TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Mihkel Ehrlich 153674IAPM

PS-INSAR PROTSESSI REALISEERIMINE KASUTADES VABAVARALISI TEEKE

Magistritöö

Juhendaja: Juhan-Peep Ernits
Andreas Kiik (AS Datel)

Tallinn 2018

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Mihkel Ehrlich
10.01.18

Annotatsioon

Töö eesmärgiks on realiseerida PSInSAR protsess kasutades vabavaralist keelt ja teeke selleks, et protsessi oleks lihtne edasi arendada ning rakendada erinevates valdkondades.

Töö põhiprobleemiks on luua alternatiiv programmile StaMPS, mis kasutab keelt MATLAB, mille eest peab maksma suuri litsentsitasusid. StaMPS'i puhul on tegemist väga raskesti loetava ja edasiarendatava rakendusega. Näiteks puuduvad testid ja dokumentatsioon on ebapiisav. Lisaks pole toetatud moodsad satelliidid, näiteks Sentinelid, mille andmed on Euroopa Kosmoseagentuurist tasuta kättesaadavad. Arvatavasti StaMPSile nende andmete ametlikku tuge ei kirjutata, sest projekti edasiarendus lõppenud.

Töö tulemusena kirjutati programmi StaMPS PSInSAR protessi sammud ümber kasutades vabavaralist programmeerimiskeelt ja teeke: Python 3, NumPy ja SciPy. See lubab programmi käivitada ja edasi arendada MATLAB litsentsi omamata.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 64 leheküljel, 5 peatükki, 12. joonist.

// Abstract

Implementation of PSInSAR process with freeware libraries

The main goal of this thesis is to implement PSInSAR process using libraries that are free to use with the purpose of making it easier to customize and use in different applications.

The main problem of the current thesis is to make an alternative implementation to a program called StaMPS. StaMPS is written in the programming language MATLAB that needs expensive licenses to run and to develop in. StaMPS code is also nontrivial to read and develop further. The lack of documentation and tests in code contributes further to the complexity. In addition, it is programmed for satellites that are not in active use today and there is no good support for Sentinel satellites that the European Space Agency (ESA) maintains and provides data from, free of charge.

The main contribution of the current thesis is that the StaMPS PSInSAR process steps are rewritten in a programming language and libraries that are free: Python 3, SciPy and NumPy. In addition, scripts have been provided to use data directly from the format provided by ESA.

The thesis is in Estonian and contains pages of text 64, 5 chapters, 12 figures.

Sisukord

1 Sissejuhatus 7
1.1 Käesoleva töö eesmärk 8
2 Valdkonna ülevaade 9
2.1 Tehisavaradar - SAR 9
2.2 Interferomeetriline tehisavaradar - InSAR 10
2.3 Diferentsiaalne InSAR - DInSAR 11
2.4 Püsivpeegeldajate InSAR - PS-InSAR 11
2.5 Copernicus programm 12
2.5.1 Sentinel satelliidid 12
2.6 Kasutatav tarkvara 13
2.6.1 SNAP 13
2.6.2 StaMPS 14
2.7 Alternatiivne tarkvara 15
2.7.1 Doris 15
2.7.2 ROI_PAC 16
2.7.3 ISCE 16
2.7.4 Gamma 16
2.7.5 GIAnT 17
2.7.6 PySAR 17
2.7.7 SARPOZ 17
3 Püsivpeegeldajate leidmise realisatsioon Pythonis 18
3.1 Kasutatud tehnoloogiad 18
3.2 Juhised käivitamiseks 19
3.2.1 Sõltuvuste seadistamine 19
3.2.2 Sättefailid ja parameetrite selgitus 19
3.2.3 Cython'i teegi kasutamine 20
3.3 Pythonis realiseeritud StaMPS sammud 20
3.3.1 Andmete lugemine algfailidest 21
3.3.2 Faasimüra hindamine 21
3.3.3 Püsivpeegeldajate valik 22

3.3.4 Püsivpeegeldajate filtreerimine ... 22
3.3.5 Faasikorrektsioon ... 22
3.4 StampsReplaceri struktuur ... 22
3.4.1 Scripts kaust ... 23
3.4.2 Testide kaust ... 24
3.5 Üldine tööpõhimõte ... 24
3.5.1 Protsessiklassid ... 24
3.5.2 Main klassi kasutamine ja tööpõhimõte ... 25
3.6 StampsReplacer koodi kirjeldus ... 27
3.6.1 Klass CreateLonLat ... 28
3.6.2 Klass PsFiles ... 29
3.6.3 Klass PsEstGamma ... 36
3.6.4 Klass PsSelect ... 46
3.6.5 Klass PsWeed ... 53
3.6.6 Klass PhaseCorrection ... 57
3.7 Andmeskeem ... 58
3.8 Pythonis ja Matlabis ekvivalentse koodi loomise probleemidest interpoleerimise funktsiooni näitel ... 59
3.9 Jõudlustestid tehtud tööle ... 61
4 Edasiarenduse võimalused ... 66
5 Kokkuvõte ... 68
Lisa A: Interpoleerimiste vahelise võrdluse joonistamise kood ... 73

1 Sissejuhatus

Euroopa Kosmoseagentuur on aastatepikkuse töö tulemusel loonud hulga satelliidisüsteeme, mille andmeid saab kasutada mitmesugusteks analüüsideks. InSAR (interferomeetriline tehisavaga radar) [1], DInSAR (differentsiaalne interferomeetriline tehisavaga radar) [1] ja PS-InSAR (püsivpeegeldajate tuvastamisega interferomeetriline tehisavaga radar) [1] analüüsiks kasutatakse tänasel päeval laialdaselt Sentinel-1 satelliidi andmeid. Põhjuseks on andmete lihtne ja tasuta kättesaadavus.

Püsivpeegeldajate analüüsis kasutatakse andmete töötluseks Euroopa Kosmoseagentuuri eestvedamisel arendatavat programmi SNAP [2] ja kogumit MATLAB koodi, mis on koondatud projekti StaMPS (*Stanford Method for Persistent Scatterers*) [3]. SNAP'is on Sentinel-1 andmetega töötlemiseks laialdane tugi, kuna seda vabavaralist projekti toetab aktiivselt Euroopa Kosmoseagentuur ja programmi kasutajaskond. Teise programmi puhul on tugi aga nõrgem, kuna see pole loodud spetsiaalselt Sentinel-1 satelliidi andmete jaoks nagu SNAP ning arendus on lõppenud ja uuendusi enam välja ei anta. SNAP'i saab kasutada andmete eksportimiseks vormingusse, mida toetab StaMPS.

Hoolimata sellest, et andmeid on võimalik SNAP'ist eksportida StaMPS'i on viimase programmiga probleeme, mis vajaksid lahendamist. Üks probleem on see, et StaMPS kasutab programmeerimiskeelt MATLAB, mille puhul on kommertskasutuseks vägagi arvestatavad litsentsitasud.

Käesoleva töö panus on suunatud eesmärgile, et tulevikus saaks loodud tarkvaraga püsivpeegeldajaid leida stabiilsemalt kui StaMPS. Hetkel on probleeme, et StaMPS's liiga väikesel või liiga suurel alal ei suuda leida püsivpeegeldajaid. See tuleneb sellest, et StaMPS on kirjutatud teistele satelliitidele kui Sentinel ja StaMPS'i loodi eelkõige demonstreerimaks, et püsivpeegeldajate leidmine on võimalik.

1.1 Käesoleva töö eesmärk

Antud töö eesmärgiks on vabaneda kinnise platvormi – MATLAB'i, sõltuvusest. Seda tehakse programmi ümberkirjutamise teel programmeerimiskeelde Python. Töö realiseerimiseks kasutatakse abistavaid teeke NumPy ja SciPy.

Lisaks laheneb Python'it kasutades ka probleem, et StaMPS'il puudub tugi Sentineli satelliitide tugi, kuna SNAP toetab skriptimist Pythonis.

Käesolevas lõputöös luuakse Python'i vasted StaMPS'is tehtavatele püsivpeegeldajate leidmise protsessidele. StaMPS protsessi sammud on kirjeldatud peatükis 2.6.2. Enne protsesse optimeerima asumist on eelkõige oluline saavutada olukord, kus Python'i programm annab MATLAB'is kirjutatud programmiga sarnaseid tulemusi, et oleks võimalik jälgida matemaatiliste sammude järjestust.

Töös kasutatakse TDD (inglise keeles *Test-driven development*) lähenemist. See tähendab, et enne keele MATLAB koodi ümberkirjutamist tehti protsess läbi muutmata kujul. Nii saame tulemused, mida saame kasutada pärast, valideerimaks, kas ümberkirjutatud protsess töötab samuti nagu esialgne. Iga protsess valideeritakse vastu tema algse protsessi, mis oli implementeeritud StaMPS'is, tulemust.

2 Valdkonna ülevaade

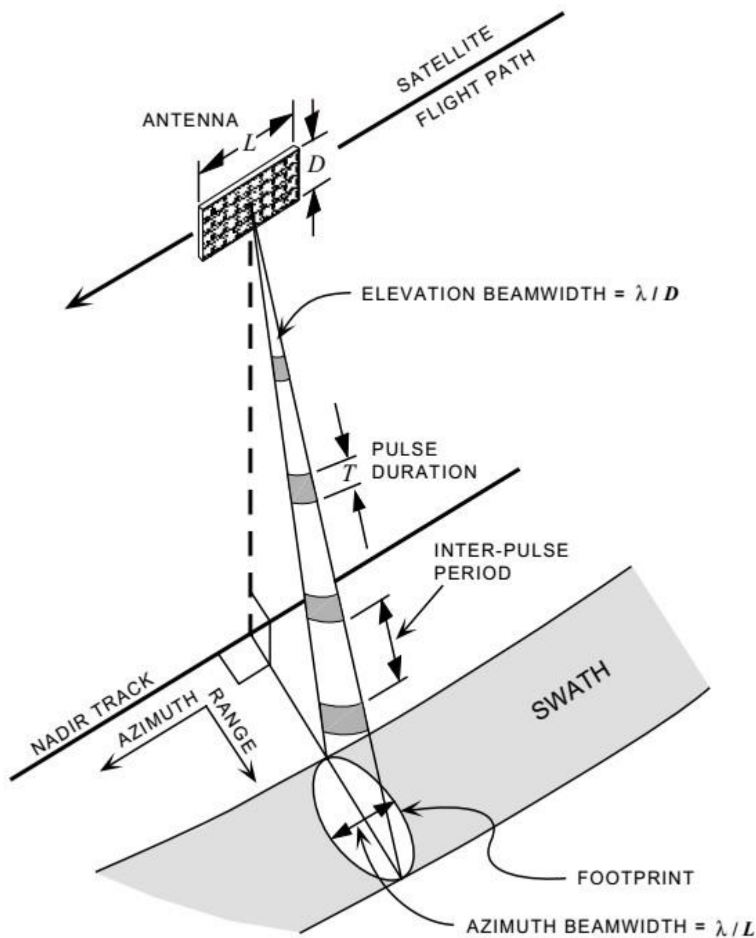
2.1 Tehisavaradar - SAR

SAR (*Synthetic Aperture Radar*) [1] [4] on kaugseire süsteem, millega saab luua kahe või kolmemõõtmelisi pilte seiratavatest aladest. Tavaliselt on seda tüüpi radar kosmoses satelliidil, kuid võib ka olla lennukil.

