HW6

Marco Boscato - 2096921

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import pandas as pd
```

Exercise 1

From the note we know that the *fully normalized associated legendre functions* is the following:

$$\bar{P}_n^n(\sin\varphi) = f_n \cos\varphi \bar{P}_{n-1}^{n-1}(\sin\varphi)$$

with

$$f_n = \sqrt{(1+\delta_{1n})\frac{2n+1}{2n}}$$

and with start seed of $\bar{P}_0^0(\sin\varphi) = 1$.

For the indices below the diagonal the formula is:

$$\bar{P}_n^{n-1}(\sin\varphi) = g_n^m \sin\varphi \bar{P}_{n-1}^{n-1}(\sin\varphi)$$

with

$$g_n^m = \sqrt{\frac{(2n+1)(2n-1)}{(n+m)(n-m)}}$$

and for a generic index:

$$\bar{P}_n^m(\sin\varphi) = g_n^m \sin\varphi \bar{P}_{n-1}^m(\sin\varphi) - h_n^m \sin\varphi \bar{P}_{n-2}^m(\sin\varphi)$$

with

$$h_n^m = \frac{g_n^m}{g_{n-1}^m}$$

Each $\bar{P}_j^i$ correspond to a value on a matrix with indices *i* = *column* and *j* = *row* where for each *j*, that correspond to the *n*, *i* (corresponding to *m*) goes from 0 to *n*

In [2]:
```
# COEFFICIENTS

def delta(i,j):
    if i == j:
        return 1
    else:
        return 0

fn = lambda n: np.sqrt((1+delta(1,n)) * (2*n+1) / (2*n))
```

```
g_mn = lambda m,n: np.sqrt((4*n**2. - 1)/(n**2. - m**2.))
h_mn = lambda m,n, g_mn: g_mn(m,n) / g_mn(m,n-1)

# LEGENDRE FUNCTIONS

def Legendre_matrix(N, phi):

    P = np.zeros([N+1, N+1], float)
    P[0,0] = 1

    for n in range(1, N+1):
        P[n, n] = fn(n) * np.cos(phi) * P[n-1, n-1]
        P[n, n-1] = g_mn(n-1, n) * np.sin(phi) * P[n-1, n-1]

        if n > 1:
            for m in range(0, n-1):
                P[n, m] = g_mn(m,n) * np.sin(phi) * P[n-1, m] - h_mn(m,n, g_mn) * P[n-2,

    return P


N = 5   # maximum degree and order N
phi = np.pi/4

print(Legendre_matrix(N, phi))
```

```
[[ 1.          0.          0.          0.          0.          0.        ]
 [ 1.22474487  1.22474487  0.          0.          0.          0.        ]
 [ 0.55901699  1.93649167  0.96824584  0.          0.          0.        ]
 [-0.46770717  1.71846589  1.81142209  0.73950997  0.          0.        ]
 [-1.21875     0.59292706  2.09631373  1.56873755  0.55463248  0.        ]
 [-1.24589169 -0.85152666  1.50195186  2.14609247  1.30072846  0.41132646]]
```

we can also write the derivative of the normalized Legendre functions in respect to the latitude given by the formulas

$$\frac{d\bar{P}_n(\sin\varphi)}{d\varphi} = \sqrt{\frac{n(n+1)}{2}}\bar{P}_n^1(\sin\varphi)$$

$$\frac{d\bar{P}_n^m(\sin\varphi)}{d\varphi} = \frac{1}{2}k_n^m\bar{P}_n^{m+1}(\sin\varphi) - l_n^m\bar{P}_n^{m-1}(\sin\varphi)$$

where

$$k_n^m = \frac{1}{2}\sqrt{(n-m)(n+m+1)}$$

$$l_n^m = \frac{1}{2}\sqrt{(1+\delta_{1m})(n+m)(n-m+1)}$$

In [3]:

```
# DERIVATE OF LEGENDRE FUNCTIONS
P = Legendre_matrix(N, phi)

def Legendre_prime_matrix(N):
    fn_prime = lambda n: np.sqrt((n*(n+1))*0.5)
    k_mn = lambda m,n: 0.5*np.sqrt((n-m)*(n+m+1))
    l_mn = lambda m,n: 0.5*np.sqrt((1 + delta(1,m))*(m+n)*(n-m+1))

    dP = np.zeros([N+1, N+1], float)

    for n in range(N+1):
        dP[n, 0] = fn_prime(n) * P[n, 1]
        #if n > 0:
```

```
        for m in range(1, n+1):
            #if m == N then P[n, m+1] is out of bound of the matrix size, I can neglect
            if m == N:
                dP[n, m] = - l_mn(m,n) * P[n, m-1]
            else:
                dP[n, m] = k_mn(m,n)*P[n, m+1] - l_mn(m,n)*P[n, m-1]
    return dP

dP = Legendre_prime_matrix(N)
print(dP)
```

```
[[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [ 1.22474487e+00 -1.22474487e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [ 3.35410197e+00 -6.66133815e-16 -1.93649167e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [ 4.20936456e+00  4.00975373e+00 -1.81142209e+00 -2.21852992e+00
   0.00000000e+00  0.00000000e+00]
 [ 1.87500000e+00  8.30097886e+00  1.67705098e+00 -3.13747510e+00
  -2.21852992e+00  0.00000000e+00]
 [-3.29794858e+00  8.79910884e+00  7.50975928e+00 -9.19753916e-01
  -3.90218539e+00 -2.05663228e+00]]
```

Exercise 2

we can check if our functions are corrected by checking that these exact relations are satisfied

$$\sum_{n=0}^{N}\sum_{m=0}^{n}\left(\bar{P}_n^m(\sin\varphi)\right)^2 = (N+1)^2$$

$$\sum_{n=0}^{N}\sum_{m=0}^{n}\left(\frac{1}{\cos\varphi}\frac{d\bar{P}_n^m(\sin\varphi)}{d\varphi}\right)^2 = \frac{N(N+2)(N+1)^2}{4}$$

In [4]:
```
Pn_prove = np.sum(np.matrix.flatten(P)**2)
dPn_prove = np.cos(phi)**(-2) * np.sum(np.matrix.flatten(dP)**2)
dPn_prove2 = np.sum(np.matrix.flatten(dP)**2)

print(Pn_prove)
print((N+1)**2)

print(dPn_prove)
print((0.25*N*(N+2)*(N+1)**2))

tollerance = 1e-5

if (np.abs(Pn_prove - (N+1)**2) < tollerance) and (np.abs(dPn_prove - 0.25*N*(N+2)*(N+1)
    print('The Legendre functions are correct')
else:
    print('The Legendre functions are not correct')

if (np.abs(Pn_prove - (N+1)**2) < tollerance) and (np.abs(dPn_prove2 - 0.25*N*(N+2)*(N+1
    print('The Legendre functions are correct')
else:
    print('The Legendre functions are not correct')
```

