Celestial navigation algorithms library

Purpose

celnav is a library contains a set of algorithms for solving typical astronavigational and navigational tasks, especially:

- Observer position fix using observed position of **celestial** objects.
- Observer position fix using observed position of **known** objects (via triangulation or trilateration).
- Orthodromy and great circle route waypoints calculation.
- Required heading at current waypoint calculation.

- True heading and magnetic compass correction calculation using observed positions of **celestial** objects.
- True heading and magnetic compass correction calculation using observed positions of **known** objects.
- Solving navigational triangle problem.

Main features

- **minimum external dependencies** - the only thing you utterly need is NumPy. System requires 3 other packages, but they are optional.
- **offline and standalone** - library has its own "astronomy engine", so you would not face a need to download ephemerides somewhere in the middle of North Atlantic.
- **precision** - CEP of position fix via stars - about 1/5 nmi in autonomous mode and about 1/10 nmi (~180 m) with AstroPy ephemeris. *(Not something extra… But this is not for to-the-door pizza delivery!)*
- **simplicity** - code kept as simple as possible on purpose. No advanced programming techniques - just plain functions and math.
- **universal** - all essential functions included
- **lightweight** - 228 Kb of code.
- **solid** - over 95% of code covered with unit tests

Detailed description

Algorithms

celnav includes algorithms for:

- determination of current GMT from local time
- sight reduction (different than sight reduction algorithm of Sumner and Bowditch)
- position fix (using 2 or 3 sight reductions)
- 3 points 3 angles position fix (using Tienstra method)
- 3 points 3 ranges position fix (using modified trilateration algorithm from this article)
- orthodromy calculation (based on information from Wikipedia article)
- ortodromy waypoint & heading calculation
- true heading and magnetic deviation (compass correction) calculation via angular coordinates of celestial objects
- true heading and magnetic deviation (compass correction) calculation via angular coordinates of known surface objects
- DR ("dead reckoning") calculation through solving navigation triangle problem

IMPORTANT! *3P3A fix, 3P3R fix and DR calculation are not counting Earth oblateness, using spherical (geocentric) Earth model. In long ranges they can produce errors! I will fix this in later releases.*

Astronomical data sources

celnav provided with two sources of astronomical information:

- AstroPy
- local and standalone "astronomical engine", utilizing algorithms from "*Fundamentals of Astrodynamics and Applications, Fourth Edition*" by **David Vallado** and "*Astronomical Algorithms* (1998), 2nd ed." by Jean Meeus.

You can choose "engine", switching it in config.ini. See below (or follow comments in this file) for additional info.

AstroPy provides more precise sight reduction, but will require downloading of ephemeris, which can be inconvenient in case if you have no internet connection.

Local "almanac engine" is standalone. Using it system does not need internet connection, but provide slightly less precision.

Celestial bodies for sight reduction

celnav can operate with all celestial bodies mentioned in Nautical Almanac. This includes:

- Sun

Navigation planets, i.e.,

- `Venus`
- `Mars`
- `Jupiter`
- `Saturn`

59 navigation stars, includes:

- `Acamar`
- `Achernar`
- `Acrux`
- `Adhara`
- `Albireo`
- `Aldebaran`
- `Alioth`
- `Alkaid`
- `Al Na-ir`
- `Alnilam`

- Alphard
- Alphecca
- Alpheratz
- Altair
- Ankaa
- Antares
- Arcturus
- Atria
- Avior
- Bellatrix
- Betelgeuse
- Canopus
- Capella
- Castor
- Deneb
- Denebola
- Diphda
- Dubhe
- Elnath
- Eltanin
- Enif
- Fomalhaut
- Gacrux
- Gienah
- Hadar
- Hamal
- Kochab
- Markab
- Menkar
- Menkent
- Miaplacidus
- Mirfak
- Nunki
- Peacock
- Polaris
- Pollux
- Procyon
- Rasalhague

- `Regulus`
- `Rigel`
- `Sabik`
- `Schedar`
- `Shaula`
- `Sirius`
- `Spica`
- `Suhail`
- `Vega`
- `Zuben-ubi`

Also, system supporting sight reduction for the Moon.