Tegemist on aktiivradariga, mis tähendab, et radari antenn on nii signaali saatja kui ka vastuvõtja. Tänu sellele saab pildi kätte ka öisel ajal. Mitte aktiivsed radarid kasutavad väliseid signaale nagu näiteks Päikse elektromagnet impulsse.

Radar kasutab mikrolaineid (lainepikkus 1 cm kuni 1m) ning tänu sellele ei mõjuta seda pilved ning taimestik.

Radari antenn saadab külgsuunas välja pulseeriva signaali ning võtab vastu saadetud signaali peegelduse [1]. Sedasi saab kokku panna SAR pildi. Joonisel 1 tähistab L antenni laiust ja D antenni kõrgust. λ tähistab lainepikkust. Antenni kõrguse ja lainepikkuse jagatis annab meile maapinnale langeva vaadeldava ala laiuse ja antenni laiuse ja lainepikkuse jagatis annab meile vaadeldava ala kõrguse. Asimuudiks nimetatakse satelliidi liikumissuunda mis on risti vaadeldava alaga.



Joonis 1. SAR tööpõhimõte [1].

2.2 Interferomeetriline tehisavaradar - InSAR

Tegemist SAR radari edasiarendusega, millega on võimalik leida maapinna deformatsioone [5]. Selle jaoks on vaja samalt alalt vähemalt kahte kohakuti asetsevat pilti, mille vahel võrrelda faasimuutu. Mida rohkem on pilte seda täpsem on tulemus. Kahe või enama faasipildi/ -nihke võrdlemist nimetataksegi inteferomeetriaks [4].

Selleks, et leida deformatsioone on olemas kaks levinud viisi [5]:

1. Punktide kaudu, mis muutuvad ajas ja erinevatel mõõtmistel võimalikult vähe ehk on koherentsed. Neid punkte nimetatakse püsivpeegeldajateks (inglise keeles *persistent scatterers*). Sellist protsessi nimetatakse ka PS-InSAR [1]. Selle protsessi kohta on lähemalt kirjutatud peatükis 2.4.
2. Tavalisel moel moodustatud interferogrammid huvipakkuvast alast ja siis nendest võetakse pöördväärtus ning läbi selle saab teada ajas muutuva deformatsiooni alast. Seda nimetatakse inglise keeles „*small baseline*“ viisiks.

2.3 Diferentsiaalne InSAR - DInSAR

Inglise keeles *differential InSAR* [1] [5]. Mõiste nimetus ise on eksitav kuna tavalise InSAR on ka võimalik leida alade kõrguse muutu. Sellisel juhul on InSAR protsessist eemaldatud topograafiline nihe, mis on väljaarvtatud digitaalse kõrgusmudeli abil. Sedasi saame me faasiinformatsiooni ala kohta, mis sisaldab maapinna deformatsiooni, atmosfääri ja sensorist tulevat müra.

2.4 Püsivpeegeldajate InSAR - PS-InSAR

Püsivpeegeldajate puhul [1] leitakse koosregistreeritud inteferogrammide kogumiku (inglise keeles „*stack*“) juures punktid, mis võimalikult vähe muutuvad läbi kõikide inteferogrammide või piltide. Kogumikus on enamasti üle viie pildi, et saada võimalikult täpne tulem. Selle jaoks võrreldakse ülempilti (inglise keeles nimetatakse „*master*“) teiste alampiltidega (inglise keeles nimetatakse „*slave*“). Muutumise all peetakse silmas faasi, mida vähem faas muutub. Seda nimetatakse koherentsuseks.

Koherentsust saab välja arvutada valemiga (1):

$$\gamma = \frac{\langle s_1 s_2^* \rangle}{\sqrt{\langle s_1 s_2^* \rangle \langle s_2 s_2^* \rangle}}, 0 \leq |\gamma| \leq 1 \tag{1}$$

Kus s_1 ja s_2 on, kaks kompleksarvu interferomeetria pildis, tärn tähistab kaaskompleks arvu ja sulud ⟨⟩ tähistavad ruumilist keskmistamist, mis tehakse läbi „libiseva aknaga“ [1].

Püsivpeegeldajateks võivad olla kivid, majanurgad, sambad jne [6]. Samuti on olemas tehislikke püsivpeegeldajaid, neid pannakse aladele, kus on loomulikke püsivpeegeldajaid on vähe, näiteks suured alad kus asustust on vähem.

Püsivpeegeldajate tehnikat kasutatakse StaMPS'is [6], et leida alade deformeerumist. Käesolevas töös kasutatakse programmi SNAP, et koosregistreerida mingi kogum pilte. Tähele tuleb panna, et SNAPis kasutusel olev algoritm võtab esimese pildi ülempildiks [7].

2.5 Copernicus programm

Euroopa Kosmoseagentuur on käivitanud programmi millega oleks võimalik saada täpseid, ajakohaseid andmeid meie planeedi kohta. Kõik andmed on inimestele tasuta kättesaadavad.

Copernicus programmi põhiosadeks on satelliidid, mida nimetatakse Sentinelideks [8]. Kõik Sentinelid suudavad vaadelda alasid pilvise ilmaga ja päeval ja öösel. Satelliite on planeeritud kuus.

2.5.1 Sentinel satelliidid

Sentinel-1 [9] on mõeldud maapinna ja mere seireks. Satelliidi andmetega on võimalik kaardistada ja vaadelda suuri maa-alasid (näiteks põllu-, metsa ja heinamaid [10]) maapinna deformatsiooni arvutada, teha õlireostuse seiret ja palju muud. Uus pilt saadakse iga kuue päeva tagant. Antud töös kasutatakse selle satelliidi andmeid.

Sentinel-2 [11] ülesandeks on maapinna, väiksemate veekogude ja taimestiku seire. Taimestiku all mõeldakse nii metsa kui ka põllumajanduslike taimi, et jälgida nende kasvu ja kaardistada maa-ala. Väiksemate veekogude all mõeldakse järvi ja jõgesid. Seda selleks, et leida võimalikke saastusi ja üleujutusi. Maapinna all saab mõelda pinnase erosiooni ja vulkaanide vaatlust. Seda kõike on vaja selleks, et võimalikult kiirest hoiatada võimalikust loodus katastroofist (vulkaanipurse, üleujutus, põud jne).

Sentinel-3 [12] ülesandeks on ookeani, jää ja atmosfääri seire. Kuna seireulatus on 1270km siis satelliit võimaldab katta suuri alasid ning tervele Maale tehakse ring peale kahe päevaga siis andmed on alati värsked. Andmeid kasutatakse ilmaennustuseks, mere veetaseme monitoorimiseks, merereostustuste avastamiseks ja mereelustiku vaatlemiseks.

Sentinel-4 ja 5 [13] ülesandeks on anda informatsiooni atmosfääri, õhu kvaliteedi, osooni, päiksekiirguse ja kliima kohta. Tegemist on Meteosat kolmanda põlve satelliitidega. Sentinel-4 on varustatud UVN (*Ultraviolet Visible Near-infrared*) sensoriga

ja Sentinel-5 UVNS (*Ultraviolet Visible Near-infrared Shortwave*) sensoriga. Sentinel-5 on olemas ka P variant, mis peaks vähendama Envisat satelliidi kadumise mõju ning andma jätkuvalt värsket informatsiooni.

Sentinel-6 [14] on kliimasatelliit, mis peaks uurima jäävabu alasid, uurides kui hoovuseid, tuuli merel ning lainekõrguseid. See aitab ennustada paremini orkaane ning mõista hoovuseid. See peaks toetama Sentinel-3, kuna andmed mida saadakse on sarnased, aga Sentinel-6 on pigem täpsustav roll.

2.6 Kasutatav tarkvara

2.6.1 SNAP

SNAP *(Sentinel Application Platform*) [2] on Euroopa Kosmoseagentuuri (ESA, *European Space Agency*) arendatav rakendus satelliitide andmete töötluseks. Antud projektis kasutatakse seda interferogrammide moodustamiseks. Põhiliselt arendatakse rakendust Sentinelide jaoks, aga toetatud on ka teised satelliidid. Rakendus on jagatud tööriistakastideks (inglise keeles *toolbox*) ehk mooduliteks, kus on erinevate satelliitide tugi, mis ühenduvad SNAP ehk põhiprogrammiga.

Rakendust saab kasutada nii töölaua rakendusena kui ka käsurealt [15]. Lisaks on olemas Python liides, millega laieneb SNAP'i skriptimisvõimalus keelde Python. Rakendus on arendatud Java's, on avatud lähtekoodiga ning inimesed saavad sinna ise arendada juurde erinevaid funktsioone. Github salve link https://github.com/senbox-org/.

SNAP ja moodulid on vabalt allalaetavad ning paigaldatavad arvutisse. Küll aga tuleb mainida, et rakendus tahab väga palju arvutusressurssi, mida tavaarvutites enamasti tänapäeval ei ole. Lisaks sellele on satelliitide pildid enamasti 5 – 10GB suured.

Hetkel kasutusel olevas protsessis tehakse SNAP koosregistreerimine, interferogrammide moodustamine ja huvipakkuva ala väljalõikamine (inglise keeles

subset). Põhimõtteliselt tehakse kõik mis on võimalik SNAP'is, sest seal on hea tugi Sentinel-1 andmetega töötamiseks ja rakendus on enamasti stabiilne.

2.6.2 StaMPS

StaMPS (*Stanford Method for Persistent Scatterers*) on rakendus millega on võimalik leida maapinna deformatsioon kasutades püsivpeegeldajaid. Seda funktsionaalsust eelmainitud rakenduses, SNAP, ei ole. Samal ajal pole see aga tehtud Sentinelide jaoks. SNAP suudab küll Sentinelide andmestiku eksportida programmi StaMPS.

StaMPS uusim versioon on hetkel 3.3b1 ning see on välja antud 2013 aastal [3].
Rakendus on vabalt saadav, küll aga kasutab programmeerimiskeele MATLAB teeke, mis on tasulised.