```
36.000000000000014
36
630.0000000000001
315.0
```

```
The Legendre functions are not correct
The Legendre functions are correct
```

Here we can see that the derivative part of the matrix does not seem to be corrected because the relation is not satisfied. But if we consider

$$\sum_{n=0}^{N}\sum_{m=0}^{n}\left(\frac{1}{\cos\varphi}\frac{d\bar{P}_n^m(\sin\varphi)}{d\varphi}\right)^2=\frac{N(N+2)(N+1)^2}{4}$$

as

$$\sum_{n=0}^{N}\sum_{m=0}^{n}\left(\frac{d\bar{P}_n^m(\sin\varphi)}{d\varphi}\right)^2=\frac{N(N+2)(N+1)^2}{4}$$

the relation is satisfied. Maybe there's an error in the function that calculates the derivative matrix

Exercise 3

The physical components of the gravitational force, in our case the acceleration, in body-fixed coordinates are

$$a_x=\cos\varphi\cos\lambda\partial_r U-\frac{1}{r}\sin\varphi\cos\lambda\partial_\phi U-\frac{1}{r}\frac{\sin\lambda}{\cos\varphi}\partial_\lambda U$$
$$a_y=\cos\varphi\sin\lambda\partial_r U-\frac{1}{r}\sin\varphi\sin\lambda\partial_\phi U+\frac{1}{r}\frac{\cos\lambda}{\cos\varphi}\partial_\lambda U$$
$$a_z=\sin\varphi\partial_r U+\frac{1}{r}\cos\varphi\partial_\phi U$$

where the terms $\frac{1}{\cos\varphi}\partial_{\lambda U}, \partial_\phi U, \partial_r U$ acn be rewritten as:

$$\frac{1}{\cos\varphi}\partial_{\lambda U}=\frac{GM}{r}\sum_{n=1}^{\infty}\sum_{m=1}^{n}\left(\frac{a_e}{r}\right)^n\bar{K}_{nm}(\lambda)m\left[r_{nm}\bar{P}_{n-1}^{m+1}(\sin\varphi)+s_{nm}\bar{P}_{n-1}^{m-1}(\sin\varphi)\right]$$
$$\partial_r U=-\frac{GM}{r^2}\left[1+\sum_{n=1}^{\infty}\sum_{m=0}^{n}(n+1)\left(\frac{a_e}{r}\right)^n\bar{H}_{nm}(\lambda)\bar{P}_n^m(\sin\varphi)\right]$$
$$\partial_\phi U=\frac{GM}{r}\sum_{n=1}^{\infty}\sum_{m=0}^{n}\left(\frac{a_e}{r}\right)^n\bar{H}_{nm}(\lambda)\frac{d\bar{P}_n^m(\sin\varphi)}{d\varphi}$$

with

$$\bar{H}_{nm}(\lambda)=C_{nm}\cos(m\lambda)+S_{nm}\sin(m\lambda)$$
$$\bar{K}_{nm}(\lambda)=S_{sm}\cos(m\lambda)+C_{nm}\sin(m\lambda)$$
$$r_{nm}=0.5\sqrt{(n-m)(n+m+1)}$$
$$s_{nm}=r_{nm}(n+m)(n+m-1)\sqrt{\frac{2n+1}{2n-1}}$$

and the calculation are done with the fully normalized functions developed above and using the EGM96 geopotential field model

In [5]:
```
# import geopotentilal model
import pandas as pd

ICGEM = pd.read_csv('EGM96.gfc', sep='\s+', skiprows=21, header=None, names=['gfc','n',
ICGEM.head(19)
```

Out[5]:

	gfc	n	m	C	S	sigmaC	sigmaS
0	gfc	0	0	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
1	gfc	2	0	-4.841654e-04	0.000000e+00	3.561063e-11	0.000000e+00
2	gfc	2	1	-1.869876e-10	1.195280e-09	1.000000e-30	1.000000e-30
3	gfc	2	2	2.439144e-06	-1.400167e-06	5.373915e-11	5.435327e-11
4	gfc	3	0	9.572542e-07	0.000000e+00	1.809424e-11	0.000000e+00
5	gfc	3	1	2.029989e-06	2.485132e-07	1.396517e-10	1.364588e-10
6	gfc	3	2	9.046278e-07	-6.190259e-07	1.096233e-10	1.118287e-10
7	gfc	3	3	7.210727e-07	1.414356e-06	9.515628e-11	9.328509e-11
8	gfc	4	0	5.398739e-07	0.000000e+00	1.042368e-10	0.000000e+00
9	gfc	4	1	-5.363216e-07	-4.734403e-07	8.567440e-11	8.240849e-11
10	gfc	4	2	3.506941e-07	6.626716e-07	1.600019e-10	1.639058e-10
11	gfc	4	3	9.907718e-07	-2.009284e-07	8.465780e-11	8.266251e-11
12	gfc	4	4	-1.885608e-07	3.088532e-07	8.731536e-11	8.785282e-11
13	gfc	5	0	6.853235e-08	0.000000e+00	5.438309e-11	0.000000e+00
14	gfc	5	1	-6.210121e-08	-9.442261e-08	2.799689e-10	2.808288e-10
15	gfc	5	2	6.524383e-07	-3.233496e-07	2.374738e-10	2.435700e-10
16	gfc	5	3	-4.519554e-07	-2.148472e-07	1.711164e-10	1.681065e-10
17	gfc	5	4	-2.953016e-07	4.966589e-08	1.198127e-10	1.184979e-10
18	gfc	5	5	1.749720e-07	-6.693843e-07	1.164256e-10	1.159003e-10

In [6]:
```
# Coefficients
H = lambda l,n,m,C,S : C[n,m]*np.cos(m*l)+S[n,m]*np.sin(m*l)
K = lambda l,n,m,C,S : S[n,m]*np.cos(m*l)-C[n,m]*np.sin(m*l)
r = lambda n,m : 0.5*((n+m)*(n+m+1))**0.5
s = lambda n,m : r(n,m)*(n+m)*(n+m-1)*((2*n+1)/(2*n-1))**0.5