IMPORTANT! *Because of very complex motion of the Moon, local almanac provides it's position not-so-precise. I am not recommend to use it for Moon sight reduction for practical purposes. With the time, I'll fix this by implementation of more precise Moon motion model.*

Usage

Installation

Requirements

- Python>=3.7
- Internet connection (**optional**, for downloading updated astronomical data).

Dependencies

- numpy>=1.2
- astropy>=5.0
- requests>=2.0
- setuptools>=65.0

Modern Python suffers from heavy dependency hell, which nowadays looks more not like a dependency but as an addiction. To avoid this, I tried to reduce number of dependencies as hard as possible.

Setup

1. Clone (or download and extract) this repo into local folder.
2. In local repository folder open console and execute command:

```
python -m pip install -e .
```

(or if you are using Windows, you can just run install.bat in the root of the repo).

3. Done.

Updating astronomical catalog

Up-to-date astronomical catalog provided with this repo in src/data/catalog.dat. Normally, **you should not touch it**. If you need to refresh it, you can run src/catalog.py in your repo. Follow instructions of the app - system will connect to online astronomical catalog and download all required data.

IMPORTANT! *You need internet connection for this operation.*

Choosing astronomical data source

As I mentioned earlier system can use as data source AstroPy or local "astronomical almanac". AstroPy require internet but provide better precision, local almanac is offline, but less precise. You may switch data source whenever you want. To switch data source, open src/config.ini and change value of parameter *AlmanacEngine* to

- `AlmanacEngine=0` (for AstroPy)
- `AlmanacEngine=1` (for local almanac, default value)

Importing

Just add import instruction in the head of your file, like this:

```
import celnav
```

And you're done!

Functions

AngleToDecimal(string)

Converting string with geographical coordinates, like **"E100°36.31'"** to decimal degrees.

```
phi=celnav.AngleToDecimal("N13°55.28'") #latitude of Bangkok

# phi == 13.921333333333333
```

LTToGMT(Y, M, D, h, m, s, lambdaAP)

Converts local time to GMT, depending on current longitude.

```
Y,M,D,h,m,s=(2025,1,20,12,0,0) #local time of Bangkok (GMT+7)
lambda_=celnav.AngleToDecimal("E100°36.31'") #longitude of Bangkok
Y,M,D,h,m,s=celnav.LTToGMT(Y,M,D,h,m,s,lambda_)

# Y,M,D,h,m,s == (2025 1 20 5 0 1.341104507446289e-05) #GMT
```

SightReduction(phiAP, lambdaAP, Y, M, D, h, m, s, celestialObjectName, el, indexCorrection=0, hoe=0, T=10, P=1010.0, limb=0)

Calculate sight reduction from assumed position, date and parameters of sighted celestial object. Getting in count all corrections you need to do for precise measurement.

```
phiAP=33.0              #latitude of assumed position
lambdaAP=-117.0   #longitude of assumed position

Y=2024      #year
M=4         #month
D=12        #day
h=16        #hour, GMT
m=0               #minute
s=0               #second

celestialObjectName="Sun"

el=32.394512#elevation of sighted celestial object over horizon

IC=0              #index correction of sextan
hoe=20            #height of an eye, m.
T=15.0            #environment temperature, degrees Celsius
P=1013.25         #environment pressure, millibars
limb=-1                 #limb we choosing (Like in Bowditch "-1": for Sun and
Moon, "0" for stars and planets)

a,Z=celnav.SightReduction(phiAP, lambdaAP, Y, M, D, h, m, s,
celestialObjectName, el, IC, hoe, T, P, limb)

#a,Z == (0.1809029519340868 -100.92368252258527)
```

TwoObjectFix(phiAP, lambdaAP, a1, Z1, a2, Z2)

Calculating position from AP (assumed position) and results of 2 sight reductions.

```
phiAP=33.0
lambdaAP=-117.0

a1=-0.25997911530550155
Z1=-151.72526130264336

a2=-0.3220529491543829
Z2=-166.80424087813753

phi,lambda_=celnav.TwoObjectFix(phiAP,lambdaAP,a1,Z1,a2,Z2)

#phi,lambda_ == (33.358288093 -116.859574639)
#observation point is Palomar observatory, with phi0,lambda0 ==
(33.3562811,-116.8651156)
#i.e., error=533.32 m.
```

ThreeObjectFix(phiAP, lambdaAP, a1, Z1, a2, Z2, a3, Z3)