StaMPS töötlus koosneb kaheksast sammust [16]:

1. Loetakse sisse andmed ning konverteeritakse andmed sobivaks järgnevate protsesside tarbeks.
2. Faasimüra hinnang (*Estimate phase noise*). Iga piksel analüüsitakse, et leida mis on tema faasimüra.
3. Püsivpeegeldaja valik (*PS selection*). Eelneva protsessi alusel valitakse välja kõige vähem mürarikkad pikslid. Nendest saavad püsivpeegeldajad.
4. Püsivpeegeldajate filtreerimine (*PS weeding*). Filtreeritakse valitud püsivpeegeldajaid, kui need on liiga mürarikkad või sattusid püsivpeegeldajate sekka, kuna asetsesid tugeva püsivpeegeldaja lähedal.
5. Faasikorrektsioon (*Phase correction*). Faas on parandatud vaatenurga vea alusel.
6. Faasi lahtipakkimine (*Phase unwrap*).
7. Vaatenurga vea arvutamine (*Estimate spatially-correlated look angle error*). Arvutatakse vaatenurga viga.
8. Veel kord kuues samm. Eelmises sammus saadud väärtused lahutatakse enne lahtipakkimist maha ja peale lahtipakkimist liidetakse juurde. Seda sammu korratakse seni kuni kõik usaldusväärsed interferogrammid on lahti pakitud.

Viimase sammu tulemusena peaks saama pikslite deformatsiooni, mida saab juba edasi töödelda SNAPis.

StaMPS'i kasutatakse selleks, et leida huvipakkuva ala deformatsioon. Paraku ei ole praeguses protsessis StaMPS väga stabiilne. Kui ala on liiga väike siis püsivpeegeldajaid ei leita. Ala suurus peab olema vähemalt $3km^2$. Paljudes olukordades, aga pole nii suurt ala vaja.

Kui ala on näiteks hõreda asustusega siis samuti püsivpeegeldajaid ei leita. StaMPS töötab paremini mägistel aladel ja linnades.

Lisaks peab vahel muutma sisendparameetreid vastavalt veateatele ning selleks, et veateateni jõuda, peab mitukümmend minutit StaMPS töötlema andmeid. Peale viga saab küll jätkata sealt sammust kust töötlus pooleli jäi, aga inimene siiski peab töötluse juures olema ning vajadusel sekkuma.

See kõik näitab, et StaMPS protsessi kasutamine on väga ajakulukas, ja mugavamaks kasutuseks peaks seda optimeerima. Nii seepärast, et protsess on ise aeglane, mis tuleneb osalt näiteks sellest, et valitud alad peavad olema ebavajalikult suured, kui ka seepärast, et kohati tuleb protsessi käigus ette vigu mida peab lahendama.

2.7 Alternatiivne tarkvara

Siin peatükis analüüsitakse alternatiivseid tarkvaralisi lahendusi püsivpeegeldajate leidmiseks. Uuritakse mis on nende võimalused ning kas neid saaks kasutada käesolevas töös.

2.7.1 Doris

Doris (*Delft object-oriented radar interferomtric software*) [17] on avatud lähtekoodiga tarkvara, mis on loodud SAR andmete analüüsiks. Puudub tugi Sentinel satelliitide andmete töötluseks. Viimane uuendus on 2009. aasta lõpust ning arvatavasti Sentinel-1 tuge ei tule.

Rakendus kasutab andmete esituseks oma formaati, mis on kirjeldatud dokumentatsioonis [18].

2.7.2 ROI_PAC

Repeat Orbit Interferometry PACkage ehk ROI_PAC [19] on tarkvara millega saab teha InSAR protsessi satelliitide andmete pealt. Tegemist on umbes sarnase tööriistaga nagu SNAP, tarkvara millega saab satelliidi andmetest teha inteferogramme. Tegemist on tasuta tarkvaraga mitte kommerts eesmärkidel kasutamiseks. Kirjutatud on programmeerimiskeeles Perl. Seda arendab WInSAR (*Western North America InSAR*).

Hoolimata sellest, et tööriista uuendati viimati 2009.-l aastal on sinna juurde tehtud liides, mis võimaldab teha inteferogramme ka Sentinel-1 andmetele [20]. Rakenduse andmete formaat on SLC.

2.7.3 ISCE

InSAR Scientific Computing Environment ehk ISCE [21] on samuti tarkvara SAR'I piltide töötlemiseks, et teha InSAR protsessi. Samuti uuendab ja arendab seda edasi WInSAR. Uuenduste järgi näib, et on ka tugi Sentinel-1 satelliitide jaoks. Kahjuks dokumentsiooni ei leia ning selleks, et seda allalaadida, peab olema WInSAR liige.

2.7.4 Gamma

Gamma Remote Sensing SAR and Interferometry Software [22] on tarkvara kogu SAR info töötlemiseks sarnaselt SNAP-le, aga on ka mõned lisavõimalused, näiteks püsivpeegeldajate analüüs, mis hetkel toimub programmi StaMPS abil.

Tarkvara on jagatud moodulitesse mida on kuus: SAR andmestiku protsessi moodul, interferomeetria/ InSAR protsessi moodul, DInSAR ja geokodeerimise moodul, punktide ja püsivpeegeldajate analüüsi (*Interferometric Point Target*) moodul, statistika, klassifitseerimine, filtreerimise ja eksportimise moodul ja maapinna geokodeerimise moodul eraldi neile, kes DInSAR protsessi ei kasuta.

Lisaks reklaamitakse Gammat kui kasutajasõbralikku vahendit, mida aga StaMPS näiteks ei ole. Tarkvaral on Sentinel-1 töötlemise tugi.

Tegemist on tasulise tarkvaraga ning seepärast ei sobi käesolevas töös kasutamiseks, kuna siin otsitakse vabavaralist alternatiivi. Küll aga saaks selle abil võrrelda kui täpselt ja kui palju kiiremini või aeglasemalt töö käigus tehtud lahendus töötab.

2.7.5 GIAnT

Generic InSAR Analysis Toolbox [23] on Python teek millega saab teha inteferogrammide töötlust ja võrdlust aegridade näol (kasutatakse MInTS ja Timefn nimelisi algoritme). Programm on avatud lähtekoodiga ja kasutab Python versiooni 2.7, aga on piiratud võimalustega.

2.7.6 PySAR

PySAR [24] on Python (versioonis 2.7) keeles tehtud teek nagu GIAnT, aga oluliselt rohkemate võimalustega. PySAR kasutab ROI_PAC tehtud faili. See tähendab, et otse SNAP'ist eksportida ei saa. PySAR võimaldab leida pikslite koherentsust, välja arvutada lahti pakkimata faasivigu, töödelda interferogramme, arvutada välja aegridu jne.

Tegemist on tasuta ja avatud lähekoodiga programmiga ning seda saaks kasutada tulevases töös. Paraku on tegemist samuti mittetäieliku tööriistaga olemasoleva vahendi muutmine on kohati keerukam kui uue tegemine. Lisaks on käesolevas töös eelistus realiseerida programm Python'i 3.-t versiooni kasutades, et oleks võimalik pruukida tüübiannotatsioone. Lisaks on Python'i arendajatel plaan lõpetada Pythoni versioon 2 toetamine alates aastast 2020 [25] ja seetõttu võimaldab kohe Python 3-s arendamine tarkvara teha tulevikukindlamaks.

2.7.7 SARPOZ

SARPOZ [26] (*The SAR PROcessing tool by periZ)* on tasuline tarkvara mis peaks asendama StaMPS. Kirjutatud on see programmeerimiskeeles MATLAB ning lähtekood avatud ei ole, aga antud keeles saab teha lisamooduleid sellele programmile. See, et

SARPOZ on MATLAB kirjutatud tähendab, et vaja on selle keele litsentsi, et see ka töötaks.

Kuna SARPOZ on tasuline ning nõuab toimimiseks MATLAB litsentsi siis otseselt sellega protsessi asendada ei saa. Küll aga saaks seda kasutada valideerimiseks, kas uuesti tehtud protsess töötab õigesti nagu oli ka Gamma puhul.

3 Püsivpeegeldajate leidmise realisatsioon Pythonis

Järgnevas kirjeldatakse magistritöö raames loodud programmi ja selle jõudlust võrreldes StaMPS-ga. Kirjeldatakse kasutatud tehnoloogiaid, kuidas programmi käivitada ja seadistada, millised sammud programmist StaMPS ümber kirjutati ning mida need teevad ja milline on projekti struktuur ning kuidas ümberkirjutatud protsessid töötavad.

Lähtekood asub GitHub salves aadressil https://github.com/Vants/stampsreplacer.

3.1 Kasutatud tehnoloogiad

Lõputöö on kirjutatud keeles Python (versioon 3.6.3). Põhilisteks abistavateks teekideks olid SciPy ja NumPy (versioon 1.13.3).

NumPy on väga populaarne teek MATLABis arendatud rakenduste realiseerimiseks Pythonis. Internetis leidub arvestataval hulgal õpetusi, kuidas realiseerida MATLABis kirjutatud programmi Pythonis kasutades NumPyt. Lisaks on abiks, et ka paljud funktsioonid töötavad samamoodi nagu MATLAB omad [27] [28].

Kõik sõltuvused mida projekti kävitamiseks läheb vaja on failis „env.txt“.

Töös ei ole kasutatud eelmainitud rakendusi PySAR ja GIAnT. Põhjusena, et nende Python'i põhiversioon oli 2, mis ei ole ühilduv loodud rakenduses kasutatava versiooniga. Lisaks oli käesoleva töö TDD lähenemise juures oluline järgida StaMPS'i arvutusprotsesside struktuuri ja seetõttu tehti valik kogu protsess nullist realiseerida.

3.2 Juhised käivitamiseks

Kõige värskem juhend, kuidas rakendus käivitada, asub git salves failis README.md.

3.2.1 Sõltuvuste seadistamine

Kõik sõltuvused, mida läheb programmi käivitamiseks vaja, on failis „env.txt“. Neid saab laadida Python'i virtuaalkeskkonda *virtualenv* käsuga *„pip install -r env.txt“*[1]. Selle käsu puhul peab vaatama, et oleks aktiivne virtuaalkeskkond kuhu seda paigaldatakse ja et terminal viitaks sinna kus on „env.txt“ fail. Lisaks peab kasutama *virtualenv* mitte Araconda/ *conda* virtuaalkeskkonda, sest sõltuvuste fail on tehtud esimese jaoks.

Lisaks on sõltuvuste fail Inteli MKL [29] teegi toega Python'i jaoks. MKL teek on optimeeritud kasutama Intel'i protsessorites leiduvaid vektorinstruktsioone ja mõnedes rakendustes annab tuntavat jõudluslisa. Seetõttu oli huvitav katsetada, kas ka käesolevas rakenduses õnnestub jõudluses puhtalt teegi vahetusega võita. Intel'i kompilaatoriga kompileeritud versioone NumPy ja SciPy teekidest ei pruugi, aga *pip*'i kaudu leida. Selle jaoks peab need alla laadima aadressilt https://www.lfd.uci.edu/~gohlke/pythonlibs/ (Windows operatsioonisüsteemile) ning paigaldama käsitsi.

