
        case 'Gleichungssystem'
            [msgString,errString] = ...
                teste_Gleichungssystem(Schritt,varargin{1:end});
        case 'xE'
            [msgString,errString] = ...
                teste_Empfaengerkoordinaten(Schritt,varargin{1:end});
        otherwise
            error('Der zu überprüfende Wert ist unbekannt.')
        end
    case 'Schritt 4'
        switch inputname(3)
        case 'h'
            [msgString,errString] = ...
                teste_h(Schritt,varargin{1:end});
        case 'phi'
            [msgString,errString] = ...
                teste_phi(Schritt,varargin{1:end});
        case 'lambda'
            [msgString,errString] = ...
                teste_lambda(Schritt,varargin{1:end});
        otherwise
            error('Der zu überprüfende Wert ist unbekannt.')
        end
    otherwise
        error('Der zu überprüfende Schritt ist unbekannt.')
    end

    if ~isempty(msgString)
        disp(msgString);
    end

    pause(0.8)

    if ~isempty(errString)
        error(errString)
    end

case 'Modellverbesserung'
    disp(['Bei den Modellverbesserungen finden keine ' ...
        'Überprüfungen statt.'])
```

```
otherwise
    error(['Die Phase muss entweder ''Modellschritt'' oder' ...
        ' ''Modellverbesserung'' lauten.'])
end
end
```

A.5.1. Die Testfunktion teste_Satellitenkoordinaten

```
function [msgString,errString] = teste_Satellitenkoordinaten(Schritt, ...
    xS,yS,zS,Datensatz,Satellitenauswahl)

msgString = sprintf(['<strong>Bitte bearbeite ' Schritt '</strong>\n' ...
    'Berechne die Satellitenkoordinaten zu den richtigen Zeitpunkten.']);

errString = sprintf(['\n<strong>' Schritt ...
        ' ist nicht korrekt gelöst.</strong>\n' ...
        'Bitte überarbeite deine Lösung.']);

if ~isfloat(xS) || ~isfloat(yS) || ~isfloat(zS)
    return
end

if ~isreal(xS) || ~isreal(yS) || ~isreal(zS)
    return
end

if ~isrow(xS) || ~isrow(yS) || ~isrow(zS)
    return
end

if all(isnan(xS)) || all(isnan(yS)) || all(isnan(zS))
    errString = sprintf(['\n<strong>Bitte bearbeite ' Schritt ...
        '</strong>\n' ...
        'Ersetze das "NaN" in der Zeile\n' ...
        '    [xS,yS,zS] = berechne_Satellitenkoordinaten(NaN,' ...
        'Ephemeriden);\n' ...
        'durch die richtigen Zeiten.']);
    return
end

Loesung = [xS yS zS];

tE = Datensatz.Empfangszeiten(:,Satellitenauswahl);
tS = Datensatz.Sendezeiten(:,Satellitenauswahl);
Eph = Datensatz.Ephemeriden(:,Satellitenauswahl);

[x,y,z] = berechne_Satellitenkoordinaten(tS,Eph);
Musterloesung = [x y z];

if any(size(Loesung) ~= size(Musterloesung))
    return
end

toleranz = 1.0e-1;

if all(abs(Loesung-Musterloesung) <= toleranz)
    msgString = sprintf(['\n<strong>' Schritt ...
        ' scheint korrekt gelöst zu sein.</strong>\n']);
    errString = '';
    return
```

```
end

toleranz = 100;

if all(abs(Loesung-Musterloesung) <= toleranz)
    errString = sprintf(['\n<strong>' Schritt ...
        ' ist nicht korrekt gelöst.</strong>\n' ...
        'Die Zeiten sind fast richtig, aber zu grob gerundet.']);
    return
end

[x,y,z] = berechne_Satellitenkoordinaten(tE,Eph);
Falsche_Loesung = [x y z];

if any(size(Loesung) ~= size(Falsche_Loesung))
    return
end

toleranz = 1.0e-1;

if all(abs(Loesung-Falsche_Loesung) < toleranz)
    errString = sprintf(['\n<strong>' Schritt ...
        ' ist nicht korrekt gelöst.</strong>\n' ...
        'Die Satellitenkoordinaten sollen nicht zu den ' ...
        'Empfangszeiten berechnet werden.']);
    return
end
end
```

A.5.2. Die Testfunktion teste_Pseudoentfernungen

```
function [msgString,errString] = teste_Pseudoentfernungen(Schritt, ...
    dS,Datensatz,Satellitenauswahl)

msgString = sprintf(['<strong>Bitte bearbeite ' Schritt '</strong>\n' ...
    'Stelle die Formel für die Pseudoentfernungen auf.']);

errString = sprintf(['\n<strong>' Schritt ...
        ' ist nicht korrekt gelöst.</strong>\n' ...
        'Bitte überarbeite deine Lösung.']);

if ~isfloat(dS)
    return
end

if ~isreal(dS)
    return
end

if ~isrow(dS)
    return
end

if all(isnan(dS))
    errString = sprintf(['\n<strong>Bitte bearbeite ' Schritt ...
        '</strong>\n' ...
        'Ersetze das "NaN" in in der Zeile\n' ...
        '    berechne_Pseudoentfernungen = @(tS,tE) NaN;\n' ...
        'durch die richtige Formel.']);
    return
```

```
end

Loesung = dS;

tS = Datensatz.Sendezeiten(:,Satellitenauswahl);
tE = Datensatz.Empfangszeiten(:,Satellitenauswahl);

Musterloesung = (tE-tS)*299792458;

if any(size(Loesung) ~= size(Musterloesung))
    return
end

toleranz = 1.0e-1;

if all(abs(Loesung-Musterloesung) <= toleranz)
    msgString = sprintf(['\n<strong>' Schritt ...
        ' scheint korrekt gelöst zu sein.</strong>\n']);
    errString = '';
    return
end

Falsche_Loesung = (tS-tE)*299792458;

if any(size(Loesung) ~= size(Falsche_Loesung))
    return
end

toleranz = 1.0e-1;

if all(abs(Loesung-Falsche_Loesung) <= toleranz)
    errString = sprintf(['\n<strong>' Schritt ...
        ' ist nicht korrekt gelöst.</strong>\n' ...
        'Das Vorzeichen der Lösung ist falsch.']);
    return
end

Falsche_Loesung = abs(tE-tS)*299792458;

if any(size(Loesung) ~= size(Falsche_Loesung))
    return
end

toleranz = 1.0e-1;

if all(abs(Loesung-Falsche_Loesung) <= toleranz)
    errString = sprintf(['\n<strong>' Schritt ...
        ' ist nicht korrekt gelöst.</strong>\n' ...
        'Für die Pseudoentfernungen darf kein Betrag verwendet ' ...
        'werden, denn sie können durch Zeitfehler auch negativ sein.']);
    return
end

Falsche_Loesung = (tE-tS)*300000000;

if any(size(Loesung) ~= size(Falsche_Loesung))
    return
end

toleranz = 1.0e-1;
```

```
if all(abs(Loesung-Falsche_Loesung) <= toleranz)
    errString = sprintf(['\n<strong>' Schritt ...
        ' ist nicht korrekt gelöst.</strong>\n' ...
        'Der Wert für die Lichtgeschwindigkeit ist nicht genau genug.']);
    return
end

Falsche_Loesung = (tS-tE)*300000000;

if any(size(Loesung) ~= size(Falsche_Loesung))
    return
end

toleranz = 1.0e-1;

if all(abs(Loesung-Falsche_Loesung) <= toleranz)
    errString = sprintf(['\n<strong>' Schritt ...
        ' ist nicht korrekt gelöst.</strong>\n' ...
        'Das Vorzeichen der Lösung ist falsch.']);
    return
end

Falsche_Loesung = abs(tE-tS)*300000000;

if any(size(Loesung) ~= size(Falsche_Loesung))
    return
end

toleranz = 1.0e-1;

if all(abs(Loesung-Falsche_Loesung) <= toleranz)
    errString = sprintf(['\n<strong>' Schritt ...
        ' ist nicht korrekt gelöst.</strong>\n' ...
        'Für die Pseudoentfernungen darf kein Betrag verwendet ' ...
        'werden, denn sie können durch Zeitfehler auch negativ sein.']);
    return
end
end
```

A.5.3. Die Testfunktion teste_Gleichungssystem

```
function [msgString,errString] = teste_Gleichungssystem(Schritt, ...
Gleichungssystem,xS,yS,zS,dS)

msgString = sprintf(['<strong>Bitte bearbeite ' Schritt '</strong>\n' ...
    'Stelle das Gleichungssystem auf.']);

errString = sprintf(['\n<strong>' Schritt ...
    ' ist nicht korrekt gelöst.</strong>\n' ...
    'Bitte überarbeite das Gleichungssystem.']);

if ~isa(Gleichungssystem,'function_handle') || ...
    nargin(Gleichungssystem) ~= 3
    errString = sprintf(['\n<strong>Das Gleichungssystem ' ...
        'ist so nicht korrekt.</strong>\n' ...
        'Das Gleichungssystem muss folgendermaßen angegeben werden:\n' ...
        '    Gleichungssystem = @(xE,yE,zE) [GLEICHUNG 1\n' ...
        '                                    GLEICHUNG 2\n' ...
        '                                    GLEICHUNG 3];']);
    return
```

```
end

Y = Gleichungssystem(0,0,0);

if ~isfloat(Y)
    return
end

if ~isreal(Y)
    return
end

if ~isvector(Y)
    return
end

if length(Y) > 3
    errString = sprintf(['\n<strong>' Schritt ...
        ' ist nicht korrekt gelöst.</strong>\n' ...
        'Das Gleichungssystem muss aus genau drei Gleichungen ' ...
        'bestehen. Du hast zuviele Gleichungen.']);
    return
end

if length(Y) < 3
    errString = sprintf(['\n<strong>' Schritt ...
        ' ist nicht korrekt gelöst.</strong>\n' ...
        'Das Gleichungssystem muss aus genau drei Gleichungen ' ...
        'bestehen. Du hast zuwenig Gleichungen.']);
    return
end

if all(isnan(Y))
    errString = sprintf(['\n<strong>Bitte bearbeite ' Schritt ...
        '</strong>\n' ...
        'Ersetze die drei "NaN" in den Zeilen\n' ...
        '    Gleichungssystem = @(xE,yE,zE) [NaN\n' ...
        '                                    NaN\n' ...
        '                                    NaN];\n' ...
        'durch die richtigen Gleichungen.\n'...
        'Die Gleichungen müssen wie eine Funktion eingegeben ' ...
        'werden, d.h. sie müssen nach Null aufgelöst werden ' ...
        'und ohne Gleichheitszeichen und Nullen eingegeben werden.']);
    return
end

Mustersystem1 = @(xE,yE,zE) ...
    [sqrt((xS(1)-xE)^2+(yS(1)-yE)^2+(zS(1)-zE)^2)-dS(1) ...
    sqrt((xS(2)-xE)^2+(yS(2)-yE)^2+(zS(2)-zE)^2)-dS(2) ...
    sqrt((xS(3)-xE)^2+(yS(3)-yE)^2+(zS(3)-zE)^2)-dS(3)];

Mustersystem2 = @(xE,yE,zE) ...
    [(xS(1)-xE)^2+(yS(1)-yE)^2+(zS(1)-zE)^2-dS(1)^2 ...
    (xS(2)-xE)^2+(yS(2)-yE)^2+(zS(2)-zE)^2-dS(2)^2 ...
    (xS(3)-xE)^2+(yS(3)-yE)^2+(zS(3)-zE)^2-dS(3)^2];

Y1 = Mustersystem1(0,0,0);
Y2 = Mustersystem2(0,0,0);

if isrow(Y)
    Y0 = Y;
```

```
else
    Y0 = Y.';
end

if any(size(Y0) ~= size(Y1)) || any(size(Y0) ~= size(Y2))
    return
end

tolY = 1.0e-1;

if ~(any(abs(Y0-Y1) <= tolY) || any(abs(Y0-Y2) <= tolY))
    return
end

if isrow(Y)
    Funktion = @(X) Gleichungssystem(X(1),X(2),X(3));
else
    Funktion = @(X) Gleichungssystem(X(1),X(2),X(3)).';
end

Musterfunktion1 = @(X) Mustersystem1(X(1),X(2),X(3));
Musterfunktion2 = @(X) Mustersystem2(X(1),X(2),X(3));

Options = optimoptions('fsolve','Display','off');

Startpunkt = [0 0 0];

[X1,Y1] = fsolve(Musterfunktion1,Startpunkt,Options);
[X2,Y2] = fsolve(Musterfunktion2,Startpunkt,Options);

Loesung1 = Funktion(X1);
Loesung2 = Funktion(X2);

if ~isfloat(Loesung1) || ~isfloat(Loesung2)
    return
end

if ~isreal(Loesung1) || ~isreal(Loesung2)
    return
end

if ~isvector(Loesung1) || ~isvector(Loesung2)
    return
end

if any(size(Loesung1) ~= size(Y1)) || any(size(Loesung2) ~= size(Y2))
    return
end

tolY = 1.0e-1;

if all(abs(Loesung1-Y1) <= tolY) || all(abs(Loesung2-Y2) <= tolY)
    msgString = sprintf(['<strong>Das Gleichungssystem scheint ' ...
        'korrekt zu sein.</strong>']);
    errString = '';
    return
end

if any(abs(Loesung1-Y1) <= tolY) || any(abs(Loesung2-Y2) <= tolY)
    errString = sprintf(['\n<strong>Sehr gut. ' ...
        'Mach weiter so.</strong>\n' ...
```