# acceleration in body-fixed RF
def acc_bf(coord, C, S, GM, a_e, N):

    R = np.linalg.norm(coord)
    l = np.arctan2(coord[1],coord[0])
    phi = np.arcsin(coord[2]/R)
    A = a_e/R

    P = Legendre_matrix(N=N, phi=phi)
    dP = Legendre_prime_matrix(N=N)

    dUr_temp = 0.0
    for n in range(1, N+1):
        for m in range(0, n+1):
            dUr_temp = dUr_temp + ((n+1) * A**n * H(l,n,m, C, S)*P[n,m])

    dUr = -GM/R**2*(1+dUr_temp)

    dUphi_temp = 0.0
```

```
    for n in range(1, N+1):
        for m in range(0, n+1):
            dUphi_temp = dUphi_temp + (A**n * H(l,n,m,C,S)*dP[n,m])

    dUphi = (GM/R) * dUphi_temp

    dUl_temp = 0.0
    for n in range(1, N+1):
        for m in range(1, n+1):
            #similar case to Ex.1
            if m==N:
                dUl_temp = dUl_temp + (A**n * K(l,n,m,C,S) * m * s(n,m)*P[n-1,m-1])
            else:
                dUl_temp = dUl_temp + (A**n * K(l,n,m,C,S) * m * (r(n,m)*P[n-1,m+1] + s(

    dUl = (GM/R) * dUl_temp

    ax = np.cos(phi)*np.cos(l)*dUr - np.sin(phi)*np.cos(l)/R*dUphi - np.sin(l)/R*dUl
    ay = np.cos(phi)*np.sin(l)*dUr - np.sin(phi)*np.sin(l)/R*dUphi + np.cos(l)/R*dUl
    az = np.sin(phi)*dUr + np.cos(phi)/R*dUphi

    return np.array([ax, ay, az])
```

In [7]:

```
# Save the coefficients up to the highest required degree a matrix.

#Index function
i = lambda n,m : n*(n+1)//2+m

C = np.zeros([N+1, N+1], float)
S = np.zeros([N+1, N+1], float)

# The values for n=1 are all zero because the origin of the RF is in the center of mass
# I insert them manually
new_rows = [['gfc', 1, 0, 0, 0, 0, 0],['gfc', 1, 1, 0, 0, 0, 0]]
df_top = ICGEM.iloc[:1,]
df_bottom = ICGEM.iloc[1:,]

df_temp = pd.DataFrame(new_rows, columns=ICGEM.columns)

print(df_temp.head())

ICGEM = pd.concat([df_top, df_temp, df_bottom]).reset_index(drop=True)