3.2.2 Sättefailid ja parameetrite selgitus

Põhiline parameetrite fail asub StampsReplacer\resources\properties.ini.sample. See tuleb kopeerida ja kustutada lõpust „.sample.“

Parameetrid failis on järgnevad:

- path - Algfailide asukoht
- patch_folder - Kui path kaust on veel mingis kaustas (tmp on üpris levinud) siis tuleb sinna see ka panna.
- geo_file - .dim fail mida kasutatakse töötluses
- save_load_path - Salvestustee. Koht kuhu tulemused (.npz failid) salvestatakse

---

[1] Mõnes operatsioonisüsteemis, näiteks Ubuntus, on Python 3 toetav pip nimega pip3.

- rand_dist_cached - Kas juhuslike arvude massiiv loetakse vahesalvestusest või mitte. Vähendab oluliselt PsEstGamma töötluse aega. Kui tegemist on uute andmetega siis peaks enne vahesalvestatud faili ära kustutama. Failid asuvad asukohas <save_load_path>\tmp.

Testklasside jaoks on oma sätete fail. Asukohast StampsReplacer\tests\resources tuleb kopeerida „properties.ini.sample" fail ja kustutada lõpust „.sample".

Erinevused mis on testi parameetrite failis ja päris tööks mõeldud failis on:

- parameeter „path" on ümbernimetatud tests_files_path
- parameetritest puudub rand_dist_cached, testides on see väärtus koguaeg „True".

Tähele tuleb panna, et kõik kataloogiteed peavad sätefailides olema absoluutteena.

3.2.3 Cython'i teegi kasutamine

Cython võimaldab kompileerida Python'i koodi C koodi. Selle jaoks kasutatakse Cython teeki. Kompileeritud kood peaks olema kiirem kui mittekompileeritud Python. Kompileerimine tehakse Python'i skriptis cython_setup.py, kus saab lisada, mis faile kompileeritakse. Hetkel on määratud kompileerimiseks kõik klassid, mis on scripts kaustas väljaarvatud kaust internal. Põhjusena, et neid klasse kasutatakse vähem ja ei ole nii kiiruskriitilised.

Kompileerimiseks tuleb käsurealt anda ette järgnev käsk:

python cython_setup.py build_ext –inplace

Selle edukal lõpetamisel kasutab Python juba C faile iseseisvalt.

3.3 Pythonis realiseeritud StaMPS sammud

Antud töös on kirjutatud keelde Python ümber sammud, mis on seotud püsivpeegeldajate leidmisega. Need on siis sisendandmete töötlus, faasimüra

hindamine, püsivpeegeldajate valik, püsivpeegeldajate filtreerimine ja faasi korrektsioon.

Järgnevates alampeatükkides on selgitatud, mida need sammud teevad.

3.3.1 Andmete lugemine algfailidest

Enne töötlust luuakse algfailid (mis meie puhul tulevad programmist SNAP) [3].

StaMPSis asub andmete laadimise funktsioonid failis „ps_load_initial_gamma.m“. Algfailid mida loetakse ja töödeldakse on „pscands.1.ph“, „pscands.1.ij“, „bperp.1.in”, “day.1.in”, “master_day.1.in”, “pscands.1.ll”, “pscands.1.da”, “pscands.1.hgt”, “look_angle.1.in”, “heading.1.in”, “lambda.1.in”, “calamp.out”, “width.txt“, „len.txt“, „slc_osfactor.1.in“. Protsessi lõpuks leitakse muutujad *ij, lonlat, bperp, day, master_day, master_ix, n_ifg, n_image, n_ps, sort_ix, ll0, calconst, master_ix, day_ix*. Need kõik salvestatakse MATLAB’i faili „ps1.mat“.

Lisaks „ps1.mat“ failile salvestatakse muutuja *D_A* eraldi faili „da1.mat“. Massiiv *D_A* saadud lugedes faili „pscands.1.da“ sisu ja sorteeritud *sort_ix* järgi.

Sarnaselt eelmisega on tehakse ka massiiv *hgt*, mis loetakse failist „pscands.1.hgt”. Tegemist on kõrgusfailiga. See massiiv salvestatakse faili „hgt.mat“.

Leitakse ka massiiv *bperp_mat*, mis salvestatakse faili „bp1.mat“.

Eraldi salvestatakse ka massiiv *la*. Faili nimeks on „la1.mat“.

3.3.2 Faasimüra hindamine

Sammus hinnatakse iga piksli faasimüra [3].

StaMPSi realisatsioon asub failis „ps_est_gamma_quick.m“. Siin leitakse muutujad *ph_patch, K_ps, C_ps, coh_ps, N_opt, ph_res, step_number, ph_grid, n_trial_wraps, grid_ij, grid_size, low_pass, i_loop, ph_weight, Nr, Nr_max_nz_ix, coh_bins, coh_ps_save, gamma_change_save* ja salvestatakse see kõik faili „pm1.mat“. Nende

leidmiseks kasutati eelnevas sammus salvestatud MATLAB'i faile „ph1.mat“, „bp1.mat“, „la1.mat“, „la1.mat“, „pm1.mat“ ja „da1.mat“.

3.3.3 Püsivpeegeldajate valik

Eelmises sammus müra järgi valitakse pikslid, mis sobivad püsivpeegeldajateks [3].

Tegevus toimub koodifailis „ps_select.m“. Lõpptulemusena salvestatakse faili „select1.mat“, kus sees on muutujad *ix, keep_ix, ph_patch2, ph_res2, K_ps2, C_ps2, coh_ps2, coh_thresh, coh_thresh_coeffs, clap_alpha, clap_beta, n_win, max_percent_rand, gamma_stdev_reject, small_baseline_flag, ifg_index*.

3.3.4 Püsivpeegeldajate filtreerimine

Leitud püsivpeegeldejatest leitakse kõige tugevamad pikslid. Selle jaoks võrreldakse püsivpeegeldajate kõrval olevaid piksleid, ning valitakse välja kõige puhtama signaaliga pikslid [3].

Loogika on MATLAB'i failis „ps_weed.m“ ja seal tehakse tulemusfail „weed1.mat“ ja sinna salvestatakse tehtud muutujad *ix_weed, ix_weed2, ps_std, ps_max, ifg_index*.

Lisaks korrigeeritakse eelnevalt leitud muutujaid. Faasi müra hindamise sammus tehtud fail pm1.mat'i korrigeeritakse muutujaid *ph_patch, ph_res, coh_ps, K_ps, C_ps*. Seda tehakse filtreerimise protsessis leitud *ix_weed* massiivi järgi, mis näitab millised indeksid on sobivad ja millised mitte. Samamoodi filtreeritakse ka failis „la1.mat“ massiiv *la* ja massiiv *hgt* failis „hgt1.mat“

3.3.5 Faasikorrektsioon

Kokkupakitud faasist võetakse välja vaatenurga viga [3].

Tulemiks on „rc1.mat“, mille sees on massiivid *ph_rc, ph_reref*.

3.4 StampsReplaceri struktuur

Lühidalt kokkuvõttes on projekti kaustade struktuur järgnev:

- StampsReplacer
    - resources
    - scripts
        - funs
        - processes
        - utils
            - internal
    - tests
        - resources
        - scripts
            - processes

Projekti juurkaustas on kolm kausta. Kaustas „resources“ on sättefail ja sinna võib ka näiteks salvestada vahetulemused. Kaustas „scripts“ on Python keeles kirjutatud kood. Kaustas „tests“ on protsessi valideerivad testiklassid, testifailid ja testide konfiguratsioonifaili.

3.4.1 Scripts kaust

Kaustas „scripts“ on kogu loogika. See on jagatud omakorda kolmeks neljaks kaustaks.

Kaustas „funs“ on erinevad funktsioonid mis ei ole protsessispetsiifilised ja on kasutavad ka teistes protsessides. Hetkel on seal vaid klass *PsTopofit*.

Kaustas „processes“ on eelnevas peatükis kirjeldatud protsesside asukoht. Kogu protsesside toimimise loogika on nendes klassifailides. Kõik need klassid implementeerivad ülemklassi *MetaSubProcess*.

Abiklassid ja nendes paiknevad abifunktsioonid on kaustas „utils“. Tegemist on funktsioonidega, mis on tihedalt kasutavad ja aitavad hoida protsessides koodi lühikese ja puhtana, kuna peidavad endas natuke keerukamat loogikat. Seal on kolm klassi, *ArrayUtils, MatlabUtils ja MatrixUtils.*

ArrayUtils klassis on abifunktsioonid massiividega töötlemiseks. Funktsioonid on staatilised.

MatlabUtils klassis on implementeeritud MATLAB keele funktsioonid nii nagu need seal keeles töötavad. Mitte kõik funktsioonid ei tööta NumPy teegis nii nagu keeles MATLAB ja vahel pole ka teegis neid funktsioone, mis viimases. Seal klassis teisemini töötavad ja üldse puuduvad funktsioonid implementeeritud staatiliste funktsioonidena. *MatrixUtils* on abistav klass maatrikstüüpi (NumPy teegis klass numpy.matrix) järjenditega töötamiseks.

Lisaks eelnimetatud klassidele on seal kaust „internal", kus on pigem sisemiseks toimimiseks mõeldud klassid. Näiteks sättefailist lugemine, logimine, tavalised kaustade nimed, tulemuste salvestamine ja protsessi juhtimine.

3.4.2 Testide kaust

„tests" kaustas on kaks alamkausta: „resources" ja „scripts".

Esimeses on testide jaoks vajaminevad algfailid (see mis seadistatakse sättefailis „tests_files_path" parameetri alla) ja „properties.ini.sample" konfiguratsioonifail.

Teises on testid „scripts" kaustas olevatele klassidele. Testiklassid on seotud testitavaga kausta ja nime järgi. Kausta nimi peab olema sama mis testitaval klassil ja nimi peab olema sama mis testitaval, aga ees peab olema eesliides „test_". Näiteks klass *CreateLonLat* on kasutas scripts/processes siis selle testiklassi kataloogitee koos klassinimega on tests/scripts/processes/test_createLonLat.