```
        'Mindestens eine Gleichung scheint schon korrekt zu sein. ' ...
        'Bearbeite noch die anderen Gleichungen.']);
    return
end
end
```

A.5.4. Die Testfunktion teste_Empfaengerkoordinaten

```
function [msgString,errString] = teste_Empfaengerkoordinaten(Schritt, ...
    xE,yE,zE,xS,yS,zS,dS)

msgString = sprintf(['<strong>Bitte bearbeite ' Schritt '</strong>\n' ...
    'Stelle das Gleichungssystem auf.']);

errString = sprintf(['\n<strong>Das Gleichungssystem ist leider ' ...
    'doch nicht korrekt.</strong>\n' ...
    'Bitte überarbeite das Gleichungssystem.']);

if ~isfloat(xE) || ~isfloat(yE) || ~isfloat(zE)
    return
end

if ~isreal(xE) || ~isreal(yE) || ~isreal(zE)
    return
end

if ~isscalar(xE) || ~isscalar(yE) || ~isscalar(zE)
    return
end

Loesung = [xE yE zE];

Mustersystem1 = @(xE,yE,zE) ...
    [sqrt((xE-xS(1))^2+(yE-yS(1))^2+(zE-zS(1))^2)-dS(1) ...
    sqrt((xE-xS(2))^2+(yE-yS(2))^2+(zE-zS(2))^2)-dS(2) ...
    sqrt((xE-xS(3))^2+(yE-yS(3))^2+(zE-zS(3))^2)-dS(3)];
Mustersystem2 = @(xE,yE,zE) ...
    [(xE-xS(1))^2+(yE-yS(1))^2+(zE-zS(1))^2-dS(1)^2 ...
    (xE-xS(2))^2+(yE-yS(2))^2+(zE-zS(2))^2-dS(2)^2 ...
    (xE-xS(3))^2+(yE-yS(3))^2+(zE-zS(3))^2-dS(3)^2];

Musterfunktion1 = @(X) Mustersystem1(X(1),X(2),X(3));
Musterfunktion2 = @(X) Mustersystem2(X(1),X(2),X(3));

Options = optimoptions('fsolve','Display','off');

Startpunkt = [0 0 0];

X1 = fsolve(Musterfunktion1,Startpunkt,Options);
X2 = fsolve(Musterfunktion2,Startpunkt,Options);

tolX = 1.0e-1;

if all(abs(Loesung-X1) <= tolX) || all(abs(Loesung-X2) <= tolX)
    msgString = sprintf(['\n<strong>' Schritt ...
        ' scheint korrekt gelöst zu sein.</strong>\n']);
    errString = '';
    return
end
end
```

A.5.5. Die Testfunktion teste_h

```
function [msgString,errString] = teste_h(Schritt,h,xE,yE,zE)

msgString = sprintf(['<strong>Bitte bearbeite ' Schritt '</strong>\n' ...
    'Stelle die Umrechnungsformel für die geographische Höhe auf.']);

errString = sprintf(['\n<strong>Die geographische Höhe ist ' ...
        'nicht korrekt.</strong>\n' ...
        'Bitte überarbeite die Formel "berechne_h" oder überprüfe ' ...
        'den Erdradius. Der Erdradius muss in Meter angegeben werden.']);

Loesung = h;

if ~isfloat(Loesung)
    return
end

if ~isreal(Loesung)
    return
end

if ~isscalar(Loesung)
    return
end

if all(isnan(Loesung))
    errString = sprintf(['\n<strong>Bitte bearbeite ' Schritt ...
        '</strong>\n' ...
        'Ersetze das "NaN" in der Zeile\n' ...
        '    berechne_h = @(xE,yE,zE) NaN;\n' ...
        'durch die richtige Formel.']);
    return
end

Musterloesung1 = sqrt(xE^2+yE^2+zE^2)-6378137.0;
Musterloesung2 = sqrt(xE^2+yE^2+zE^2)-6371000.8;

if any(size(Loesung) ~= size(Musterloesung1)) || ...
    any(size(Loesung) ~= size(Musterloesung2))
    return
end

toleranz = 1.0e-03;

if all(abs(Loesung-Musterloesung1) <= toleranz) || ...
    all(abs(Loesung-Musterloesung2) <= toleranz)
    msgString = sprintf(['<strong>Die Formel für "h" scheint ' ...
        'korrekt zu sein.</strong>']);
    errString = '';
    return
end

toleranz = 1.0;

if all(abs(Loesung-Musterloesung1) <= toleranz) || ...
    all(abs(Loesung-Musterloesung2) <= toleranz)
    errString = sprintf(['<strong>Die Formel für "h" scheint ' ...
```

```
        'korrekt zu sein, aber der Erdradius ist nicht exakt ' ...
        'genug.</strong>\n' ...
        'Verwende entweder 6378137.0 Meter (Äquatorradius) ' ...
        'oder 6371000.8 Meter (mittlerer Erdradius).']);
    return
end
end
```

A.5.6. Die Testfunktion teste_phi

```
function [msgString,errString] = teste_phi(Schritt,phi,xE,yE,zE)

msgString = sprintf(['<strong>Bitte bearbeite ' Schritt '</strong>\n' ...
    'Stelle die Umrechnungsformel für die geographische Breite auf.']);

errString = sprintf(['\n<strong>Die geographische Breite ist ' ...
    'nicht korrekt.</strong>\n' ...
    'Bitte überarbeite die Formel "berechne_phi".']);

Loesung = phi;

if ~isfloat(Loesung)
    return
end

if ~isreal(Loesung)
    return
end

if ~isscalar(Loesung)
    return
end

if all(isnan(Loesung))
    errString = sprintf(['\n<strong>Bitte bearbeite ' Schritt ...
        '</strong>\n' ...
        'Ersetze das "NaN" in der Zeile\n' ...
        '    berechne_phi = @(xE,yE,zE) NaN;\n' ...
        'durch die richtige Formel.']);
    return
end

Musterloesung = atand(zE/sqrt(xE^2+yE^2));

if any(size(Loesung) ~= size(Musterloesung))
    return
end

toleranz = 1.0e-03;

if all(abs(Loesung-Musterloesung) <= toleranz)
    msgString = sprintf(['<strong>Die Formel für "phi" scheint ' ...
        'korrekt zu sein.</strong>']);
    errString = '';
    return
end

Falsche_Loesung = atan(zE/sqrt(xE^2+yE^2));

if any(size(Loesung) ~= size(Falsche_Loesung))
```

```
    return
end

toleranz = 1.0e-03;

if all(abs(Loesung-Falsche_Loesung) <= toleranz)
    msgString = sprintf(['<strong>Die Formel für "phi" scheint ' ...
        'korrekt zu sein, aber du verwendest das falsche ' ...
        'Winkelmaß.</strong>']);
    errString = sprintf(['\n<strong>Die Formel für "phi" scheint ' ...
        'korrekt zu sein, aber du verwendest das falsche ' ...
        'Winkelmaß.</strong>\n' ...
        'Verwende bitte Grad und nicht Radiant.']);
    return
end

Falsche_Loesung_1 = asind(zE/6378137.0);
Falsche_Loesung_2 = asind(zE/6371000.8);
Falsche_Loesung_3 = acosd(sqrt(xE^2+yE^2)/6378137.0);
Falsche_Loesung_4 = acosd(sqrt(xE^2+yE^2)/6371000.8);

if any(size(Loesung) ~= size(Falsche_Loesung_1)) || ...
    any(size(Loesung) ~= size(Falsche_Loesung_2)) || ...
    any(size(Loesung) ~= size(Falsche_Loesung_3)) || ...
    any(size(Loesung) ~= size(Falsche_Loesung_4))
    return
end

toleranz = 1.0e-03;

if all(abs(Loesung-Falsche_Loesung_1) <= toleranz) || ...
    all(abs(Loesung-Falsche_Loesung_2) <= toleranz) || ...
    all(abs(Loesung-Falsche_Loesung_3) <= toleranz) || ...
    all(abs(Loesung-Falsche_Loesung_4) <= toleranz)
    msgString = sprintf(['<strong>Die Formel für "phi" ist fast ' ...
        'korrekt, aber du machst eine falsche Annahme.</strong>']);
    errString = sprintf(['\n<strong>Die Formel für "phi" ist fast ' ...
        'korrekt, aber du machst eine falsche Annahme.</strong>\n' ...
        'Beachte bitte, dass der Empfänger nicht genau auf der ' ...
        'Erdkugel sein muss (es gibt Berge und Täler).']);
    return
end

Falsche_Loesung_1 = asin(zE/6378137.0);
Falsche_Loesung_2 = asin(zE/6371000.8);
Falsche_Loesung_3 = acos(sqrt(xE^2+yE^2)/6378137.0);
Falsche_Loesung_4 = acos(sqrt(xE^2+yE^2)/6371000.8);

if any(size(Loesung) ~= size(Falsche_Loesung_1)) || ...
    any(size(Loesung) ~= size(Falsche_Loesung_2)) || ...
    any(size(Loesung) ~= size(Falsche_Loesung_3)) || ...
    any(size(Loesung) ~= size(Falsche_Loesung_4))
    return
end

toleranz = 1.0e-03;

if all(abs(Loesung-Falsche_Loesung_1) <= toleranz) || ...
    all(abs(Loesung-Falsche_Loesung_2) <= toleranz) || ...
    all(abs(Loesung-Falsche_Loesung_3) <= toleranz) || ...
    all(abs(Loesung-Falsche_Loesung_4) <= toleranz)
```

```
    msgString = sprintf(['<strong>Die Formel für "phi" ist fast ' ...
        'korrekt, aber du verwendest das falsche Winkelmaß und ' ...
        'machst eine falsche Annahme.</strong>']);
    errString = sprintf(['\n<strong>Die Formel für "phi" ist fast ' ...
        'korrekt, aber du verwendest das falsche Winkelmaß und ' ...
        'machst eine falsche Annahme.</strong>\n' ...
        'Verwende bitte Grad und nicht Radiant. Beachte auch, dass ' ...
        'der Empfänger nicht genau auf der Erdkugel sein muss ' ...
        '(es gibt Berge und Täler).']);
    return
end
end
```

A.5.7. Die Testfunktion teste_lambda

```
function [msgString,errString] = teste_lambda(Schritt,lambda,xE,yE,~)

msgString = sprintf(['<strong>Bitte bearbeite ' Schritt '</strong>\n' ...
    'Stelle die Umrechnungsformel für die geographische Länge auf.']);

errString = sprintf(['\n<strong>Die geographische Länge ist ' ...
    'nicht korrekt.</strong>\n' ...
    'Bitte überarbeite die Formel "berechne_lambda".']);

Loesung = lambda;

if ~isfloat(Loesung)
    return
end

if ~isreal(Loesung)
    return
end

if ~isscalar(Loesung)
    return
end

if all(isnan(Loesung))
    errString = sprintf(['\n<strong>Bitte bearbeite ' Schritt ...
        '</strong>\n' ...
        'Ersetze das "NaN" in der Zeile\n' ...
        '    berechne_lambda = @(xE,yE,zE) NaN;\n' ...
        'durch die richtige Formel.']);
    return
end

Musterloesung = atan2d(yE,xE);

if any(size(Loesung) ~= size(Musterloesung))
    return
end

toleranz = 1.0e-03;

if all(abs(Loesung-Musterloesung) <= toleranz)
    msgString = sprintf(['<strong>Die Formel für "lambda" scheint ' ...
        'korrekt zu sein.</strong>']);
    errString = '';
    return
```

```
end

Falsche_Loesung = atan2(yE,xE);

if any(size(Loesung) ~= size(Falsche_Loesung))
    return
end

toleranz = 1.0e-03;

if all(abs(Loesung-Falsche_Loesung) <= toleranz)
    msgString = sprintf(['<strong>Die Formel für "lambda" scheint ' ...
        'korrekt zu sein, aber du verwendest das falsche ' ...
        'Winkelmaß.</strong>']);
    errString = sprintf(['\n<strong>Die Formel für "lambda" scheint ' ...
        'korrekt zu sein, aber du verwendest das falsche ' ...
        'Winkelmaß.</strong>\n' ...
        'Verwende bitte Grad und nicht Radiant.']);
    return
end
end
```

A.6. Die Lösung der Modellschritte gps_master.m

```
%   Wie funktioniert eigentlich... GPS
%       ... uns was hat das mit Mathe zu tun?