ICGEM.head(10)
```

```
   gfc  n  m  C  S  sigmaC  sigmaS
0  gfc  1  0  0  0       0       0
1  gfc  1  1  0  0       0       0
```

Out[7]:

	gfc	n	m	C	S	sigmaC	sigmaS
0	gfc	0	0	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
1	gfc	1	0	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
2	gfc	1	1	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
3	gfc	2	0	-4.841654e-04	0.000000e+00	3.561063e-11	0.000000e+00
4	gfc	2	1	-1.869876e-10	1.195280e-09	1.000000e-30	1.000000e-30
5	gfc	2	2	2.439144e-06	-1.400167e-06	5.373915e-11	5.435327e-11
6	gfc	3	0	9.572542e-07	0.000000e+00	1.809424e-11	0.000000e+00
7	gfc	3	1	2.029989e-06	2.485132e-07	1.396517e-10	1.364588e-10
8	gfc	3	2	9.046278e-07	-6.190259e-07	1.096233e-10	1.118287e-10
9	gfc	3	3	7.210727e-07	1.414356e-06	9.515628e-11	9.328509e-11

In [8]:
```
#Index function
i = lambda n,m : n*(n+1)//2+m

C[0,0] = ICGEM.iloc[0]['C']
S[0,0] = ICGEM.iloc[0]['S']


for n in range(2, N+1):
    for m in range(0, n+1):
        C[n,m] = ICGEM.iloc[i(n,m)]['C']
        S[n,m] = ICGEM.iloc[i(n,m)]['S']

GM = 0.3986004415e15  # Gravitational parameter
R_E = 6.3781363006  # Radius of the Earth
```

Exercise 4

To develop a function to integrate the equations of motion of an Earth satellite subject to the gravitational perturbation of the geopotential, we use the library *pyerfa* to convert the coordinates from International Terrestrial RF (ITRS) to Geocentric Celestial RS (GCRS) to compute the numerical integration of the equations of motion, since I had errors with the installation of *pysofa*.

In [9]:
```
import erfa

def acc_i(x_i, date1, date2, C, S, GM, a_e, N):

  P = erfa.pnm06a(date1, date2)
  E = erfa.era00(date1, date2)  #Earth rotation angle at date
  R = erfa.rz(E,P)  #Final Rotation matrix
  x_b = np.dot(R, x_i)  #position in the body-fixed RF
  a_b = acc_bf(x_b, C, S, GM, a_e, N)

  return np.dot(np.transpose(R), a_b)
```

Exercise 5

from the file *LAGEOS 1 - 2022 Initial Conditions.txt* we take the initial condition to propagate the orbit up to a specific date and we use the *pyerfa* library to convert the time from UT to TT (Terrestial Time) with *erfa.dtf2d* which encode date and time fields into 2-part Julian Date

In [10]:
```
# from the txt file
x0 = np.array([-3925648.12725143,    4994759.41318484,  -10562295.01282353])
v0 = np.array([709.82404964822,    5180.59677349323,    2200.47213474637])

date1, _ = erfa.dtf2d("UTC", 2020, 1, 1, 0,0,0)
date2, _ = erfa.dtf2d("UTC", 2022, 3, 9, 0, 0, 0)

deltaT = date2 - date1

y0 = np.concatenate((x0, v0), axis=None)
```

to propagate the orbit and to solve the differential equation we use *solve_ivp* by *scipy*

In [12]:
```
from scipy.integrate import solve_ivp
```

```
def f(t,y,GM, C, S, a_e, date, N):
  x_i = y[0:3]
  v_i = y[3:6]
  a_i = acc_i(x_i, date, t, C, S, GM, a_e, N)
  return np.concatenate((v_i, a_i), axis=None)

Y = solve_ivp(f, y0=y0, t_span=(0,deltaT), t_eval=np.arange(0,deltaT,0.01), args=(GM, C,
```