3.5 Üldine tööpõhimõte

3.5.1 Protsessiklassid

Kõik protsessiklassid (klassid mis on kaustas *scripts/processes*) kasutavad ülemklassi *MetaSubProcess* mis on abstraktne klass. Seal klassis on abstraktsete funktsioonidega tehtud kõigis protsessides vajaminevad avalikud funktsioonid. Nendeks on protsessi alustamine (*start_process*), salvestamine (*save_results*) ja salvestatud andmete laadimine (*load_results*). Kõigis protsessiklassides on need funktsioonid

implementeeritud vastavad nende klasside vajadustele (näiteks mis muutujaid on vaja salvestada, mis muutujaid on vaja failist laadida ja mis protsessis teha tuleb). Tekkinud polümorfismi kasutatakse pärast ära klassis *ProcessHandler* (peatükk 3.5.2).

3.5.2 Main klassi kasutamine ja tööpõhimõte

Programm käivitatakse klassist *Main,* failist Main.py, kus on kaks parameetrit (sarnaselt programmiga StaMPS [16]). Esimene parameeter näitab millisest protseduurist alustatakse ja viimane näitab millisega lõpetatakse. Mõlemad parameetrid on täisarvud 0-ist 5-ni. Kui neid ei määra tehakse kogu protsess.

Parameetrite numbrid vastavad järgnevatele protsessidele:

0 - Programmiga SNAP loodud failide lugemine ja töötlemine (klass *CreateLonLat*).
1 - Algandmete laadimine. Loetakse ja konverteeritakse SNAP eksporditud failid, mis olid tehtud StaMPS programmile formaadiks, Python/ NumPy failideks (klass *PsFiles*).
2 - Faasimüra hindamine (klass *PsEstGamma*).
3 - Püsivpeegeldajate valik (klass *PsSelect*).
4 - Püsivpeegeldajate filtreerimine (klass *PsWeed*).
5 - Faasikorrektsioon (klass *PhaseCorrection*).

Kui parameetreid ei määra siis tehakse kõik protsessid. See käsk näeb välja selline:

python Main.py

See on võrdne järgnevaga, sest nagu eelnevalt parameetrite juures on kirjeldatud, algavad parameetrid 0-ist ja lõppevad 5-ega:

python Main.py 0 5

Järgnev käsk käivitab ainult esimese, SNAP failide lugemise ja töötlemise, sammu:

python Main.py 0 0

Lisaks saab teha nii, et ei määratle lõpp-parameetrit. Siis alustatakse töötlust sellest protsessist mis on näidatud (antud juhul on selleks *PsEstGamma*) ja teeb lõpuni:

python Main.py 1

Kui protsessi alustada mitte esimesest (parameeter 0) siis peab tähele panema, et eelnevalt tehtavate protsesside tulemused oleksid salvestatud kohta, mis on seadistatud konfiguratsioonifaili (parameeter save_load_path). Seda sellepärast, et eelnevalt leitud protsessi tulemeid läheb vaja järgnevates sammudes ning sellisel juhul need laetakse salvestatutest. Kui faili ei ole või algne failinimi, millega salvestati, on muudetud siis saab protsess erindi nimega *FileNotFoundError*. Näiteks eelneva käsu puhul peab vaatama, et *CreateLonLat* protsessi tulem oleks olemas ja õiges kohas.

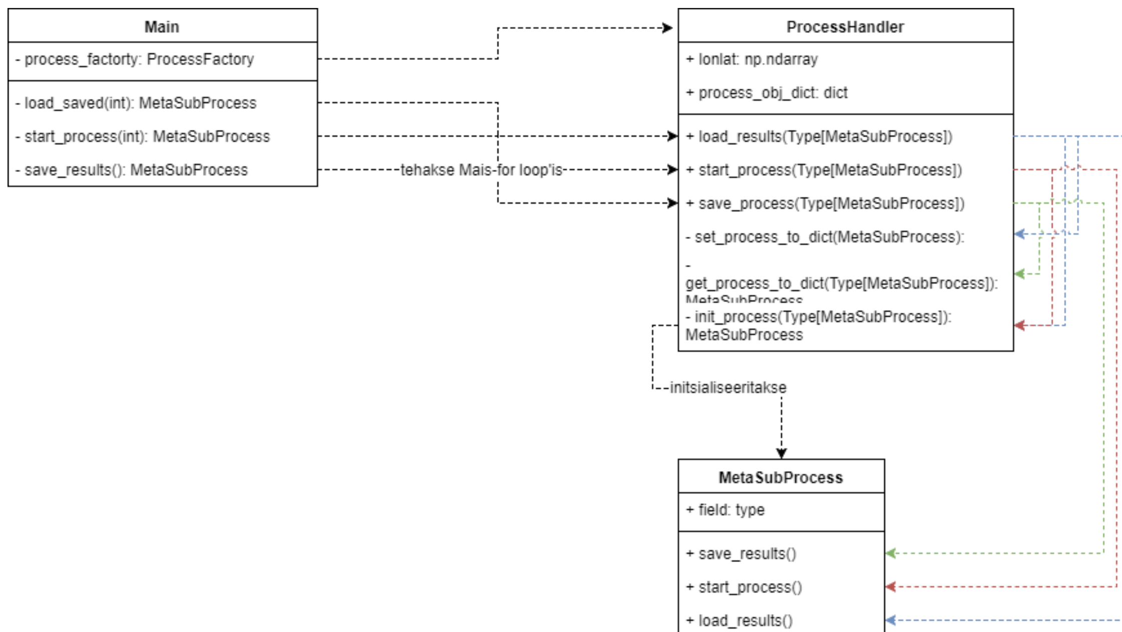
Käivitades Main klassi loetakse parameetrid failist propertes.ini mis asub „resources“ kasutas. Lisaks initsialiseeritakse klass *ProcessHandler*.

Kui *Main* klassis toimub töötlus, et mis protsessi klass on järgmine ja kas selle peaks laadima salvestusest või töötlema (start parameetri järgi) siis *ProcessHandler* vastutab selle eest, et protsessiklass saaks initsialiseeritud, töödeldud või laetud ja lõpuks salvestatud. Kuna kõik protsessiklassid pärivad ühte abstrakset klassi *MetaSubProcess*, kus on abstraktsete funktsioonidega need tegevused implementeeritud siis saab kõigile klassidele ühe funktsiooniga iga tegevuse teha, mitte igale protsessile oma funktsioon salvestamiseks ja laadimiseks. Klassis *ProcessHandler* on andmete laadimiseks funktsioon *load_results* ja salvestamiseks *save_process*.

Erandiks on protsess *CreateLonLat*, mis tagastab protsessi tulemusena massiivi, mitte ei toimi nii nagu teised, mis protsessi lõpus ei tagasta midagi vaid kõik töödeldud andmed salvestatakse klassimuutujatesse. *CreateLonLat* klassi jaoks on salvestamise jaoks *save_lonlat* ja laadimiseks *load_lonlat.*

Lisaks peab *ProcessHandler* klassis iga klassi jaoks eraldi initsaliseerimismeetodi tegema, sest need ei ole ühtsed kõigil. Põhjusena, et kõik klassid vajavad erinevaid algandmeid ja algklasse. Protsessiklasside loomine on implementeeritud funktsioonis *init_process.* Iga protsess pannakse peale tegevusi (kas siis tegemist oli salvestusest lugemisega või töötlemisega) Python sõnasikutüüpi (tüüp *dict*). Sealt on pärast, kas teiste protsesside initsialiseerimise või lõpptulemuse salvestamiseks, hea vastavaid objekte võtta.

Eelnevat seletatut aitab mõista joonis 2.



Joonis 2. Lihtsustatud joonis, kuidas Main klass kasutab *ProcessHandler* klassi.

3.6 StampsReplacer koodi kirjeldus

Järgnevates alampeatükkides on kirjeldatud magistritöö käigus loodud programmi koodi, mis on ümberkirjutatud StaMPS. Sellest ka nimi StampsReplacer. Kirjeldatakse, mida tehti erinevalt võrreldes MATLAB's tehtuga ja miks selliselt tehti.

Palju koodi sai muudetud tehnilistel põhjusel, et keeles MATLAB algavad massiivide indeksid 1-st aga Python's algavad 0'st. Paljud failid millest indekseid loetakse on aga tehtud MATLAB'i jaoks. Lisaks oli erinev ka see, et MATLAB'is on massiivid vaikimisi justkui veeruvektorid, NumPy's on nad justkui tavalised massiivid/ järjendid. Ehk need massiivid olid teisipidi. Lisaks oli palju muid asju, mille jaoks oli suureks abiks NumPy dokumentatsioonis leiduv leht, kuidas Malab'i koodi ümber kirjutada NumPy teegi abil (inglise keelne pealkiri *NumPy for MATLAB users*) [28]. Lisaks ei andnud mõningad MATLAB keele funktsioonid samu tulemusi, mis NumPy samanimelised funktsioonid. Need on funktsioonid on klassis *MatlabUtils* (kirjeldatud peatükis 3.4.1) ja kus ja kuidas neid kasutatakse on alampeatükkides eraldi välja toodud.

Järgnevalt kirjeldatavad koodifailid asuvad kaustas *scripts/processes*.

3.6.1 Klass CreateLonLat

See protsess pole otseselt programmi StaMPS osa. Küll aga tehakse seal edasistes protsessides vajaminevaid muutujaid *lonlat* ja *pscands_ij*. Originaalfail on TTÜ Tesla nimelise serveri peal ning faili nimeks on „create_lonlat.m“.

Protsessi initsialiseerimiseks on vaja failiteed, kus algfailid asuvad, ja SNAP'i produktifaili. Need leiab „properties.ini“ failist, esimene on parameetri „path“ all ja teine parameetri „geo_file“.

Tähele tuleb panna kus on PATCH_1 kaust. Kas see on seal samas kuhu näitab konfiguratsiooniparameeter või on see kuskil alamkaustas. Kui PATCH_1 on alamkaustas siis tuleb see lisada parameetrisse „patch_folder“. Vastasel juhul ei leita faili pscands.1.ij.

Tegemist on ainsa protsessiga, kus kasutatakse Snappy teeki. Snappy on teek, mis lubab Python'is kasutada programmi SNAP tagarakendit.

Antud juhul kasutatakse Snappy teeki selleks, et lugeda SNAP'i failidest pikslid. Selleks leitakse SNAP'i failist vertikaalsihis ja ristisihis lainepikkused (lon, lat), millest loetakse välja lõpuks pikslite väärtused ujukomaarvudena (float32), mis on kompleksarvu koefitsient. Fail mida loetakse on kirjutatud konfiguratsioonifaili parameetrisse „geo_file“.

Lisaks loetakse failist „pscands.1.ij“ sisse kõik read ning pannakse need muutujasse *pscands_ij*. Väärtusi ei töödelda mitte mingil kujul. Lõpptulemuseks on massiiv, mille veerus on kolm täisarvu (int32) ja mille pikkus sõltub ala suurusest, aga tavaliselt on see sadades tuhandetes.