%   CAMMP - COMPUTATIONAL AND MATHEMATICAL MODELING PROGRAM
%   RWTH AACHEN UNIVERSITY 2015

%   Starte das Skript durch Eingabe von >> gps << im Command Window

clc
format
clearvars -except Satellitendaten

Phase = 'Modellschritt';

Modellschritt = 0;

%% Einlesen der Satellitendaten
display(sprintf('\n<strong>Einlesen der Satellitendaten</strong>\n'))

%   Einlesen der Satellitendaten falls nicht vorhanden
if ~exist('Satellitendaten','var')
    Dateien = {'114255.obs','114255.nav'};
    Satellitendaten = einlesen_Satellitendaten(Dateien);
end

%   Auswählen eines Datensatzes
Datensatznummer = 1;
Datensatz = Satellitendaten(Datensatznummer);

%   Satelliten, deren Nachrichten empfangen wurden
Satelliten = Datensatz.Satelliten;

%   Auswählen der Satelliten
Satellitenauswahl = [1 2 3];

%   Auslesen der ausgewählten Satellitendaten
tE = Datensatz.Empfangszeiten(Satellitenauswahl);
tS = Datensatz.Sendezeiten(Satellitenauswahl);
Ephemeriden = Datensatz.Ephemeriden(:,Satellitenauswahl);

%   Anzeigen der Ergebnisse
format shortG
display(Datensatz,sprintf('Datensatz Nummer %d',Datensatznummer))
display(Satelliten, ...
    'Satelliten, deren Nachrichten empfangen wurden [Satelliten]')
display(Satellitenauswahl, ...
    'Nummern der ausgewählten Satelliten [Satellitenauswahl]')
format bank
display(tE, ...
    'Empfangszeiten der Nachrichten in Sekunden seit Wochenbeginn [tE]')
display(tS, ...
    'Sendezeiten der Nachrichten in Sekunden seit Wochenbeginn [tS]')
display(sprintf(['\nDie Positionsdaten der Satelliten werden nicht ' ...
    'angezeigt [Ephemeriden]\n']))
clearvars Dateien Datensatznummer

%% Schritt 1  |  Berechnen der Satellitenkoordinaten
display(sprintf(['\n<strong>Schritt 1  |  Berechnen der ' ...
    'Satellitenkoordinaten</strong>\n']))
```

```
%   Berechnen der Satellitenkoordinaten
if Modellschritt == 1
    [xS,yS,zS] = berechne_Satellitenkoordinaten(NaN,Ephemeriden);
else
    [xS,yS,zS] = berechne_Satellitenkoordinaten(tS,Ephemeriden);
end

%   Überprüfen der Satellitenkoordinaten
ueberpruefe_Loesung(Phase,'Schritt 1', ...
    xS,yS,zS,Datensatz,Satellitenauswahl);

%   Anzeigen der Ergebnisse
format bank
display([xS(1) yS(1) zS(1)], ...
    'Koordinaten des ersten Satelliten in Meter [xS(1) yS(1) zS(1)]')
display([xS(2) yS(2) zS(2)], ...
    'Koordinaten des zweiten Satelliten in Meter [xS(2) yS(2) zS(2)]')
display([xS(3) yS(3) zS(3)], ...
    'Koordinaten des dritten Satelliten in Meter [xS(3) yS(3) zS(3)]')

%% Schritt 2  |  Berechnen der Pseudoentfernungen
display(sprintf(['\n<strong>Schritt 2  |  Berechnen der ' ...
    'Pseudoentfernungen</strong>\n']))

%   Aufstellen der Berechnungsformel
if Modellschritt == 2
    berechne_Pseudoentfernungen = @(tS,tE) NaN;
else
    berechne_Pseudoentfernungen = @(tS,tE) (tE-tS)*299792458;
end

%   Berechnen der Pseudoentfernungen
dS = berechne_Pseudoentfernungen(tS,tE);

%   Überprüfen der Pseudoentfernungen
ueberpruefe_Loesung(Phase,'Schritt 2',dS,Datensatz,Satellitenauswahl);

%   Anzeigen der Ergebnisse
format bank
display([dS(1) dS(2) dS(3)], ...
    'Pseudoentfernungen zu den Satelliten in Meter [dS(1) dS(2) dS(3)]')

%% Schritt 3  |  Berechnen der Empfängerkoordinaten
display(sprintf(['\n<strong>Schritt 3  |  Berechnen der ' ...
    'Empfängerkoordinaten</strong>\n']))

%   Aufstellen des Gleichungssystems
if Modellschritt == 3
    Gleichungssystem = @(xE,yE,zE) [NaN
                                    NaN
                                    NaN];
else
    Gleichungssystem = @(xE,yE,zE) ...
       [sqrt((xS(1)-xE)^2+(yS(1)-yE)^2+(zS(1)-zE)^2)-dS(1)
        sqrt((xS(2)-xE)^2+(yS(2)-yE)^2+(zS(2)-zE)^2)-dS(2)
        sqrt((xS(3)-xE)^2+(yS(3)-yE)^2+(zS(3)-zE)^2)-dS(3)];
end

%   Überprüfen des Gleichungssystems
ueberpruefe_Loesung(Phase,'Schritt 3',Gleichungssystem,xS,yS,zS,dS);
```

```
%   Definiere Funktion und Startpunkt für fsolve
Funktion = @(X) Gleichungssystem(X(1),X(2),X(3));
Startpunkt = [0 0 0];

%   Berechnen einer Lösung in der Nähe des Startpunktes
X = fsolve(Funktion,Startpunkt);

%   Speichern der berechneten Lösung
xE = X(1);
yE = X(2);
zE = X(3);

%   Überprüfen der Empfängerkoordinaten
ueberpruefe_Loesung(Phase,'Schritt 3',xE,yE,zE,xS,yS,zS,dS);

%   Anzeigen der Ergebnisse
format bank
display([xE yE zE],'Koordinaten des Empfängers in Meter [xE yE zE]')
clearvars Funktion Startpunkt X

%% Schritt 4 | Umrechnen in geographische Höhe, Breite und Länge
display(sprintf(['\n<strong>Schritt 4 | Umrechnen in ' ...
    'geographische Höhe, Breite und Länge</strong>\n']))

%   Aufstellen der Umrechnungsformeln
if Modellschritt == 4
    berechne_h      = @(xE,yE,zE) NaN;
    berechne_phi    = @(xE,yE,zE) NaN;
    berechne_lambda = @(xE,yE,zE) NaN;
else
    berechne_h      = @(xE,yE,zE) sqrt(xE^2+yE^2+zE^2)-6378137.0;
    berechne_phi    = @(xE,yE,zE) atan(zE/sqrt(xE^2+yE^2))*180/pi;
    berechne_lambda = @(xE,yE,zE) atan2(yE,xE)*180/pi;
end

%   Berechnen der geographischen Koordinaten
h      = berechne_h(xE,yE,zE);
phi    = berechne_phi(xE,yE,zE);
lambda = berechne_lambda(xE,yE,zE);

%   Überprüfen der geographischen Koordinaten
ueberpruefe_Loesung(Phase,'Schritt 4',h,xE,yE,zE);
ueberpruefe_Loesung(Phase,'Schritt 4',phi,xE,yE,zE);
ueberpruefe_Loesung(Phase,'Schritt 4',lambda,xE,yE,zE);

%   Anzeigen der Ergebnisse
format shortG
display(h,'Geographische Höhe des Empfängers in Meter [h]')
display([phi lambda], ...
    'Geographische Breite und Länge des Empfängers in Grad [phi lambda]')

%% Anzeigen der Empfängerposition
display(sprintf('\n<strong>Anzeigen der Empfängerposition</strong>\n'))

%   Auswählen des Kartenausschnitts
Kartenausschnitt_manuell = true;
Zentrum = '50.5,6.8';
Zoom = '8';

%   Erzeugen der URL
```

```
url = sprintf('https://www.google.de/maps/place/%f,%f',phi,lambda);
if Kartenausschnitt_manuell
    url = [url '/@' Zentrum ',' Zoom 'z'];
end

%   Anzeigen der URL im Browser
web(url,'-browser')
display(url,'URL für die Anzeige im Webbrowser')
clearvars Kartenausschnitt_manuell Zentrum Zoom url
```

A.7. Die Lösung der Modellverbesserungen gps_solution.m

```
%   Wie funktioniert eigentlich... GPS
%       ... uns was hat das mit Mathe zu tun?

%   CAMMP - COMPUTATIONAL AND MATHEMATICAL MODELING PROGRAM
%   RWTH AACHEN UNIVERSITY 2015

%   Starte das Skript durch Eingabe von >> gps << im Command Window

clc
format
clearvars -except Satellitendaten

Modellverbesserung = [1 2 3 4 5];

%% Einlesen der Satellitendaten
display(sprintf('\n<strong>Einlesen der Satellitendaten</strong>\n'))

%   Einlesen der Satellitendaten falls nicht vorhanden
if ~exist('Satellitendaten','var')
    Dateien = {'114255.obs','114255.nav'};
    Satellitendaten = einlesen_Satellitendaten(Dateien);
end

%   Auswählen eines Datensatzes
Datensatznummer = 1;
Datensatz = Satellitendaten(Datensatznummer);

%   Satelliten, deren Nachrichten empfangen wurden
Satelliten = Datensatz.Satelliten;

%   Auswählen der Satelliten
if any(Modellverbesserung == 4)
    Satellitenauswahl = Satelliten;
elseif any(Modellverbesserung == 2)
    Satellitenauswahl = [1 2 3 4];
else
    Satellitenauswahl = [1 2 3];
end

%   Auslesen der ausgewählten Satellitendaten
tE = Datensatz.Empfangszeiten(Satellitenauswahl);
tS = Datensatz.Sendezeiten(Satellitenauswahl);
if any(Modellverbesserung == 5)
    G = Datensatz.Signalstaerken(Satellitenauswahl);
end
Ephemeriden = Datensatz.Ephemeriden(:,Satellitenauswahl);

%   Anzeigen der Ergebnisse
format shortG
display(Datensatz,sprintf('Datensatz Nummer %d',Datensatznummer))
display(Satelliten, ...
    'Satelliten, deren Nachrichten empfangen wurden [Satelliten]')
display(Satellitenauswahl, ...
    'Nummern der ausgewählten Satelliten [Satellitenauswahl]')
format bank
display([tE.' tS.'], ...
   ['Empfangszeiten und Sendezeiten der Nachrichten in Sekunden '...
    'seit Wochenbeginn [tE tS]'])
if any(Modellverbesserung == 5)
```

```
    format shortG
    display(G,'Signalstärken der ausgewählten Nachrichten in dBHz [G]')
end
display(sprintf(['\nDie Positionsdaten der Satelliten werden nicht ' ...
    'angezeigt [Ephemeriden]\n']))
clearvars Dateien Datensatznummer

%% Schritt 1 | Berechnen der Satellitenkoordinaten
display(sprintf(['\n<strong>Schritt 1 | Berechnen der ' ...
    'Satellitenkoordinaten</strong>\n']))

if any(Modellverbesserung == 1)
    %  Berechnung der Satellitenuhrfehler
    Delta_tS = berechne_Satellitenuhrfehler(tS,Ephemeriden);
    Delta_tS = berechne_Satellitenuhrfehler(tS-Delta_tS,Ephemeriden);
end

%  Berechnen der Satellitenkoordinaten
if any(Modellverbesserung == 1)
    [xS,yS,zS] = berechne_Satellitenkoordinaten(tS-Delta_tS,Ephemeriden);
else
    [xS,yS,zS] = berechne_Satellitenkoordinaten(tS,Ephemeriden);
end

%  Anzeigen der Ergebnisse
if any(Modellverbesserung == 1)
    format shortG
    display(Delta_tS.', ...
        'Fehler der Satellitenuhren in Sekunden [Delta_tS]')
end
format bank
display([xS.' yS.' zS.'], ...
    'Koordinaten der Satelliten in Meter [xS yS zS]')

%% Schritt 2 | Berechnen der Pseudoentfernungen
display(sprintf(['\n<strong>Schritt 2 | Berechnen der ' ...
    'Pseudoentfernungen</strong>\n']))

%  Aufstellen der Berechnungsformel
berechne_Pseudoentfernungen = @(tS,tE) (tE-tS)*299792458;

%  Berechnen der Pseudoentfernungen
if any(Modellverbesserung == 1)
    dS = berechne_Pseudoentfernungen(tS-Delta_tS,tE);
else
    dS = berechne_Pseudoentfernungen(tS,tE);
end

%  Anzeigen der Ergebnisse
format bank
display(dS.','Pseudoentfernungen zu den Satelliten in Meter [dS]')

%% Schritt 3 | Berechnen der Empfängerkoordinaten
display(sprintf(['\n<strong>Schritt 3 | Berechnen der ' ...
    'Empfängerkoordinaten</strong>\n']))

%  Aufstellen des Gleichungssystems
if any(Modellverbesserung == 5) && any(Modellverbesserung == 2)
    Gleichungssystem = @(xE,yE,zE,Delta_dS) ...
        sqrt(G).*(sqrt((xS-xE).^2+(yS-yE).^2+(zS-zE).^2)-(dS-Delta_dS));
elseif any(Modellverbesserung == 5)
```