Calculating position from AP (assumed position) and results of 3 sight reductions.

```
phiAP=33.0
lambdaAP=-117.0

a1=-0.22178062985341995
Z1=-143.8725861697037

a2=-0.29762793886806804
Z2=-160.01234344989314

a3=-0.35566029746129857
Z3=-178.50219034779053

phi,lambda_=celnav.ThreeObjectFix(phiAP,lambdaAP,a1,Z1,a2,Z2,a3,Z3)

#phi,lambda_ == (33.358716255 -116.861853266)
#observation point is Palomar observatory, with phi0,lambda0 ==
(33.3562811,-116.8651156)
#now error=401.10 m.
```

ThreeAnglesFix(phi1, lambda1, phi2, lambda2, phi3, lambda3, alpha12, alpha23, alpha31)

Determine position from 3 known points on surface and angles between them. Implemented Tienstra method. ***IMPORTANT!*** *Angles named this way on purpose. Alpha12 means angle BETWEEN p1 and p2 (not an azimuth from North) and so on.*

```
phi1=51.0731459 #Grand Mosque
lambda1=71.410925
phi2=51.1256397 #Harzat Sultan Mosque
lambda2=71.4724698
phi3=51.1090188 #Barys Arena
lambda3=71.3953671

alpha12=96.6
alpha23=144+83.4
alpha31=36

phi,lambda_=celnav.ThreeAnglesFix(phi1,lambda1,phi2,lambda2,phi3,lambda3,alpha
12,alpha23,alpha31)

#phi,lambda_ == (51.134704256 71.432621797)
#observation point was Baiterek tower, Astana, with phi0,lambda0 ==
(51.1282921,71.4305781)
#error is about 726 m.
```

ThreeRangesFix(phi1, lambda1, phi2, lambda2, phi3, lambda3, r1, r2, r3)

Determine position from 3 known points on surface and ranges towards them. Trilateration. ***IMPORTANT!*** *Ranges ALWAYS in KILOMETERS!*

```
phi1=51.0731459 #Grand Mosque
lambda1=71.410925
phi2=51.1256397 #Harzat Sultan Mosque
lambda2=71.4724698
phi3=51.1090188 #Barys Arena
lambda3=71.3953671

r1=6.31 #km;
r2=2.97
r3=3.25

phi,lambda_=celnav.ThreeRangesFix(phi1,lambda1,phi2,lambda2,phi3,lambda3,r1,r2
,r3)
```

```
#phi,lambda_ == (51.136092319 71.430788396)
#observation point was Baiterek tower, Astana, with phi0,lambda0 ==
(51.1282921,71.4305781)
#error is about 873 m.
```

As you can see, 3P3A and 3P3R fixes are not so precise in short ranges. For better precision you should use points on longer distances spread more equally.

Orthodromy(phi1, lambda1, phi2, lambda2)

Calculating orthodromy, returning heading at start, heading at destination, degrees of arc of great circle and distance in kilometers.

```
phi1=59.7986813         #Pulkovo airport
lambda1=30.2689333
phi2=51.0257228         #Astana airport
lambda2=71.4493939

alpha1,alpha2,delta12,s12=celnav.Orthodromy(phi1,lambda1,phi2,lambda2)

#alpha1 == 92.4963366599314         #HDG at start
#alpha2 == 126.96312067753773       #HDG at destination
#delta12 == 24.489647649865553      #degrees of arc of GC between points
#s12== 2726.1751060897222                 #actual distance between points, km
```

HeadingAtWaypoint(phi1, lambda1, alpha1, d)

Calculating coordinates of waypoint on the orthodromy and heading at it, if orthodromy starting at [*phi1, lambda1*], with heading *alpha1*, and waypoint is placed on distance *d*.

```
phi1=59.7986813         #Pulkovo airport
lambda1=30.2689333
alpha1=92.4963366599314 #exact heading to Astana, from prev. example
d=2726.1751060897222          #exact distance to Astana, from prev. example

phi2,lambda2,alpha2=celnav.HeadingAtWaypoint(phi1,lambda1,alpha1,d)

#phi2 == 51.0257228                #Astana airport latitude
#lambda2== 71.4493939              #Astana airport longitude
#alpha2 == 126.96312067753773 #Arrival direction to airport
```