Tähele tasub panna seda, et meetod millega saadakse pikslite väärtus, *read_pixel*, on väga aeglane kuna väärtusi loetakse ühekaupa. Kui antud meetodi saaks lugema mitme kaupa, näiteks ühes sihis kõik pikslid kõik korraga, oleks kiiruse vahe juba märgatav. Antud töö raames aga seda osa ei optimeeritud.

3.6.2 Klass PsFiles

PsFiles klassis jätkatakse sellega millega alustas *CreateLonLat*, ehk algandmete lugemisega failidest ja nende sättimisega klassimuutujatesse edasiseks kasutamiseks. Protsessi käigus täidetakse järgnevad muutujad *heading, mean_range, wavelength, mean_incidence, master_nr (*StaMPS'is *master_ix), bperp_meaned, bperp (*StaMPS'is *bperp_mat), ph, ll, xy, da, sort_ind, master_date, ifgs, hgt, ifg_dates (*StaMPS'is *oli day)*.

Protsessi initsialiseerimiseks on tarvis klassi konstruktorile anda failitee, kus asub PATCH_1 kaust ja *CreateLonLat* objekt kus on leitud muutujate *lonlat* ja *pscands_ij* väärtused.

Esimese parameetri puhul on failiteega, mis tekib konfiguratsiooni parameetrite *path* ja *patch_folder* liitmisel. Näiteks kui parameeter *path* on sätitud „C:\testi_andmed\test_1“ (jutumärkideta) ja *patch_folder* on „tmp“ siis konstruktori antakse tee „C:\testi_andmed\test_1\tmp“. Kui *patch_folder* jätta tühjaks siis antakse konstruktorisse kaasa see mis on konfiguratsioonis seadistatud *path* parameetriks. Lihtsalt peab vaatama kas PATCH_1 kaust on nendelt teedelt kättesaadav. Eelmiste näidete puhul siis peab vaatama kas PATCH_1 kataloogitee on „C:\testi_andmed\test_1\tmp\ PATCH_1“ või ilma „tmp“ nimelise vahekaustata .

Üldiselt asub kogu loogika failis „ps_load_initial_gamma.m“, aga palju lugemisi tehti ka teistes failides. Seetõttu autor otsustas, et võimalikult palju failidest lugemisi tehakse selles klassis. On üks fail mida loetakse klassis *PsWeed*. Seda seepärast, et seda kasutatakse ainult seal ja pole mõtet koguda kokku *PsFiles* klassis andmeid mida võib-olla ei kasutatagi, sest *PsWeed* on selles protsessis eelviimane samm. *PsWeed* protsessi kohta saab lähemalt lugeda peatükist 3.6.5.

Funktsioonis *load_params_from_rsc_file* olev loogika oli algselt faasi lugemise protsessis. Esmalt loetakse failist „rsc.txt“ failitee millest lugeda parameetreid. Sealt loetakse ainult teatud parameetrid ja need salvestatakse esialgu Python'i sõnastikku. Parameetrid mis loetakse Python'i on järgnevad: *azimuth_lines, heading, range_pixel_spacing, azimuth_pixel_spacing, radar_frequency, prf, sar_to_earth_center, earth_radius_below_sensor, near_range_slc, center_range_slc,*

date. Kui loetakse parameeter nimega *state_vector_position_1* siis lõpetatakse lugemine, sest peale selle parameetri pole enam huvipakkuvaid andmeid. Kogutud andmeid kasutatakse pärast teiste väärtuste leidmiseks ning mõned ka salvestatakse otse klassimuutujatesse, et pärast teistes protsessides neid kasutada.

Väärtused eelmisest funktsioonist leitud väärtused *heading* ja *center_range_slc* salvestatakse klassi muutujatesse heading ja mean_range. Tegemist on ujukomaarvudega. StaMPS'is salvestati väärtus „heading“ MATLAB keskkonna parameetritesse (kasutades funktsiooni *setparm*). Sellist võimalust Python keeles pole ja need salvestatakse klassimuutujatesse. Lisaks leiab autor, et selline stiil, et mõned muutujad on MATLAB'i parameetrites ja mõned .mat failides, raskendab loetavust.

Leitud signaali sageduse, *load_params_from_rsc_file* funktsiooni abil loetud parameetri *radar_frequency* järgi leitakse signaali lainepikkus. Selle jaoks jagatakse signaali levimise kiirus, 299792458 m/s, sagedusega. Leitud sagedus on GHz'ides. Välja arvutatud sagedus lisatakse klassimuutujasse *wavelength*.

Funktsioonis *load_ifg_info_from_pscphase* loetakse failist „pscphase.in“ kataloogiteed. Failist ei loeta esimest rida, sest see pole kataloogitee. Loetu salvestatakse klassimuutujasse *ifgs* ja muutuja tüübiks on NumPy massiiv (ndarray). Kataloogitees on nimetatud mis on masteri kuupäev ja mis on iteferogrammi kuupäev (kuupäeva formaat on YYYYMMDD). Näiteks kataloogitee on „C:\testifailid\test_1\ resources\export\diff0\20160614_20160509.diff“ siis siin on ülempildi kuupäevaks 14.06.2016 ja inteferogrammi kuupäevaks on 09.05.2016. Leitud järjendit, inteferogrammide failiteedest, kasutatakse pärast, et leida mitu interferogrammi on enne masteri kuupäeva (funktsioon *get_nr_ifgs_less_than_master*) ja et leida bperp ja bperp_meanded (funktsioon *get_bprep*).

Enne kui me hakkame leidma mitu interferogrammi on enne ülempilti, peab leidma selle pildi kuupäeva. Selle leiab funktsioon *get_master_date.* Leidmiseks kasutatakse *load_params_from_rsc_file* funktsioonis leitud muutujat *date*. Sõnastikus on kuupäev sõnena ja kujul „aasta kuu päev“, kus eraldajaks kaks tühikut. Selleks, et sellest saaks kuupäeva objekti teha (nimetus *date,* paketis *datetime*) on siis tuleb alguses sõne

jaotada eraldajate järgi (funktsioon *split*) ning anda õiges järjekorras ette *date* objektile. Õige parameetrite järjekord on aasta, kuu, päev.

Nüüd kus ülempildi kuupäev on teada, saame leida mitu pilti on enne ülempilti, mille kaudu leiame mitmes on ülempilt interferogrammide kogumikus. Selleks kasutame privaatset funktsiooni *get_nr_ifgs_less_than_master,* mille sees on kasutatakse avalikku funktsiooni *get_nr_ifgs_copared_to_master*, kus antakse võrdlusfunktsioon, mille järgi võrrelda. Nii tehakse sest funktsiooni *get_nr_ifgs_copared_to_master* kasutatakse ka mujal, aga võrreldakse teistpidi, mitu pilti on peale ülempilti. Aga selle asemel, et teha eraldi funktsioon selle jaoks antakse funktsioonile ette funktsioon kuidas võrrelda.

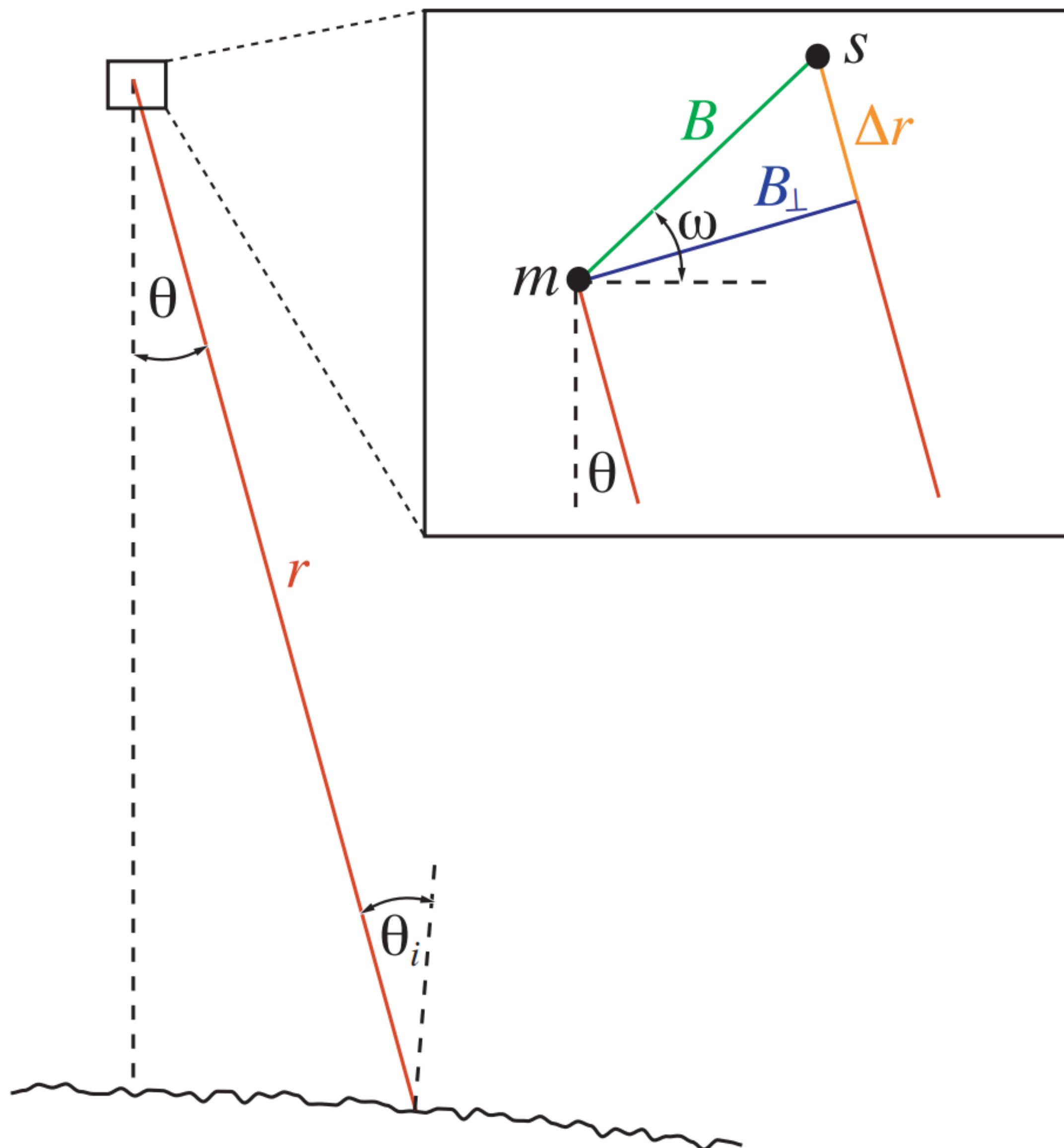
Peale seda leitakse samast järjendist, *ifgs*, millest eelmises sammus leiti mitmes pilt ülempilt, kõik interferogrammide kuupäevad. Selle jaoks kasutatakse funktsiooni *get_ifg_dates.* Võib tähele panna, et meetod on väga sarnane eelnevalt kasutatud funktsiooniga *get_nr_ifgs_copared_to_master*. Seda saaks refaktoreerida, et enne leitakse interferogrammide kuupäevad ja siis leitakse mitmes on ülempilt. Küll aga tuleb ümberkirjutamise juures tähele panna, et kohad kus seda funktsiooni kasutatakse jääks toimima.