```
    Gleichungssystem = @(xE,yE,zE) ...
        sqrt(G).*(sqrt((xS-xE).^2+(yS-yE).^2+(zS-zE).^2)-dS);
elseif any(Modellverbesserung == 4) && any(Modellverbesserung == 2)
    Gleichungssystem = @(xE,yE,zE,Delta_dS) ...
        sqrt((xS-xE).^2+(yS-yE).^2+(zS-zE).^2)-(dS-Delta_dS);
elseif any(Modellverbesserung == 4)
    Gleichungssystem = @(xE,yE,zE) ...
        sqrt((xS-xE).^2+(yS-yE).^2+(zS-zE).^2)-dS;
elseif any(Modellverbesserung == 2)
    Gleichungssystem = @(xE,yE,zE,Delta_dS) ...
       [sqrt((xS(1)-xE)^2+(yS(1)-yE)^2+(zS(1)-zE)^2)-(dS(1)-Delta_dS)
        sqrt((xS(2)-xE)^2+(yS(2)-yE)^2+(zS(2)-zE)^2)-(dS(2)-Delta_dS)
        sqrt((xS(3)-xE)^2+(yS(3)-yE)^2+(zS(3)-zE)^2)-(dS(3)-Delta_dS)
        sqrt((xS(4)-xE)^2+(yS(4)-yE)^2+(zS(4)-zE)^2)-(dS(4)-Delta_dS)];
else
    Gleichungssystem = @(xE,yE,zE) ...
       [sqrt((xS(1)-xE)^2+(yS(1)-yE)^2+(zS(1)-zE)^2)-dS(1)
        sqrt((xS(2)-xE)^2+(yS(2)-yE)^2+(zS(2)-zE)^2)-dS(2)
        sqrt((xS(3)-xE)^2+(yS(3)-yE)^2+(zS(3)-zE)^2)-dS(3)];
end

%   Definiere Funktion und Startpunkt für fsolve
if any(Modellverbesserung == 2)
    Funktion = @(X) Gleichungssystem(X(1),X(2),X(3),X(4));
    Startpunkt = [0 0 0 0];
else
    Funktion = @(X) Gleichungssystem(X(1),X(2),X(3));
    Startpunkt = [0 0 0];
end

%   Berechnen einer Lösung in der Nähe des Startpunktes
if any(Modellverbesserung == 4) || any(Modellverbesserung == 5)
    X = lsqnonlin(Funktion,Startpunkt);
else
    X = fsolve(Funktion,Startpunkt);
end

%   Speichern der berechneten Lösung
xE = X(1);
yE = X(2);
zE = X(3);
if any(Modellverbesserung == 2)
    Delta_tE = X(4)/299792458;
end

%   Anzeigen der Ergebnisse
if any(Modellverbesserung == 2)
    format shortG
    display(Delta_tE,'Fehler der Empfängeruhr in Sekunden [Delta_tE]')
end
format bank
display([xE yE zE],'Koordinaten des Empfängers in Meter [xE yE zE]')
clearvars Funktion Startpunkt X

%% Schritt 4 | Umrechnen in geographische Höhe, Breite und Länge
display(sprintf(['\n<strong>Schritt 4 | Umrechnen in ' ...
    'geographische Höhe, Breite und Länge</strong>\n']))

%   Aufstellen der Umrechnungsformeln
berechne_h      = @(xE,yE,zE) sqrt(xE^2+yE^2+zE^2)-6378137.0;
berechne_phi    = @(xE,yE,zE) atan(zE/sqrt(xE^2+yE^2))*180/pi;
```

```
berechne_lambda = @(xE,yE,zE) atan2(yE,xE)*180/pi;

%   Berechnen der geographischen Koordinaten
h      = berechne_h(xE,yE,zE);
phi    = berechne_phi(xE,yE,zE);
lambda = berechne_lambda(xE,yE,zE);

if any(Modellverbesserung == 3)
    %   Parameter des Referenzellipsoiden WGS 84
    a = 6378137.0;
    b = 6356752.3142;
    epsilon = sqrt(1-(b/a)^2);

    %   Aufstellen der Gleichungen
    Gleichungen = @(h,phi,lambda) ...
        [(a/sqrt(1-epsilon^2*(sin(phi)^2))+h)*cos(phi)*cos(lambda)-xE
         (a/sqrt(1-epsilon^2*(sin(phi)^2))+h)*cos(phi)*sin(lambda)-yE
         (a*(1-epsilon^2)/sqrt(1-epsilon^2*(sin(phi)^2))+h)*sin(phi)-zE];

    %   Definiere Funktion und Startpunkt für fsolve
    Funktion = @(X) Gleichungen(X(1),X(2),X(3));
    Startpunkt = [h phi*pi/180 lambda*pi/180];

    %   Berechnen einer Lösung in der Nähe des Startpunktes
    X = fsolve(Funktion,Startpunkt);

    %   Speichern der berechneten Lösung
    h      = X(1);
    phi    = X(2)*180/pi;
    lambda = X(3)*180/pi;
end

%   Anzeigen der Ergebnisse
format shortG
display(h,'Geographische Höhe des Empfängers in Meter [h]')
display([phi,lambda], ...
    'Geographische Breite und Länge des Empfängers in Grad [phi lambda]')
clearvars Funktion Startpunkt X

%% Anzeigen der Empfängerposition
display(sprintf('\n<strong>Anzeigen der Empfängerposition</strong>\n'))

%   Auswählen des Kartenausschnitts
Kartenausschnitt_manuell = true;
Zentrum = '50.5,6.8';
Zoom = '8';

%   Erzeugen der URL
url = sprintf('https://www.google.de/maps/place/%f,%f',phi,lambda);
if Kartenausschnitt_manuell
    url = [url '/@' Zentrum ',' Zoom 'z'];
end

%   Anzeigen der URL im Browser
web(url,'-browser')
display(url,'URL für die Anzeige im Webbrowser')
clearvars Kartenausschnitt_manuell Zentrum Zoom url
```

B. Arbeitsblätter, Hilfekarten und Lösungen

B.1. Arbeitsblatt zu den Modellschritten

CAMMP Day

Wie funktioniert eigentlich GPS und was hat das mit Mathe zu tun?

CAMMP | RWTH AACHEN UNIVERSITY

Modell zur Positionsbestimmung

GPS ist ein globales Navigationssatellitensystem zur Positionsbestimmung welches mittlerweile in vielen Bereichen eingesetzt wird. Sei es im Navi, um von A nach B zu kommen oder bei einer Handy App, die einem sagt, wann der nächste Bus kommt. Hinter allen Anwendungen steckt die aktuelle Positionsbestimmung. Aber wie funktioniert das?

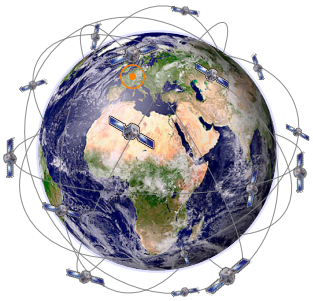


Bild 1: GPS Satelliten umkreisen die Erde.

Die Idee hinter GPS

Mehrere Satelliten umkreisen in ca. 20 000 km Höhe die Erde und senden jede Millisekunde eine Nachricht mit ihrer aktuellen Zeit. Die Satelliten verfügen über eine eingebaute Atomuhr. Da die Nachricht mit Lichtgeschwindigkeit (299792458 $\frac{m}{s}$) geschickt wird, werden zwischen 65 und 90 Millisekunden benötigt, um auf der Erde empfangen zu werden. Die Übertragungsdauer hängt davon ab, wie weit der Empfänger von dem Satelliten entfernt ist.

Durch Vergleich der Sendezeit mit der Empfangszeit kann mit Hilfe der bekannten Geschwindigkeit der Nachricht die sogenannte *Pseudoentfernung* zu dem Satelliten berechnet werden. Mit drei Satelliten lässt sich die Position des Empfängers auf der Erde bestimmen.

Problembeschreibung | Bestimmen der Empfängerposition

Es liegen selbst empfangene Satellitendaten vor, zu denen ihr die Position auf der Erde bestimmen sollt. Öffnet nun das MATLAB-Script *gps.m* mit einem Doppelklick und startet es durch durch einen Klick auf den *Run*-Button oder durch Eingabe von `gps` im *Command Window* (siehe Kasten am Ende der Seite). Alle Ausgaben erscheinen im *Command Window*.

Bearbeitet die folgenden Schritte:

Schritt 1 | Berechnen der Satellitenkoordinaten

Die Bahnkurven der Satelliten sind bekannt durch die sogenannten *Ephemeriden*. Mit der Funktion `berechne_Satellitenkoordinaten(t,Ephemeriden)` lassen sich durch Angabe eines Zeitpunktes t die Positionen der Satelliten zu dem gewünschten Zeitpunkt berechnen.
Verbessert die Zeile

```
%   Berechnen der Satellitenkoordinaten
[xS,yS,zS] = berechne_Satellitenkoordinaten(NaN,Ephemeriden);
```

so, dass die Koordinaten der drei ausgewählten Satelliten zu den richtigen Zeiten berechnet werden. Ersetzt dazu das `NaN` (*„Not a Number“*) durch die richtigen Zeiten.

Die Koordinaten der Satelliten werden in den Variablen `xS`, `yS` und `zS` gespeichert.

Ausführen des MATLAB-Skripts *gps.m*

Um das geöffnete Script zu starten, müsst ihr auf den *Run*-Button klicken oder im *Command Window* den Befehl `gps` eintippen. Vor der Eingabe von `gps` solltet ihr eure Änderungen am Code mit dem *Save*-Button abspeichern. Beim Klick auf den *Run*-Button wird automatisch gespeichert. Ihr könnt das Skript auch abschnittsweise ausführen, indem ihr auf den *Run Section*-Button oder auf den *Run and Advance*-Button klickt. Zum nächsten Abschnitt kommt ihr dann mit dem *Advance*-Button oder einfach durch einen Klick ins Editor-Fenster. In dem *Workspace* befinden sich danach die Variablen. Ihr könnt sie euch jederzeit mit einem Doppelklick ansehen oder im *Command Window* ausgeben lassen.

1/2

Schritt 2 | Berechnen der Pseudoentfernungen

Bestimmt die Pseudoentfernungen der Satelliten zum Empfänger auf der Erde. Vervollständigt dazu die Funktion *berechne_Pseudoentfernungen*. Ersetzt in der Zeile

```
%   Aufstellen der Berechnungsformel
berechne_Pseudoentfernungen = @(tS,tE) NaN;
```

das `NaN` durch die richtige Formel.

Hinweis: Bei einer Formel müssen alle Variablen hinter dem @-Zeichen aufgeführt werden. Alle Variablen, die nicht hinter dem @-Zeichen aufgeführt sind, werden als Konstanten betrachtet. Eine korrekt eingegebene Formel sieht z. B. so aus:

```
berechne_Kreiskegelvolumen = @(r,h) 1/3*pi*r^2*h;
```

Schritt 3 | Berechnen der Empfängerkoordinaten

Nun ist also nicht nur die Position der Satelliten im Weltall, sondern auch ihre Entfernung zum Empfänger auf der Erde bekannt. Für die drei unbekannten Empfängerkoordinaten x_E, y_E und z_E lassen sich drei Gleichungen aufstellen.

Überlegt euch auf dem Papier zunächst eine Gleichung für den ersten Satelliten. Sie sollte die bekannten Satellitenkoordinaten $x_{S(1)}$, $y_{S(1)}$ und $z_{S(1)}$, die bekannte Pseudoentfernung $d_{S(1)}$ und die unbekannten Empfängerkoordinaten x_E, y_E und z_E enthalten. Stellt danach Gleichungen für den zweiten und dritten Satelliten analog auf.

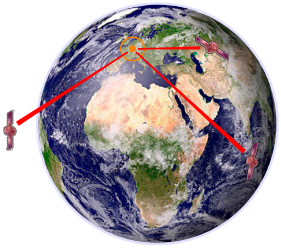


Bild 2: Entfernung der Satelliten vom Empfänger.

Das Gleichungssystem lassen wir automatisch von MATLAB lösen, indem wir es als Nullstellenproblem formulieren.Formt dazu jede einzelne Gleichung so um, dass hinter dem Gleichheitszeichen eine Null steht. Gebt das Gleichungssystem ohne Gleichheitszeichen und Nullen anstelle der drei `NaN` in folgender Zeile ein:

```
%   Aufstellen des Gleichungssystems
Gleichungssystem = @(xE,yE,zE) [NaN
                                NaN
                                NaN];
```

Beispiel: Das Gleichungssystem $\{x^2 - y - 5 = 0,\ x - \sqrt{y} - 1 = 0\}$ muss so eingegeben werden:

```
Gleichungssystem = @(x,y) [x^2-y-5
                           x-sqrt(y)-1];
```

Schritt 4 | Umrechnen in geographische Höhe, Breite und Länge

Um sich die Position des Empfängers auf einer Karte anzeigen zu lassen, müssen die Koordinaten x_E, y_E und z_E in geographische Breite ϕ (*phi*) und Länge λ (*lambda*) umgerechnet werden.

Modelliert dazu die Erde durch eine Kugel und überlegt euch die Formeln für die Höhe h über der Erdoberfläche sowie für die geographische Breite ϕ und Länge λ. Ersetzt die `NaN` in den Umrechnungsformeln durch die richtigen Formeln.

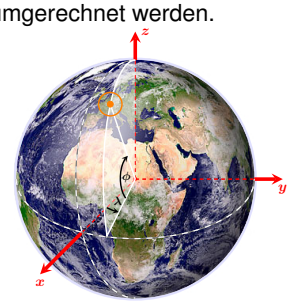


Bild 3: geographische Breite und Länge