CompassCorrectionCOZ(magHdg, phi, lambda_, Y, M, D, h, m, s, celestialObjectName, celestialObjectAz)

Calculating current true heading and compass correction from current magnetic heading and parameters of observed celestial object. ***IMPORTANT!*** *celestialObjectAz here is RAW MAGNETIC azimuth, straightly read from compass.*

```
magHdg=0.0

phi=51.1282921 #Baiterek, Astana
lambda_=71.4305781

Y=2025
M=1
D=17
h=6 #UTC
m=0
s=0

celestialObjectName="Sun"
celestialObjectAz=149.421395  #RAW azimuth to celestial object from MAGNETIC
COMPASS

trueHdg,corr=celnav.CompassCorrectionCOZ(magHdg,phi,lambda_,Y,M,D,h,m,s,celest
ialObjectName,celestialObjectAz)

#trueHdg,corr == (10.517144451964327 10.517144451964327)
#magnetic deviation in Astana at January 2025 equals -10.1
```

CompassCorrectionPOZ(magHdg, phiPos, lambdaPos, phiObj, lambdaObj, azObj)

Calculating current true heading and compass correction from position of known object and magnetic azimuth to it. ***IMPORTANT!*** *Same as above, azObj here is RAW MAGNETIC azimuth, straightly read from compass.*

```
magHdg=0.0

phiPos=51.1282921 #Baiterek, Astana
lambdaPos=71.4305781
phiObj=51.0731459 #Grand Mosque
lambdaObj=71.410925
```

```
azObj=360.0-166.8

trueHdg,corr=celnav.CompassCorrectionPOZ(magHdg,phiPos,lambdaPos,phiObj,lambda
Obj,azObj)

#trueHdg,corr == (6.415140248887724 6.415140248887724)
#magnetic deviation in Astana at January 2025 equals -10.1.
#Not so precise as with celestial objects, but if there is no other way...
```

DeadReckoning(phi0, lambda0, V, Vdir, D, Ddir, dt)

Calculating DR point from velocity, drift and time.

```
phi0=0            #initial position latitude
lambda0=0         #initial position longitude

V=110*1.852       #speed, km/h
Vhdg=90                 #flight direction

D=10*1.852        #drift, km/h
Dhdg=90                 #drift direction

dt=0.5            #time of flight, h

phi1,lambda1=celnav.DeadReckoning(phi0,lambda0,V,Vhdg,D,Dhdg,dt)

#phi1,lambda1 == (0.0, 1.0)
```

Further improvements

1. Precise ephemeris for Moon, Mars, Saturn, Sun
2. Improvement precision of 3P3A fix, 3P3R fix and DR calculation.
3. Additional methods of position fix.
4. Improvements for usage on other planets of Solar System (Moon, Mars, Jupiter satellites).

Feedback and questions

Feel free to ask me through PM if you have any questions or offers.

Practical usage notice

THIS IS EDUCATIONAL PROJECT. PRECISE CALCULATIONS AND ALGORITHMS ARE IMPLEMENTED BUT **CELNAV** NEVER BEEN TESTED ENOUGH IN PRACTICE.

BEFORE PRACTICAL APPLICATION CHECK RESULTS MULTIPLE TIMES IN SAFETY ENVIRONMENT.

Disclaimer

The "**celnav**" Python library is provided on an "AS IS" and "AS AVAILABLE" basis, without any warranties or representations, express or implied. To the fullest extent permissible under applicable law, the author disclaims all warranties, express or implied, including, but not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

The author does not warrant that the functions contained in the library will meet your requirements, that the operation of the library will be uninterrupted or error-free, or that any defects in the library will be corrected. No oral or written information or advice given by the author shall create a warranty.

In no event shall the author be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

The user assumes all responsibility for any damages resulting from the use of the "celnav" library, including, but not limited to, errors or inaccuracies in the data produced, misuse of the library, or any harm that may occur in reliance on its functionality. This includes any incidents leading to personal injury, death, or significant physical or environmental damage.

By using the "**celnav**" library, you acknowledge that you have read this disclaimer and agree to its terms.

Legal notice

This software is distributing under MIT license with Attribution Clause. You may read it in file LICENSE.md

Artwork of **celnav** official logo was made by sci-fi digital artist Anna Apalkova (Amiko Panda). Used and published with all required permissions.