Vahepeal leitakse muutuja *rg,* mis on NumPy massiiv. *CreateLonLat* protsessist leitud *pscands_ij* massiivist võetakse kolmas veerg (NumPy indeksite puhul on tegemist teisega). See korrutatakse läbi funktsioonist *load_params_from_rsc_file* loetud parameetriga *„range_pixel_spacing“* ja saadud tulemusele liidetakse juurde parameeter *„near_range_slc“*, mis on saadud samast kohast kust eelmine.

Seejärel leitakse muutuja *sat_look_angle,* ehk satelliidi vaatenurk. See arvutatakse välja *rsc* faili kaudu loetud parameetritest „sar_to_earth_center“ ja „earth_radius_below_sensor“. Leitud muutujat ei lisata klassimuutujatesse, aga seda kasutatakse hiljem teiste andmete leidmiseks.

Eelnevalt leitud arvu kasutatakse muutujate *bperp* ja *bperp_meaned* leidmiseks. StaMPS'is oli viimane nimetatud kui *bperp_mat*.

bperp tähistab paralleelset baasliini (inglise keeles *perpendicular/ parallel baseline).* See näitab kui palju muutub ülempildi ja alampildi satelliitide kaugus üksteisest (Joonis 3).



Joonis 3. m tähistab ülempildi satelliidi asukohta, s alampildi satelliidi asukohta. r on sensori ja maapinna vaheline kaugus. B tähistab baasliini ja $B_{\perp}$ paralleelset baasliini. Θ on nurk satelliidi ja vaadeldava objekti vahel (ehk vaatenurk, inglise keeles *look angle*) ja ω on nurk baasliini ja maapinna vahel [6].

Funktsioonist *load_ifg_info_from_pscphase* saadud interferogrammide kataloogiteed me kasutame, et leida failid muutujad *tcn* (on ka nimetatud kui „initial_baseline“) ja *baseline_rate*. See toimub funktsioonis *get_baseline_params.* Parameetrid loetakse failiteest, mis on interferogrammide massiivis, aga seda muudetakse natuke. Asendatakse olemasolev faililaiend „.base“. Sellest failist loetakse parameetrite

„initial_baseline(TCN)“ ja *„initial_baseline_rate“* väärtused. Nende abil leitakse muutujate *bc* ja *bn* väärtused. Valem (valem 2) mõlema leidmiseks on ühtne see on järgnev (koodis on see implementeeritud lambda avaldisena, nimega *bc_bn_formula*):

$$bc/\,bn = tcn + baseline_rate * \frac{(ij_lon - mean_azimuth_line)}{prf} \qquad (2)$$

Kus *tcn* ja *baseline_rate* on loetud funktsiooni *get_baseline_params* abil. *bc* jaoks kasutatakse massiivi esimest elementi nii *tcn*'ist kui ka *baseline_rate*'st (ehk siis tcn[1], mitte segi ajada nullinda elemendiga) ja *bn*'i jaoks teist. *ij_lon* tuleb massiivist *pscands_ij*, mis leiti protsessi *CreateLonLat* abil. Sellest massiivist võetakse esimene veerg. Valemis „mean_azimuth_line“ saadakse parameetritest, mis saadi funktsioonist *load_params_from_rsc_file,* kus selle nimi on *„azimuth_lines“.* Parameetrist saadud tulemus jagatakse kahega ja jagamise tulemusest lahutatakse 0.5. Muutuja *„prf“* saadakse samadest parameetritest, kust saadi eelmine väärtus.

Kui „tcn“ ja „baseline_rate“ on leitud saab leida *bperp* massiivi väärtused. Selleks kasutatakse valemit (3) mida kutsutakse välja *for*-tsüklis.

$$bperp = bc * \cos(sat_look_angle) - bn * \sin(sat_look_angle) \qquad (3)$$

Kus *sat_look_angle* on muutuja mille eelmises funktsioonis leidsime. Siinus ja koosinus väärtused vaatenurgast leitakse enne tsüklisse sisenemist. Nii leitakse need muutujad vaid korra ja see aitab hoida kokku arvutusaega.

bperp massiiv leitakse kõikidele iterferogammidele (muutuja *ifgs*) ja lõpptulemuseks on massiiv, mis on suurusega kõikide pikslite arv * interferogrammide arv.

Kui *bperp* massiiv on leitud siis saadakse igale massiivi reale keskmised (kasutades funktsiooni *numpy.mean*). Tulemuseks on massiiv mille pikkus on võrdne interferogrammide arvuga.

Peale keskmiste leidmise kustutatakse ülempildi veerg *bperp* massiivist ning funktsioonist tagastatakse *bperp* ja *bperp_meaned*.

Seejärel leitakse keskmine langemisnurk ehk incidence. See saadakse massiivist *rg* ja parameetritest muutujad „sar_to_earth_center“ ja „earth_radius_below_sensor“. Valem millega nurk leitakse on järgnev (valem 4).

$$incidence = arcos\frac{\text{sar_to_earth_center}^2 - \text{earth_radius_below_sensor}^2 - rg^2}{2 * \text{earth_radius_below_sensor} * rg} \quad (4)$$

Tähele tuleb panna, et antud koodis leitakse muutujate *sar_to_earth_center* ja *earth_radius_below_sensor* astme väärtused enne kui langemisnurk.

Leitud muutujast võetakse keskmine (funktsiooniga numpy.mean) ja tagastatakse.

Seejärel leitakse muutuja *ph*. See loetakse failist „pscands.1.ph“. Fail asub PATCH_1 kaustas. Tähele tuleb panna, et fail on binaarkujul ja seal on kompleksarvud. Selle jaoks antakse *numpy.formfile* meetodile parameetrina kaasa andmete tüüp. Tüübiks on >c8 mis on „big-endian“ 64bit kompleksarvud [30]. Kirjutamise hetkel tavapärast NumPy 128bit kompleksarvu sedasi laadida ei saa.

Seejärel leitakse *ll*, kasutades funktsiooni „get_ll_array“. See leitakse massivist *lonlat*, kus *lonlat* massiivi rea maksimaalne ja minimaalne liidetakse omavahel ja jagatakse kahega.

Siis leitakse *xy* ja sorteerimise järjekord (massiiv nimetusega *sort_ind*). See tehakse funktsioonis *get_xy*. *xy* massiivi jaoks võetakse nullis ja teine veerg massiivist *pscands_ij*. See on natuke teisem kui programmis StaMPS oli. Seal lisati esimeseks veeruks ka järjekorranumbrid. Aga siinkirjutaja leidis, et kuna seda kuskil ei kasutata ja kui kasutataks siis saaks selle leida läbi indeksi. See tähendab, et antud implementatsioonis massiiv *xy* on suurusega 2 * *pscands_ij* pikkus.

Kui *pscands_ij* veerust on esimesed kaks veergu selekteeritud ja pööratud peegelpilti, kasutades funktsiooni *numpy.fliplr*, korrutatakse nullis veerg ja esimene veerg konstantidega. Nullis veerg korrutatakse läbi 20’ega ja esimene veerg korrutatakse läbi neljaga.

Seejärel parandatakse *xy* väärtust keerates pilti suunda. Selleks kasutatakse klassimuutujat *heading*. Toimub see funktsioonis *screne_rotate*.

Kui see tehtud siis leitakse *sort_ind*. Selle jaoks kasutatakse NumPy funktsiooni *lexsort*. Sorteerimise järjekorraks on nullis veerg ja esimene veerg. *lexsort* tagastab massiivi indeksitest, mille järgi sorteerida. Selle järgi hetkel sorteeritakse *xy* massiiv ja see ka tagastatakse koos *sort_ind*'ga. Hiljem kasutatakse sorteerimise järjekorda funktsioonis *sort_results*, et sorteerida leitud massiivid mis on salvestatud klassimuutujatesse.

Funktsioon *get_da* on järjekorras järgmine. Sellega leitakse massiiv *da*. Massiiv *da* leitakse failist „pscands.1.da“ ning asub sama moodi PATCH_1 kaustas nagu teised „pscands“-nimetusega failid. Faili laadimiseks kasutatakse funktsiooni *loadtxt*. Kuna fail on ise suhteliselt väike ja ühe tulbaga siis suurt kiiruse erinevust teiste funktsioonidega väga palju ei olnud. Alternatiivseks lahenduseks oleks võinud kasutada funktsiooni *genfromtxt*, mis on kiirem.

Seejärel loetakse *hgt* massiivi andmed. See tehakse funktsioonis *get_hgt*. Selleks loetakse sisse fail „pscands.1.hgt“. Sarnaselt *ph* massiivi laadimisega on ka see faili binaarkujul. Andmetüübiks on >f4 mis on „big-endian“ 32bit ujukomaga arvud [30].

Kõige lõpus sorteeritakse leitud andmed. See tehakse funktsioonis *sort_results.* Sorteerimise jaoks kasutatakse sort_ind massiivi järjekordadest, mis saadi funktsioonist *get_xy.* Sorteeritakse järgnevad klassimuutujad: *ph, bperp, da, pscands_ij, lonlat, sat_look_angle* (nime alla *sort_ind*) ja *hgt*.

pscands_ij, lonlat ja sorteeritakse natuke teisemini kui teised, kuna need on maatriksid (*numpy.matrix*) mitte massiivid. Sorteerimine tehakse *MatrixUtils* klassis oleva funktsiooni *sort_matrix_with_sort_array* abil.

Klassimuutuja *sort_ind* on tegelikult sorteeritud *sat_look_angle.* Seetõttu salvestatakse see ka nii, klassimuutujatesse. *StaMPS*'is oli tegemist muutujaga la mis salvestati eraldi faili *la1.mat*.

Eraldi salvestati ka *bperp_mat*, mis meil on nime all *bperp*. Faili nimeks oli *bp1.mat*. Lisaks salvestati eraldi muutuja *ph*, nime alla *ph1.mat*.

3.6.3 Klass PsEstGamma

Järgimine samm protsessis on *PsEstGamma*. Selles sammus analüüsitakse võimalike püsivpeegeldajate faasimüra. Protsess on tehtud StaMPS'i faili „ps_est_gamma_quick.m“ järgi.

Protsessis täidetakse järgmised muutujad: *low_pass, weights, weights_org, rand_dist, nr_max_nz_ind, grid_ij, ph_patch, k_ps, c_ps, coh_ps, n_opt, ph_res, ph_grid, coherence_bins, nr_trial_wraps*.