```
%   Aufstellen der Umrechnungsformeln
berechne_h      = @(xE,yE,zE) NaN;
berechne_phi    = @(xE,yE,zE) NaN;
berechne_lambda = @(xE,yE,zE) NaN;
```

Hinweis: Die beiden Winkel ϕ und λ müssen in der Einheit Grad bestimmt werden.

CAMMP Day
Wie funktioniert eigentlich GPS und was hat das mit Mathe zu tun?

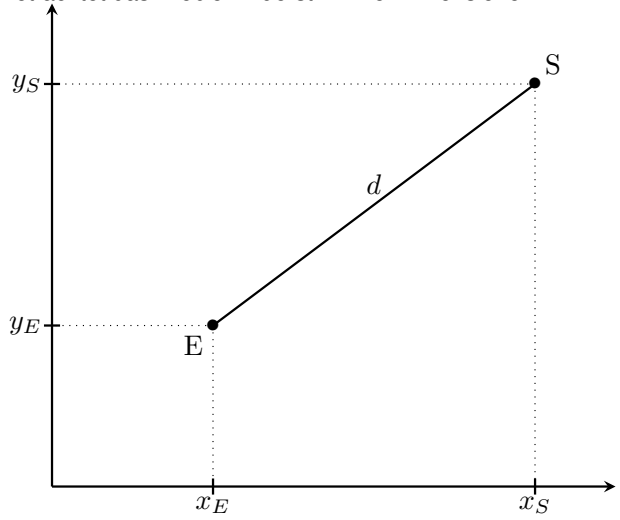
CAMMP | RWTH AACHEN UNIVERSITY

Hilfekarten zu den Modellschritten

Schritt 3 | Berechnen der Empfängerkoordinaten | Hilfekarte 1

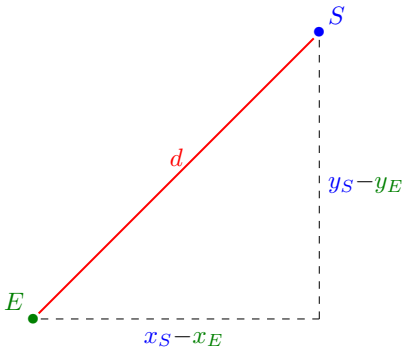
Aufstellung von Distanzgleichungen vom Satelliten zum unbekannten Standort des Empfängers.

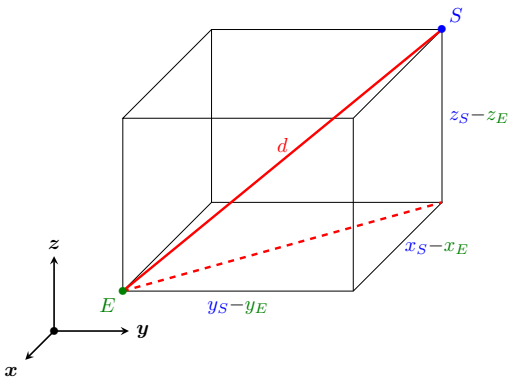
- Betrachtet das Problem zuerst in zwei Dimensionen:



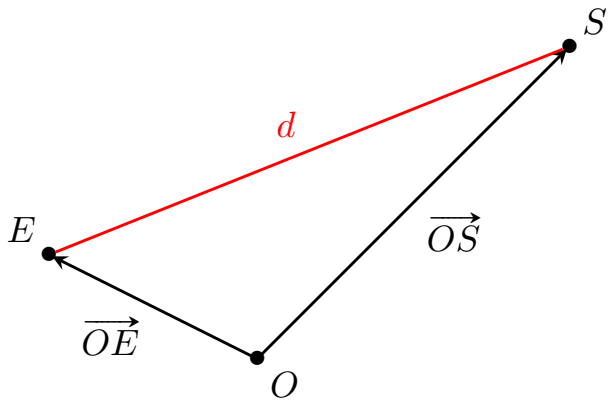
- Stellt einen Zusammenhang zwischen den Empfängerkoordinaten (x_E, y_E), den Satellitenkoordinaten (x_S, y_S) und der Pseudoentfernung d her.
- Verallgemeinert eurer Ergebnis danach auf drei Dimensionen.

Schritt 3 | Berechnen der Empfängerkoordinaten | Hilfekarte 2





Schritt 3 | Berechnen der Empfängerkoordinaten | Hilfekarte 1 (Vektoren)



Schritt 3 | Berechnen der Empfängerkoordinaten | Hilfekarte 2 (Vektoren)

Die Gleichung

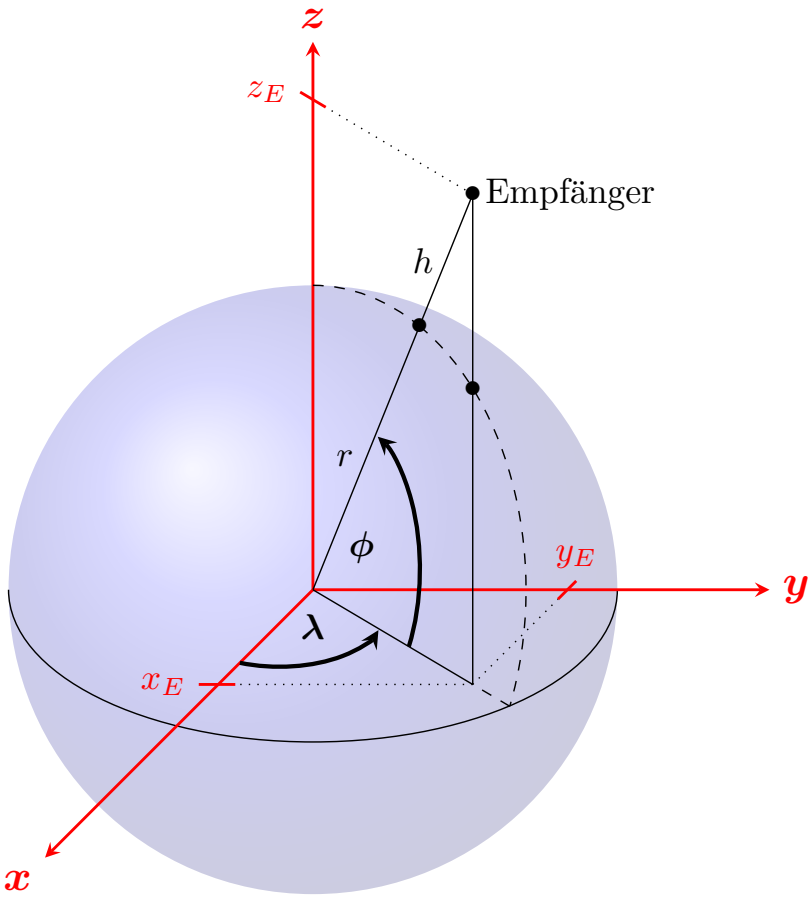
$$\left|\overrightarrow{OS} - \overrightarrow{OE}\right| = d$$

lautet für den ersten Satelliten wie folgt:

$$\left|\begin{pmatrix} x_{S(1)} \\ y_{S(1)} \\ z_{S(1)} \end{pmatrix} - \begin{pmatrix} x_E \\ y_E \\ z_E \end{pmatrix}\right| = d_{S(1)}$$

Der Betrag eines Vektors kann in MATLAB mit den Befehl `norm` berechnet werden.

Schritt 4 | Umrechnen in geographische Breite, Länge und Höhe | Hilfekarte 1



Schritt 4 | Umrechnung von Bogenmaß in Grad | Hilfekarte 2

Der Vollwinkel hat 2π Radiant im Bogenmaß oder 360 Grad:

$$2\pi \text{ rad} = 360^\circ$$

Somit gilt:

$$1 \text{ rad} = \frac{360^\circ}{2\pi} = \frac{180^\circ}{\pi}$$

CAMMP Day
Wie funktioniert eigentlich GPS und was hat das mit Mathe zu tun?

CAMMP | RWTH AACHEN UNIVERSITY

Musterlösung zu den Modellschritten

Schritt 1 | Berechnen der Satellitenkoordinaten

Die Satellitenkoordinaten müssen zu den Sendezeiten t_S der Nachrichten berechnet werden:

```
[xS,yS,zS] = berechne_Satellitenkoordinaten(tS,Ephemeriden);
```

Hier gibt es keine Hilfekarte

Schritt 2 | Berechnen der Pseudoentfernungen

Die Nachrichten wurden zu den Zeitpunkten t_S von den Satelliten abgeschickt und zu den Zeitpunkten t_E auf der Erde empfangen. Insgesamt waren die Nachrichten damit $t_E - t_S$ Sekunden unterwegs. Da die Nachrichten mit Lichtgeschwindigkeit (299792458 m/s) gesendet wurden, müssen die Satelliten $(t_E - t_S) \cdot 299792458$ Meter entfernt sein:

```
berechne_Pseudoentfernungen = @(tS,tE) (tE-tS)*299792458;
```

Hier gibt es keine Hilfekarte

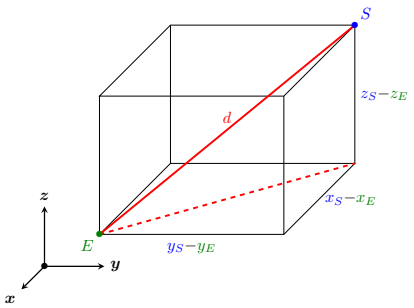
Schritt 3 | Berechnen der Empfängerkoordinaten

Mit Schritt 1 und 2 sind die Koordinaten der Satelliten x_S, y_S und z_S und ihre Entfernung d_S zu unserem Standort bekannt. Mittels des Satzes von Pythagoras lässt sich anhand der Koordinaten die Distanz zwischen unserem Standort (x_E, y_E, z_E) und dem Satelliten bestimmen, die ja gerade der Pseudoentfernung d_S entsprechen muss. Damit ergeben sich für drei Satelliten drei Gleichungen:

$$\sqrt{(x_{S(1)} - x_E)^2 + (y_{S(1)} - y_E)^2 + (z_{S(1)} - z_E)^2} = d_{S(1)}$$
$$\sqrt{(x_{S(2)} - x_E)^2 + (y_{S(2)} - y_E)^2 + (z_{S(2)} - z_E)^2} = d_{S(2)}$$
$$\sqrt{(x_{S(3)} - x_E)^2 + (y_{S(3)} - y_E)^2 + (z_{S(3)} - z_E)^2} = d_{S(3)}$$

Dieses Gleichungssystem muss noch als Nullstellenproblem ausgedrückt werden, indem einfach die rechte Seite von der linken Seite subtrahiert wird:

```
Gleichungssystem = @(xE,yE,zE) ...
   [sqrt((xS(1)-xE)^2+(yS(1)-yE)^2+(zS(1)-zE)^2)-dS(1)
    sqrt((xS(2)-xE)^2+(yS(2)-yE)^2+(zS(2)-zE)^2)-dS(2)
    sqrt((xS(3)-xE)^2+(yS(3)-yE)^2+(zS(3)-zE)^2)-dS(3)];
```



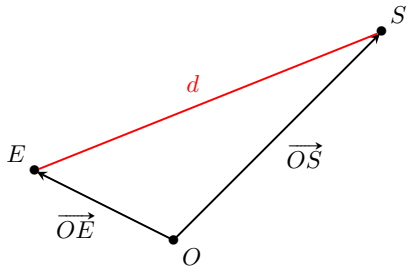
Hier gibt es zwei Hilfekarten. Die erste ist eine zweidimensionale Darstellung, die zweite zeigt die obige dreidimensionale Grafik.

Alternative Lösung (mit Vektoren)
Das Gleichungssystem kann auch vektoriell dargestellt werden:

$$\left|\begin{pmatrix} x_{S(1)} \\ y_{S(1)} \\ z_{S(1)} \end{pmatrix} - \begin{pmatrix} x_E \\ y_E \\ z_E \end{pmatrix}\right| = d_{S(1)}\ ,\ \left|\begin{pmatrix} x_{S(2)} \\ y_{S(2)} \\ z_{S(2)} \end{pmatrix} - \begin{pmatrix} x_E \\ y_E \\ z_E \end{pmatrix}\right| = d_{S(2)}\ ,\ \left|\begin{pmatrix} x_{S(3)} \\ y_{S(3)} \\ z_{S(3)} \end{pmatrix} - \begin{pmatrix} x_E \\ y_E \\ z_E \end{pmatrix}\right| = d_{S(3)}$$

Als MATLAB-Code:

```
Gleichungssystem = @(xE,yE,zE) ...
   [norm([xS(1) yS(1) zS(1)]-[xE yE zE])-dS(1)
    norm([xS(2) yS(2) zS(2)]-[xE yE zE])-dS(2)
    norm([xS(3) yS(3) zS(3)]-[xE yE zE])-dS(3)];
```



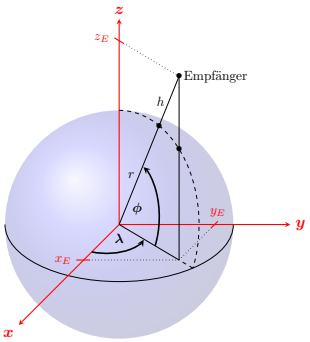
Auch für diese alternative Lösung mit Vektoren gibt es zwei Hilfekarten.

Schritt 4 | Umrechnen in geographische Breite, Länge und Höhe

Um sich die Position des Empfängers auf einer Karte anzeigen zu lassen, müssen die Empfängerkoordinaten x_E, y_E und z_E in geographische Breite ϕ (*phi*) und Länge λ (*lambda*) umgerechnet werden.

Da die Erde als eine Kugel mit Radius r angenommen wird, lässt sich mittels trigonometrischer Funktionen der Breiten- und Längengrad, sowie mittels Pythagoras die Höhe bestimmen:

$$h = \sqrt{x_E^2 + y_E^2 + z_E^2} - r$$
$$\phi = \arctan \frac{z_E}{\sqrt{x_E^2 + y_E^2}}$$
$$\lambda = \arctan \frac{y_E}{x_E}$$



Damit ergibt sich mit dem Äquatorradius $r = 6378137.0\,m$ folgender MATLAB-Code:

```
berechne_h      = @(xE,yE,zE) sqrt(xE^2+yE^2+zE^2)-6378137.0;
berechne_phi    = @(xE,yE,zE) atan(zE/sqrt(xE^2+yE^2))*180/pi;
berechne_lambda = @(xE,yE,zE) atan(yE/xE)*180/pi;
```

Eine allgemeingültige Lösung für λ muss jedoch mit dem `atan2` berechnet werden:

```
berechne_lambda = @(xE,yE,zE) atan2(yE,xE)*180/pi;
```

Eine alternative Darstellung für `berechne_lambda` mittels Indikatorfunktionen, um `atan2` zu umgehen:

```
berechne_lambda = @(xE,yE,zE) atan(yE/xE) * 180/pi * (xE >= 0) ...
                            + (atan(yE/xE) + pi) * 180/pi * (xE < 0) * (yE >= 0) ...
                            + (atan(yE/xE) - pi) * 180/pi * (xE < 0) * (yE < 0);
```

Hier gibt es zwei Hilfekarten. Eine mit der dargestellten Grafik und eine mit der Umrechnung von Bogenmaß in Grad.

2/2

CAMMP Day

Wie funktioniert eigentlich GPS und was hat das mit Mathe zu tun?

CAMMP | RWTH AACHEN UNIVERSITY

Modellverbesserungen

Nach Bearbeitung aller vier Schritte habt ihr einen Standort des Empfängers auf der Karte bestimmt. Die berechnete Position ist jedoch von unserem Messort in Aachen viel zu weit entfernt. Ziel ist es, das Modell nun so zu verbessern, dass die Abweichung auf wenige Meter genau ist.
Verändert bitte zuerst die Zeile »*Phase = 'Modellschritt'*« am Anfang des Skriptes in »*Phase = 'Modellverbesserung'*«.

Es gibt die folgenden vier Modellverbesserungen:

Modellverbesserung 1 | Satellitenuhr

Die Atomuhren in den Satelliten sind zwar die genausten Uhren der Welt, aber sie befinden sich nicht in Ruhe auf der Erde sondern sie bewegen sich mit den Satelliten in einem geringen Schwerefeld hoch über der Erde. Nach der Relativitätstheorie vergeht deshalb die Zeit in den Satelliten schneller als die Zeit auf der Erde. Für eine genaue Positionsbestimmung ist es notwendig, diese Zeitfehler zu berücksichtigen.

Berücksichtigt den Zeitfehler der Satelliten in den vorherigen Modellierungsschritten. Die Funktion `berechne_Satellitenuhrfehler` aus der Funktionsdatei `berechne_Satellitenuhrfehler.m` bestimmt dabei den Zeitfehler der Satelliten zum Zeitpunkt t. Der Aufruf geschieht folgendermaßen:
`Delta_t = berechne_Satellitenuhrfehler(t,Ephemeriden)`

Modellverbesserung 2 | Empfängeruhr

Die Empfänger (beispielsweise Handys) verfügen meistens nicht über eine präzise Atomuhr, so dass die Zeitmessung im Empfänger nicht sehr genau ist. Der unbekannte Zeitfehler Δt_E (`Delta_tE`) der Empfängeruhr muss bei der Berechnung der Empfängerkoordinaten berücksichtigt werden. Verbessert dazu das Gleichungssystem im dritten Schritt.

Modellverbesserung 3 | Form der Erde

Die Erde hat nicht genau die Form einer Kugel, denn durch die Erdrotation ist sie an den Polen etwas zusammengedrückt und hat eher die Form einer Mandarine. Dieses muss man bei der Berechnung der geographischen Koordinaten berücksichtigen, wenn man genauere Breiten- und Längengrade erhalten will.

Eure Aufgabe ist es, die tatsächliche Form der Erde bei der Berechnung der geographischen Koordinaten zu berücksichtigen.
Hinweis: Verwendet den Referenzellipsoiden WGS 84. Sucht im Internet nach Gleichungen, die eine Umrechnung von geodätischen Koordinaten in das ECEF-System ermöglichen.

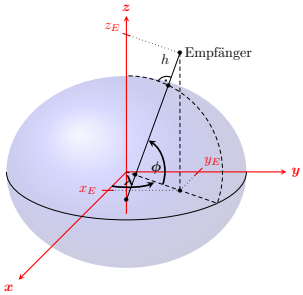


Bild 1: Die Erde als Rotationsellipsoid.

Modellverbesserung 4 | Verwendung aller Satelliten

Wenn ihr beim Auswählen der Satelliten vor dem ersten Schritt andere Satelliten auswählt (z. B. [2, 4, 7]), so werdet ihr feststellen, dass nun eine ganz andere Position auf der Erde bestimmt wird. Es stellt sich die Frage, welche Satellitensignale man nun verwenden soll, um seinen Standort zu bestimmen.

Eine Idee ist, alle Satellitensignale zu verwenden. Bei acht empfangenen Satellitensignalen bedeutet dies jedoch auch, dass man acht Gleichungen erhält. Damit gibt es also mehr Gleichungen als Unbekannte und das *überbestimmte Gleichungssystem* lässt sich somit nicht mehr exakt lösen.

Ersetzt *fsolve* durch einen geeigneteren MATLAB Befehl. Beachtet dazu die Warnung, die *fsolve*

anzeigt, wenn man mehr Gleichungen als Unbekannte hat.

Wenn ihr das geschafft habt, versucht auch noch das Gleichungssystem so umzuformen, dass ihr es flexibel für weniger oder mehr als acht Satelliten ohne Veränderungen benutzen könnt. Verwendet dazu die Vektorschreibweise von MATLAB.

Vergleichswerte für den Datensatz 114255 vom 18.07.2013

Dateien = {'*114255.obs*', '*114255.nav*'}
Datensatznummer = 1
Satellitenauswahl = [1 2 3] bzw. [1 2 3 4] (Modellverb. 2) bzw. [1 2 3 4 5 6 7 8] (Modellverb. 4)

Modellverbesserungen	Breite ϕ	Länge λ	Höhe h	
keine	50.343	10.297	-18 174 m	
1	50.970	7.2818	58 530 m	
2	49.199	6.7694	-233 660 m	
3	50.533	10.297	-5 472.8 m	
4	50.799	6.2933	262 160 m	
1 und 2	50.589	6.0769	-12 550 m	
1 und 3	51.156	7.2818	71 460 m	
1 und 4	50.639	6.2804	154 340 m	
2 und 3	49.397	6.7694	-221 370 m	
2 und 4	50.714	5.8972	-58 544 m	
3 und 4	50.979	6.2933	275 020 m	
1, 2 und 3	50.778	6.0769	241.43 m	
1, 2 und 4	50.590	6.0768	-12 511 m	
1, 3 und 4	50.822	6.2804	167 140 m	
2, 3 und 4	50.904	5.8972	-45 707 m	
1, 2, 3 und 4	50.778	6.0768	280.57 m	

Modellverbesserung 5 | Gewichtung von Gleichungen (Zusatzaufgabe)

Wir haben ursprünglich angenommen, dass die von den Satelliten ausgesendeten Signale mit Lichtgeschwindigkeit zur Erde übertragen werden. Nun ist es aber so, dass die Erdatmosphäre das Signal etwas abbremst. Wie stark ein Signal abgebremst wird hängt davon ab, durch wie viel Atmosphäre das Signal geschickt wird. Signale von Satelliten die direkt über uns stehen werden demnach weniger stark abgebremst als Signale von Satelliten die weiter weg sind.

Wenn ihr euch die Variable `Datensatz` anseht, bemerkt ihr dass nicht alle Signale mit der selben Signalstärke empfangen worden sind. Als einfaches Modell für die Atmosphärenkorrektur wollen wir den Signalen mit großer Signalstärke eine höhere Gewichtung bei der Ausgleichsrechnung geben als den Signalen mit kleiner Stärke.

Aufgabe: Jede Gleichung soll eine Gewichtung bekommt, welche proportional zur Signalstärke ist, und die Gewichtung soll als Faktor bei der Summenbildung berücksichtigt werden. Wenn z. B. das Gleichungssystem aus den Gleichungen $f_1(x) = 0$, $f_2(x) = 0$ und $f_3(x) = 0$ besteht und die die positiven Zahlen g_1, g_2 und g_3 die zugehörigen Gewichtungen sind, soll der Wert für x bestimmt werden für den die gewichtete Summe $g_1 \cdot f_1(x)^2 + g_2 \cdot f_2(x)^2 + g_3 \cdot f_3(x)^2$ minimal ist.

Hinweis: Beim Auslesen der ausgewählten Satellitendaten vor dem ersten Modellschritt müsst ihr noch die Signalstärken der ausgewählten Satelliten aus dem Datensatz kopieren.

Vergleichswerte für den Datensatz 114255 vom 18.07.2013

Dateien = {'*114255.obs*', '*114255.nav*'}
Datensatznummer = 1
Satellitenauswahl = [1 2 3 4 5 6 7 8]

Modellverbesserungen	Breite ϕ	Länge λ	Höhe h	
1, 2, 3 und 4	50.778262	6.076800	280.57 m	
1, 2, 3, 4 und Gewichtung	50.778254	6.076813	278.98 m	

CAMMP Day
Wie funktioniert eigentlich GPS und was hat das mit Mathe zu tun?

CAMMP | RWTH AACHEN UNIVERSITY

Hilfekarten zu Modellverbesserungen

Modellverbesserung 1 | Satellitenuhr | Hilfekarte 1

Die Satellitenuhrfehler betreffen sowohl die Satellitenkoordinaten als auch die Pseudoentfernungen.

Modellverbesserung 1 | Satellitenuhr | Hilfekarte 2

Da die Fehler der Satellitenuhren selber von den Sendezeitpunkten abhängen und da die Sendezeitpunkte fehlerbehaftet sind, müssten die Satellitenuhrfehler Δt_S eigentlich iterativ bestimmt werden. Ihr macht glücklicherweise keinen großen Fehler, wenn ihr dies nicht beachtet.

Modellverbesserung 1 | Satellitenuhr | Hilfekarte 3

t_S : Falsche Uhrzeit der Satelliten (bekannt)
t_S^* : Wahre Uhrzeit der Satelliten (unbekannt)
$\Delta t_S = f(t_S)$: Fehler der Satellitenuhren zur falschen Zeit t_S (bekannt, da f bekannte Funktion)
$\Delta t_S^* = f(t_S^*)$: Fehler der Satellitenuhren zur wahren Zeit t_S^* (unbekannt, da t_S^* unbekannt)

Zusammenhang:

$$t_S = t_S^* + \Delta t_S^*$$

Also gilt für die wahre Uhrzeit $t_S^* = t_S - \Delta t_S^*$. Leider ist Δt_S^* nicht bekannt, da t_S^* unbekannt ist. Δt_S^* müsste eigentlich iterativ bestimmt werden. Glücklicherweise ist aber $\Delta t_S^* \approx \Delta t_S$, und somit:

$$t_S^* \approx t_S - \Delta t_S$$

Modellverbesserung 2 | Empfängeruhr | Hilfekarte 1

Der Zeitunterschied Δt_E (`Delta_tE`) zwischen der ungenauen Empfängeruhrzeit und der tatsächlichen Uhrzeit ist eine vierte Unbekannte. Daher werden auch vier Satelliten benötigt, siehe dazu die Variable `Satellitenauswahl` beim Auswählen der Satelliten vor dem ersten Schritt.

Modellverbesserung 2 | Empfängeruhr | Hilfekarte 2

- Überlegt euch, welche Auswirkung es für die Pseudoentfernungen hat, wenn die Uhr im Empfänger um eine Sekunde vorgeht (das wäre ein Uhrfehler von +1 Sekunden).
- Verallgemeinert: Wie müssen die Pseudoentfernungen korrigiert werden, wenn der Fehler der Empfängeruhr t Sekunden beträgt?
- Überlegt: Wie muss das Gleichungssystem verändert werden, wenn der Fehler der Empfängeruhr unbekannt ist?

Hilfekarten zu den Modellverbesserungen – Seite 2

Modellverbesserung 3 | Form der Erde | Hilfekarte 1

Die Umrechnungsformeln für einen Rotationsellipsoiden lauten:

$$x_E = (N_\varphi + h) \cdot \cos(\varphi) \cdot \cos(\lambda)$$
$$y_E = (N_\varphi + h) \cdot \cos(\varphi) \cdot \sin(\lambda)$$
$$z_E = \left(N_\varphi(1-\varepsilon^2) + h\right) \cdot \sin(\varphi)$$

Dabei ist ε die *numerische Exzentrizität* und ergibt sich aus der großen Halbachse a und der kleinen Halbachse b des Ellipsoiden. N_φ ist der *Krümmungsradius des Ersten Vertikals* und ergibt sich aus der großen Halbache a, aus der numerischen Exzentrizität ε und aus der geographischen Breite φ.

$$\varepsilon = \sqrt{a^2 - b^2}/a$$
$$N_\varphi = a/\sqrt{1 - \varepsilon^2 \sin^2(\varphi)}$$

Für den Referenzellipsoiden WGS 84 ist $a = 6378137.0\,\text{m}$ und $b = 6356752.3142\,\text{m}$.

Modellverbesserung 3 | Form der Erde | Hilfekarte 2

Löst das Gleichungssystem für die drei Unbekannten h, φ und λ mit dem MATLAB-Befehl `fsolve`. Formt dazu euer Gleichungssystem in ein Nullstellenproblem um (wie bereits in Modellschritt 3). Achtet auf einen möglichst guten Startpunkt für `fsolve`.

Modellverbesserung 4 | Verwendung aller Satelliten | Hilfekarte 1

Wählt bei der Auswahl vor dem ersten Schritt alle empfangenen Satelliten aus. Informiert euch im Internet, wie man mit überbestimmten Gleichungssystemen umgeht, passt das Gleichungssystem an und ersetzt `fsolve` durch eine für dieses Gleichungssystem geeignetere MATLAB Routine.

Modellverbesserung 4 | Verwendung aller Satelliten | Hilfekarte 2

Die MATLAB-Routine `fslove` sucht einen Vektor X, sodass gilt:

$$f_1(X) = 0 \text{ und } f_2(X) = 0 \text{ und } \ldots \text{ und } f_n(X) = 0.$$

Die MATLAB-Routine `lsqnonlin` sucht einen Vektor X, sodass gilt:

$$f_1(X)^2 + f_2(X)^2 + \cdots + f_n(X)^2 = \text{minimal}$$

CAMMP Day
Wie funktioniert eigentlich GPS und was hat das mit Mathe zu tun?

CAMMP | RWTH AACHEN UNIVERSITY

Musterlösung zu den Modellverbesserungen

Modellverbesserung 1 | Satellitenuhr

Hintergrundinformation: Die Berechnung der Satellitenuhrfehler zum Zeitpunkt t in der Funktion „berechne_Satellitenuhrfehler“ geschieht über ein quadratisches Fehlermodell $a_{f0}+a_{f1}(t-t_{oe})+a_{f2}(t-t_{oe})^2+\Delta t_r$. Dabei werden die Konstanten a_{f0}, a_{f1} und a_{f2} und der Referenzzeitpunkt t_{oe} mit den Ephemeriden der Satelliten übertragen. Diese werden von Bodenstationen alle vier Stunden aktualisiert. Der zusätzliche Fehlerterm Δt_r hängt mit den Ellipsenbahnen der Satelliten zusammen. Die korrekte Sendezeit der Satelliten ist somit nicht t_S, sondern $t_S - \Delta t_S$, wobei Δt_S (`Delta_tS`) der Satellitenuhrfehler zur wahren Sendezeit der Nachrichten ist.

- Zunächst werden die Satellitenuhrfehler Δt_S zur wahren Sendezeit berechnet.

```
%   Berechnung der Satellitenuhrfehler
Delta_tS = berechne_Satellitenuhrfehler(tS,Ephemeriden);
Delta_tS = berechne_Satellitenuhrfehler(tS-Delta_tS,Ephemeriden);
```

- Anschließend werden die Satellitenkoordinaten zum korrigierten Zeitpunkt berechnet:

```
%   Berechnen der Satellitenkoordinaten
[xS,yS,zS] = berechne_Satellitenkoordinaten(tS-Delta_tS,Ephemeriden);
```

- Ebenso müssen für die Pseudoentfernungen die Satellitenuhrfehler berücksichtigt werden:

```
%   Berechnen der Pseudoentfernungen
dS = berechne_Pseudoentfernungen(tS-Delta_tS,tE);
```

Der gesamte MATLAB-Code lautet:

```
%%  Schritt 1   |   Berechnen der Satellitenkoordinaten

%   Berechnung der Satellitenuhrfehler
Delta_tS = berechne_Satellitenuhrfehler(tS,Ephemeriden);
Delta_tS = berechne_Satellitenuhrfehler(tS-Delta_tS,Ephemeriden);

%   Berechnen der Satellitenkoordinaten
[xS,yS,zS] = berechne_Satellitenkoordinaten(tS-Delta_tS,Ephemeriden);

%%  Schritt 2   |   Berechnen der Pseudoentfernungen

%   Aufstellen der Berechnungsformel
berechne_Pseudoentfernungen = @(tS,tE) (tE-tS)*299792458;

%   Berechnen der Pseudoentfernungen
dS = berechne_Pseudoentfernungen(tS-Delta_tS,tE);
```

Alternative Lösung: Da die Fehler der Satellitenuhren selber von den Sendezeitpunkten abhängen und da die Sendezeitpunkte fehlerbehaftet sind, müssen die Satellitenuhrfehler Δt_S iterativ berechnet werden. Es ist jedoch glücklicherweise auch ein Verzicht auf diese Iteration möglich, denn der dadurch gemachte Zeitfehler liegt nur in der Größenordnung von 10^{-14} Sekunden, was bei den Pseudoentfernungen zu einem vernachlässigbaren Fehler in der Größenordnung von 10^{-6} Meter führt. Der Fehler bei den Satellitenkoordinaten ist sogar noch kleiner. Somit genügt eine Zeile zur Berechnung der Fehler:

```
%   Berechnung der Satellitenuhrfehler
Delta_tS = berechne_Satellitenuhrfehler(tS,Ephemeriden);
```

Modellverbesserung 2 | Empfängeruhr

Der Fehler der Empfängeruhr ist Δt_E. Die korrekte Empfängeruhrzeit ist demnach nicht t_E, sondern $t_E - \Delta t_E$. Der Fehler der Empfängeruhr Δt_E verändert jede Pseudoentfernung um den selben Betrag $\Delta d_S = c\Delta t_E$, wobei c die Lichtgeschwindigkeit ist. Die korrigierten Pseudoentfernungen lauten demnach $d_S - \Delta d_S = d_S - c\Delta t_E$.
Im Gleichungssystem müssen nur die Pseudoentfernungen $d_{S(i)}$ durch die korrigierten Pseudoentfernungen $d_{S(i)} - c\Delta t_E$ ersetzt werden. Da nun vier Unbekannte x_E, y_E, z_E und Δt_E vorliegen, wird noch eine vierte Gleichung eines vierten Satelliten benötigt. Das korrigierte Gleichungssystem lautet somit:

$$\sqrt{(x_{S(1)} - xE)^2 + (y_{S(1)} - yE)^2 + (z_{S(1)} - zE)^2} - (d_{S(1)} - c\Delta t_E) = 0$$
$$\sqrt{(x_{S(2)} - xE)^2 + (y_{S(2)} - yE)^2 + (z_{S(2)} - zE)^2} - (d_{S(2)} - c\Delta t_E) = 0$$
$$\sqrt{(x_{S(3)} - xE)^2 + (y_{S(3)} - yE)^2 + (z_{S(3)} - zE)^2} - (d_{S(3)} - c\Delta t_E) = 0$$
$$\sqrt{(x_{S(4)} - xE)^2 + (y_{S(4)} - yE)^2 + (z_{S(4)} - zE)^2} - (d_{S(4)} - c\Delta t_E) = 0$$

Jedoch führt die Multiplikation mit der sehr großen Konstante c im Gleichungssystem zu einem deutlichen Genauigkeitsverlust bei der numerischen Lösung. Deshalb sollte das Gleichungssystem aus numerischen Gründen mit den vier Unbekannten x_E, y_E, z_E und Δd_S formuliert werden.

$$\sqrt{(x_{S(1)} - xE)^2 + (y_{S(1)} - yE)^2 + (z_{S(1)} - zE)^2} - (d_{S(1)} - \Delta d_S) = 0$$
$$\sqrt{(x_{S(2)} - xE)^2 + (y_{S(2)} - yE)^2 + (z_{S(2)} - zE)^2} - (d_{S(2)} - \Delta d_S) = 0$$
$$\sqrt{(x_{S(3)} - xE)^2 + (y_{S(3)} - yE)^2 + (z_{S(3)} - zE)^2} - (d_{S(3)} - \Delta d_S) = 0$$
$$\sqrt{(x_{S(4)} - xE)^2 + (y_{S(4)} - yE)^2 + (z_{S(4)} - zE)^2} - (d_{S(4)} - \Delta d_S) = 0$$

```
%%  Einlesen der Satellitendaten

%   Auswählen der Satelliten
Satelliten = Datensatz.Satelliten;
Satellitenauswahl = [1 2 3 4];

%%  Schritt 3  |  Berechnen der Empfängerkoordinaten

%   Aufstellen des Gleichungssystems
Gleichungssystem = @(xE,yE,zE,Delta_dS) ...
   [sqrt((xS(1)-xE)^2+(yS(1)-yE)^2+(zS(1)-zE)^2)-(dS(1)-Delta_dS)
    sqrt((xS(2)-xE)^2+(yS(2)-yE)^2+(zS(2)-zE)^2)-(dS(2)-Delta_dS)
    sqrt((xS(3)-xE)^2+(yS(3)-yE)^2+(zS(3)-zE)^2)-(dS(3)-Delta_dS)
    sqrt((xS(4)-xE)^2+(yS(4)-yE)^2+(zS(4)-zE)^2)-(dS(4)-Delta_dS)];

%   Berechnen der Empfängerkoordinaten
Funktion = @(X) Gleichungssystem(X(1),X(2),X(3),X(4));
Startpunkt = [0 0 0 0];
X = fsolve(Funktion,Startpunkt);
xE = X(1);
yE = X(2);
zE = X(3);
Delta_tE = X(4)/299792458;
```

Hinweis: Da der Fehler der Empfängeruhr Δt_E im Gegensatz zum Fehler der Satellitenuhren Δt_S unbekannt ist, müssen die Pseudoentfernungen im Gleichungssystem korrigiert werden und können nicht wie bei der Modellverbesserung 1 vorher korrigiert werden.

Lösungen zu den Modellverbesserungen – Seite 3

Modellverbesserung 3 | Form der Erde

Die SuS sollen die Formeln für die Berechnung der geographischen Koordinaten auf einem Ellipsoiden im Internet suchen (z. B. in Wikipedia unter „Referenzellipsoid“). Die Gleichungen eines Rotationsellipsoiden lauten:

$$\begin{aligned}(N_\phi + h) \cdot \cos(\phi) \cdot \cos(\lambda) &= x_E \\ (N_\phi + h) \cdot \cos(\phi) \cdot \sin(\lambda) &= y_E \\ \left(N_\phi(1-\varepsilon^2) + h\right) \cdot \sin(\phi) &= z_E\end{aligned}$$

Dabei gilt für den Referenzellipsoiden WGS84:

$$\begin{aligned} a &= 6378137.000\,m && \text{(große Halbachse)} \\ b &= 6356752.3142\,m && \text{(kleine Halbachse)} \\ \varepsilon &= \frac{\sqrt{a^2-b^2}}{a} && \text{(numerische Exzentrizität)} \\ N_\phi &= \frac{a}{\sqrt{1-\varepsilon^2 \sin^2(\phi)}} && \text{(Krümmungsradius des Ersten Vertikals)} \end{aligned}$$

Die Lösung erfolgt wieder mit *fsolve* analog zur Berechnung der Empfängerkoordinaten im dritten Modellschritt. Die Gleichungen müssen dafür als Nullstellenproblem umgeformt werden. Als Startwerte für `fsolve` sollen die Werte von ϕ, λ und h aus dem vierten Modellschritt verwendet werden.

```
%%  Schritt 4  |  Umrechnen in geographische Höhe, Breite und Länge

%   Aufstellen der Umrechnungsformeln
berechne_h      = @(xE,yE,zE) sqrt(xE^2+yE^2+zE^2)-6378137.0;
berechne_phi    = @(xE,yE,zE) atan(zE/sqrt(xE^2+yE^2))*180/pi;
berechne_lambda = @(xE,yE,zE) atan2(yE,xE)*180/pi;

%   Berechnen der geographischen Koordinaten
h      = berechne_h(xE,yE,zE);
phi    = berechne_phi(xE,yE,zE);
lambda = berechne_lambda(xE,yE,zE);

%   Parameter des Referenzellipsoiden WGS 84
a = 6378137.0;
b = 6356752.3142;
epsilon = sqrt(a^2-b^2)/a;

%   Aufstellen der Gleichungen
Gleichungen = @(h,phi,lambda) ...
   [(a/sqrt(1-epsilon^2*(sin(phi)^2))+h)*cos(phi)*cos(lambda)-xE
    (a/sqrt(1-epsilon^2*(sin(phi)^2))+h)*cos(phi)*sin(lambda)-yE
    (a*(1-epsilon^2)/sqrt(1-epsilon^2*(sin(phi)^2))+h)*sin(phi)-zE];

%   Berechnen der korrigierten geographischen Koordinaten
Funktion = @(X) Gleichungen(X(1),X(2),X(3));
Startpunkt = [h phi*pi/180 lambda*pi/180];
X = fsolve(Funktion,Startpunkt);
h      = X(1);
phi    = X(2)*180/pi;
lambda = X(3)*180/pi;
```

Modellverbesserung 4 | Verwendung aller Satelliten

In MATLAB kann die Funktion `lsqnonlin` (least-squares nonlinear) verwendet werden um ein überbestimmte Gleichungssysteme mittels der Methode der kleinsten Fehlerquadrate zu lösen. Die gefundene Lösung sorgt dafür, dass in Summe die Fehler bezüglich jeder Gleichung minimal ist.

```
%   Auswählen der Satelliten (für acht Satelliten)
Satelliten = Datensatz.Satelliten;
Satellitenauswahl = [1 2 3 4 5 6 7 8];
```

Der Rest vom Auswahlschritt bleibt unverändert. Das Gleichungssystem im dritten Schritt muss noch erweitert werden und der Aufruf von *fsolve* ist durch den Aufruf von *lsqnonlin* zu ersetzen:

```
%%  Schritt 3  |  Berechnen der Empfängerkoordinaten

%   Aufstellen des Gleichungssystems (für acht Satelliten)
Gleichungssystem = @(xE,yE,zE) ...
   [sqrt((xS(1)-xE)^2+(yS(1)-yE)^2+(zS(1)-zE)^2)-dS(1)
    sqrt((xS(2)-xE)^2+(yS(2)-yE)^2+(zS(2)-zE)^2)-dS(2)
    sqrt((xS(3)-xE)^2+(yS(3)-yE)^2+(zS(3)-zE)^2)-dS(3)
    sqrt((xS(4)-xE)^2+(yS(4)-yE)^2+(zS(4)-zE)^2)-dS(4)
    sqrt((xS(5)-xE)^2+(yS(5)-yE)^2+(zS(5)-zE)^2)-dS(5)
    sqrt((xS(6)-xE)^2+(yS(6)-yE)^2+(zS(6)-zE)^2)-dS(6)
    sqrt((xS(7)-xE)^2+(yS(7)-yE)^2+(zS(7)-zE)^2)-dS(7)
    sqrt((xS(8)-xE)^2+(yS(8)-yE)^2+(zS(8)-zE)^2)-dS(8)];

%   Berechnen der Empfängerkoordinaten
Funktion = @(X) Gleichungssystem(X(1),X(2),X(3));
Startpunkt = [0 0 0];
X = lsqnonlin(Funktion,Startpunkt);
xE = X(1);
yE = X(2);
zE = X(3);
```

Um das Gleichungssystem auch flexibel für weniger oder mehr als acht Satelliten verwenden zu können, muss es nur in Vektorschreibweise mit den Vektoren x_S, y_S, z_S und d_S dargestellt werden:

```
%   Auswählen der Satelliten (für beliebig viele Satelliten)
Satelliten = Datensatz.Satelliten;
Satellitenauswahl = Satelliten;

%%  Schritt 3  |  Berechnen der Empfängerkoordinaten

%   Aufstellen des Gleichungssystems (für beliebig viele Satelliten)
Gleichungssystem = @(xE,yE,zE) sqrt((xS-xE).^2+(yS-yE).^2+(zS-zE).^2)-dS;

%   Berechnen der Empfängerkoordinaten
Funktion = @(X) Gleichungssystem(X(1),X(2),X(3));
Startpunkt = [0 0 0];
X = lsqnonlin(Funktion,Startpunkt);
xE = X(1);
yE = X(2);
zE = X(3);
```

Modellverbesserungen 2 & 4 | Empfängeruhr & Ausgleichsrechnung

Beide Modellverbesserungen lassen sich problemlos zusammenführen. Zunächst die Satellitenauswahl:

```
%   Auswählen der Satelliten (für acht Satelliten)
Satelliten = Datensatz.Satelliten;
Satellitenauswahl = [1 2 3 4 5 6 7 8];

%   Auswählen der Satelliten (für beliebig viele Satelliten)
Satelliten = Datensatz.Satelliten;
Satellitenauswahl = Satelliten;
```

Das Gleichungssystem für 8 Satelliten lautet:

```
%   Aufstellen des Gleichungssystems (für acht Satelliten)
Gleichungssystem = @(xE,yE,zE,Delta_dS) ...
   [sqrt((xS(1)-xE)^2+(yS(1)-yE)^2+(zS(1)-zE)^2)-(dS(1)-Delta_dS)
    sqrt((xS(2)-xE)^2+(yS(2)-yE)^2+(zS(2)-zE)^2)-(dS(2)-Delta_dS)
    sqrt((xS(3)-xE)^2+(yS(3)-yE)^2+(zS(3)-zE)^2)-(dS(3)-Delta_dS)
    sqrt((xS(4)-xE)^2+(yS(4)-yE)^2+(zS(4)-zE)^2)-(dS(4)-Delta_dS)
    sqrt((xS(5)-xE)^2+(yS(5)-yE)^2+(zS(5)-zE)^2)-(dS(5)-Delta_dS)
    sqrt((xS(6)-xE)^2+(yS(6)-yE)^2+(zS(6)-zE)^2)-(dS(6)-Delta_dS)
    sqrt((xS(7)-xE)^2+(yS(7)-yE)^2+(zS(7)-zE)^2)-(dS(7)-Delta_dS)
    sqrt((xS(8)-xE)^2+(yS(8)-yE)^2+(zS(8)-zE)^2)-(dS(8)-Delta_dS)];
```

Das Gleichungssystem in Vektorschreibweise lautet:

```
%   Aufstellen des Gleichungssystems (für beliebig viele Satelliten)
Gleichungssystem = @(xE,yE,zE,Delta_dS) ...
    sqrt((xS-xE).^2+(yS-yE).^2+(zS-zE).^2)-(dS-Delta_dS);
```

Zum Lösen des überbestimmten Gleichungssystems muss `lsqnonlin` anstelle von `fsolve` verwendet werden, sowie vier anstelle von drei Unbekannten berücksichtigt werden:

```
%   Berechnen der Empfängerkoordinaten
Funktion = @(X) Gleichungssystem(X(1),X(2),X(3),X(4));
Startpunkt = [0 0 0 0];
X = lsqnonlin(Funktion,Startpunkt);
xE = X(1);
yE = X(2);
zE = X(3);
Delta_tE = X(4)/299792458;
```

Modellverbesserung 5 | Gewichtung von Gleichungen (Zusatzaufgabe)

Hier ist nur die Gewichtung der Gleichungen so zu wählen, dass sie proportional zu den Signalstärken ist. Die einfachste Lösung ist, die Gewichtungen genau gleich den Signalstärken zu wählen. Zunächst müssen vor dem ersten Schritt noch die Signalstärken der ausgewählten Satelliten aus dem Datensatz kopiert werden:

```
%   Kopieren der ausgewählten Satellitendaten
tE = Datensatz.Empfangszeiten(Satellitenauswahl);
tS = Datensatz.Sendezeiten(Satellitenauswahl);
G = Datensatz.Signalstaerken(Satellitenauswahl);
Ephemeriden = Datensatz.Ephemeriden(:,Satellitenauswahl);
```

Um das Minimum der gewichtete Summe zu finden, muss nur die Quadratwurzel der einzelnen Gewichtungen an die Gleichungen multipliziert werden, denn wenn z. B. das Gleichungssystem aus den Gleichungen $f_1(x) = 0$, $f_2(x) = 0$ und $f_3(x) = 0$ besteht und die die positiven Zahlen g_1, g_2 und g_3 die zugehörigen Gewichtungen sind, soll der Wert für x bestimmt werden für den die Summe $g_1 \cdot f_1(x)^2 + g_2 \cdot f_2(x)^2 + g_3 \cdot f_3(x)^2$ minimal ist. Definiert man die Funktionen $h_i(x) := \sqrt{g_i} \cdot f_i(x)$ und berechnet die Kleinste-Quadrate-Lösung x für das Gleichungssystem $\{h_1(x) = 0 \wedge h_2(x) = 0 \wedge h_3(x) = 0\}$, so ist $h_1(x)^2 + h_2(x)^2 + h_3(x)^2$ minimal, und somit ist auch die gewichtete Summe $g_1 \cdot f_1(x)^2 + g_2 \cdot f_2(x)^2 + g_3 \cdot f_3(x)^2 = h_1(x)^2 + h_2(x)^2 + h_3(x)^2$ minimal.

Also lautet das Gleichungssystem in Vektorschreibweise:

```
%   Aufstellen des Gleichungssystems
Gleichungssystem = @(xE,yE,zE,Delta_dS) ...
    sqrt(G).*(sqrt((xS-xE).^2+(yS-yE).^2+(zS-zE).^2)-(dS-Delta_dS));
```

C. Evaluationsbogen

C.1. Bewertung des Workshops

CAMMP Day

Wie funktioniert eigentlich ...,
und was hat das mit Mathe zu tun?

Evaluation

Es besteht immer die Möglichkeit unsere Programme zu verbessern und wir würden gerne deine Meinung erfahren. Vielen Dank für deine Rückmeldung.

1. Persönliche Angaben

Jahrgangsstufe: _____ Geschlecht: □ weiblich □ männlich

Leistungskurse: _____

2. Bewertung des Workshops

	Trifft gar nicht zu (--)	Trifft eher nicht zu (-)	Trifft zum Teil zu (+)	Trifft voll zu (++)
Durch den Workshop habe ich mathematisches Modellieren besser begriffen.	◯	◯	◯	◯
Der Vortrag über Modellierung war hilfreich.	◯	◯	◯	◯
Der einführende Kurzfilm war hilfreich.	◯	◯	◯	◯
Die Einführung in MATLAB war hilfreich.	◯	◯	◯	◯
Der Umgang mit MATLAB fiel mir schwer.	◯	◯	◯	◯
Die Aufgaben waren zu einfach.	◯	◯	◯	◯
Die Aufgaben waren zu schwierig.	◯	◯	◯	◯
Die Hilfekarten waren hilfreich.	◯	◯	◯	◯
Das Hilfesystem in MATLAB war hilfreich.	◯	◯	◯	◯

Kommentar:

Was würdest du am Workshop verändern bzw. verbessern wollen?

_____

_____

_____

_____

_____

Bitte wenden...

1/2

C.2. Fragen des zdi

Was hast du für dich persönlich durch die Teilnahme am Workshop gelernt?

3. Weiterführende Fragen

	Trifft gar nicht zu (--)	Trifft eher nicht zu (-)	Trifft zum Teil zu (+)	Trifft voll zu (++)
1. Der Workshop hat mein Interesse an den Naturwissenschaften und Technik gesteigert.	◯	◯	◯	◯
2. Die Anleitungen zu den Aufgaben waren zufriedenstellend und gut verständlich.	◯	◯	◯	◯
3. Die Dozenten haben die Inhalte plausibel und klar dargestellt.	◯	◯	◯	◯
4. Die Betreuer haben das selbstständige Arbeiten gefördert.	◯	◯	◯	◯
5. Es wurden interessante Berufe und Studiengänge vorgestellt.	◯	◯	◯	◯
6. Ich kann mir gut vorstellen, später ein Studium oder eine Ausbildung im Bereich Technik oder Naturwissenschaften aufzunehmen.	◯	◯	◯	◯
7. Ich würde einen solchen Workshop gerne noch einmal besuchen.	◯	◯	◯	◯

Kommentar:

Abschließender persönlicher Kommentar (z.B. Lob, Kritik, Verbesserungsvorschläge):

2/2

D. Dateien auf der beiliegenden CD

Auf der beiliegenden CD sind folgende Dateien verfügbar:

1. `114255.nav`: Die RINEX-Navigationsdatei einer GPS-Messung
2. `114255.obs`: Die RINEX-Observationsdatei einer GPS-Messung
3. `berechne_Satellitenkoordinaten.m`: Die Funktion zum Berechnen der Satellitenkoordinaten
4. `berechne_Satellitenuhrfehler.m`: Die Funktion zum Berechnen der Satellitenuhrfehler
5. `einlesen_Satellitendaten.m`: Die Funktion zum Einlesen der Satellitendaten aus den RINEX-Dateien
6. `gps.m`: Das Hauptprogramm für die Schülerinnen und Schüler
7. `gps_master.m`: Die Lösungen der Modellschritte für die Betreuer
8. `gps_solution.m`: Die Lösungen der Modellverbesserungen für die Betreuer
9. `ueberpruefe_Loesung.m`: Die Überprüfungsfunktion für die Lösungen der Modellschritte

E. Erklärung

Ich versichere, dass ich die schriftliche Hausarbeit – einschließlich beigefügter Zeichnungen, Kartenskizzen und Darstellungen – selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, habe ich in jedem Fall unter Angabe der Quelle deutlich als Entlehnung kenntlich gemacht.

Datum

Unterschrift