Muutuja *coherence_bins* initsialiseeritakse klassi konstruktoris. Tegemist on massiiviga mis algab 0.005 ja lõppeb 0.995'ega, sammuks on 0.01. Seda kasutatakse pärast histogrammide juures ja need on histogrammide vahemikud.

Lisaks *coherence_bins* massiivile initsialiseeritakse konstruktoris ka sisemised parameetrid. Seda tehakse funktsioonis „set_internal_params“. StaMPS'is loeti need *setparam* funktsiooniga MATLAB'i keskkonda. Väärtused mis järgnevalt kirjeldatakse olid MATLAB'is failis „ps_params_default.m“: [16]

- filter_grid_size = 50. Piksli suurus meetrites. Kanditaatpikslid resampeldatakse ja pannakse maatriksisse/ võrku sellisete mõõtmetega enne faasi korrigeerimist.
- filter_weighting = „P-square“. Kaalude leidmise algoritm (*Weighting scheme*). Kanditaatpikslid korrutatakse läbi kaaludega resampeldamise protsessis vastavalt sellele algoritmile. StaMPS'is oli teiseks võimaluseks „SNR“, siin seda pole. Implementeeritud on vaid ja see mis seal on seadistatud.
- clap_win = 32. CLAP (*Combined Low-pass and Adaptive Phase*) filterdamise akna suurus. Koos *filter_grid_size* parameetriga näitab ala mida kasutatakse faasi analüüsimiseks.
- clap_low_pass_wavelength = 800. Filter lainepikkusele millest alates enam ei arvestata.
- clap_alpha = 1. Muutuja CLAP funktsiooni jaoks. Koos beta'ga annab jaotuse madalpääs ja adaptiivsete faasielementide suhtelisteks osakaaludeks CLAP filtri tegemisel.
- clap_beta = 0.3. Muutuja CLAP funktsiooni jaoks.

- max_topo_err = 5. Maksimaalne lubatud mitte korreleeritud digitaalse kõrgusmudeli (DEM) viga. Väärtus meetrites. Kui pikslite mitte korreleeritud kõrgusmudeli viga on suurem siis neid ei valita.
- gamma_change_convergence = 0.005. Vahe koherentsuse arvutamisel millest edasi enam koherentsust ei arvutata, kuna maksimaalne tulem on saavutatud.
- mean_range = 830000. StaMPS'is oli see nimetatud „rho". Kasutatakse, et leida max_k väärtus. See polnud vaikimisi muudetav, vaid loeti lihtsalt MATLAB'i keskkonda.
- low_coherence_thresh = 31. StaMPS'is oli nimetatud „low_coh_thresh". Selline oli väärtus kui small_baseline_flag oli „N". Kasutatakse, et välja arvutada muutuja „rand_dist" väärus funktsioonis „sw_loop". Sarnaselt eelmisele, polnud ka see parameeter MATLAB'is muudetav vaid loeti lihtsalt MATLAB'i keskkonda.

Protsessis esmalt leitakse massiiv *low_pass* funktsioonis *get_low_pass*. Tegemist on madalpääsufiltriga. See arvutatakse välja parameetritest *clap_win, filter_grid_size* ja *clap_low_pass_wavelength*.

Seejärel leitakse muutujad *PsFiles* protessist, mida pärast läheb tarvis. See tehakse funktsioonis *load_ps_params*. Kui midagi on vaja muutujatega teha, valida vastavad indeksid näiteks, siis see on see koht.

Esmalt võetakse *PsFiles* klassist muutujad. Selle jaoks kasutatakse *PsFiles* klassi funktsiooni *get_ps_variables*. Sealt tagastatakse muutujad *ph, bperp, nr_ifgs, nr_ps, xy, da*.

nr_ifgs muutujast lahutati maha üks. Aga StaMPS'is tehti seda vaid siis kui *small_baseline_flag* oli väärtusega „N". Siin protsessis ja kõikidest teistes on kood tehtud sedasi, et kõik kohad kus *small_baseline_flag* on „N".

ph massiivi samuti muudetakse. Massiiv mis tagastatakse on klassist *PsFiles ph* jagatud sama massiivi elementide absoluutväärtustega. Viimasel massiivi elemendid, mis on võrdsed nulliga asendatakse ühega, et ei tekiks nulliga jagamist.

Lisaks eemaldatakse bperm_meaned massiivist ülempildi rida. Selle jaoks kasutatakse funktsiooni *delete* NumPy teekist, kus eemaldatava rea indeks on *PsFiles* klassis olev *master_nr*. Viimasest parameetrist on lahutatud üks, et olla võrdne massiivi indeksitega. Lisaks tasub tähele panna, et siin me ei kasuta *MatrixUtils.delete_master_col* funktsiooni, sest kustutatavas massiivis on vaid üks rida ja *MatrixUtils*'is olev kustutab veergudest.

PsFiles klassist loetakse sisse ja muudetakse ka massiivi *sort_ind*, mis StaMPS'is oli „la1.mat“ failis olev muutuja *la*. Massiivi keskmistatakse kasutades NumPy funktsiooni *mean* ilma lisaparameetriteta. Sellisel juhul võetakse kõik massiivi elemendid ja võetakse nendest keskmine väärtus [31]. Keskmistatule liidetakse juurde kolm kraadi, mis on teisendatud radiaanideks. StaMPS'is teisendust ei tehtud, oli kood kirjutatud otse 0,052. Seetõttu on ümberkirjutatu selles osas natuke erinevate (täpsemate) tulemustega (võimalik näha testides). Tekkinud ujukomaarv salvestatakse muutujasse *sort_ind_meaned*. StaMPS'is oli see nimetusega *inc_mean*. StaMPS'is oli ka võimalus leida *inc_mean*, kui faili „la“ ei olnud. See leiti valemiga 21 * pi / 180.

Kui parameetrid *PsFiles* klassist andmed laetud siis leitakse *nr_trial_wraps*, mis on massiiv. Esmalt leitakse konstant *k* all oleva valemiga (valem 5). Selle tulemiga jagatakse läbi *max_topo_error parameeter* ja saadakse parameeter *max_k*.

$$k = \lambda * mean_range * \frac{\sin(sort_ind_meaned)}{4} * \pi \quad (5)$$

Valemis lambda on lainepikkus klassis PsFiles (muutuja *wavelength*), *mean_range* on privaatne parameeter mis sätiti konstruktoris ja *sort_ind_meaned* on parameeter mis tuli *load_ps_params* funkstioonist ühe tagastusparameetrina.

Seejärel leitakse *bperp_meaned* massiivist maksimaalse ja minimaalse vahe. See on nimetatud koodis kui *bperp_range*.

Kui see on leitud siis saabki välja arvutada *nr_trial_wraps* muutuja väärtuse. See leitakse *bperp* maksimaalse ja minimaalse vahe (äsja leitud muutuja *bperp_range*) korrutatud alguses leitud *max_k*'ga ja see jagatud kahe pi'ga. Lõpptulemuseks on murdarv (*numpy.float64*).

Seejärel leitakse klassimuutujad *rand_dist* ja *nr_max_nz_ind*. Programmis StaMPS oli *rand_dist* muutuja nimi *Nr*. r*and_dist* on juhusliku jaotuse massiiv. Teine on komaga arv, mis näitab mitu tükki juhuslikust massiivist ei ole nullid ehk on kasutatavad väärtused.

Kõigepealt tehakse juhuarvudest massiiv. Selle jaoks kasutatakse NumPy paketis olevat objekti *random*. See saadakse *numpy.random.RandomState*, kus parameetriks on juhuarvude seeme (inglise keeles seed). See on 2005. Sama arv oli ka StaMPS'is seemneks. Tähele tuleb panna aga, et juhuarvud mis MATLAB'is genereeritakse ei ole samad mis NumPys. Seega võivad tulla väikesed erinevused tulemustes.

Väike erinevus on ka juhusliku massiivi suuruses. StaMPS'is oli massiivi pikkuseks 300 000. Siin implementatsioonis on selleks pikslite arv ehk *nr_ps*, mis tagastati *load_ps_params* funktsioonist.

Kui massiiv juhuslikest on loodud siis selle järgi täidetakse juhuslike koherentsustega massiiv. Koherentsus leitakse *PsTopofit* funktsiooniga *ps_topofit_fun*, kus faas (parameeter *phase*) on väärtus juhuslikust massiivist, mis on tehtud imaginaararvuks ja siis võetud eksponent sellest (funktsioon *numpy.exp*). *PsTopofit* funktsioonist kasutatakse vaid teist tagastusparameetrit. See salvestatakse massiivi *random_coherence*. See tehakse *for*-tsüklis ning see tehakse nii mitu korda kui palju on ridu juhuslikus massiivis. See tähendab, et see on üpris ajakulukas protsess.

Seejärel tehakse histogramm loodud juhusliku koherentsuse massiivile, kus vahemikud tulevad muutujast coherence_bins, mis loodi klassi konstruktoris. Selle jaoks kasutatakse *MatlabUtils* klassis olevat funktsiooni *hist*. Põhjus on selles, et keeles MATLAB võetakse histogrammi vahemikud servadest, aga NumPy teegis ja Python'is

võetakse keskelt [32]. Funktsioon tagastab kaks parameetrit millest kasutame vaid esimest.

Histogrammi tulemus ongi muutuja *rand_distr* ja sellest vaadatakse mitu muutujat on mitte nullid (kasutades funktsiooni *numpy.count_nonzero*) ja sellest saab *nr_max_nz_ind*.

Funktsioonis *make_random_dist* on ka võimalus juhuslike arvude massiiv vahesalvestada ja salvestatu laadida Python'isse. See on tarvis seepärast, et massiivi loomine võib üpris kaua aega võtta ning arendusprotsessi kiirendamiseks sai teha selliselt. Vahesalvestatud faili nimi on „tmp_rand_dist.npz“ ja selleks, et vahesalvestust kasutada peab konstruktoris selle lubama (parameeter *rand_dist_cached_file).* Kasutaja juhib seda konfiguratsioonist parameetriga *rand_dist_cached*. Vahesalvestamisest on rohkem kirjutatud konfiguratsiooniparameetrite seadistamise peatükis (peatükk 3.2.2). Kui vahesalvestatud objekti ei leita siis *rand_distr* massiiv luuakse nii nagu eelnevalt kirjeldatud ning salvestatakse. Tähele tuleb panna, et kui vahesalvestatud juhumassiivi pole lubatud kasutada (sättefailis *rand_dist_cached* määratud „False“) siis seda ka ei salvestata. Vahesalvestuse loogika asub *make_random_dist* funktsioonis olevas sisemises funktsioonis *use_cached_from_file.*

Lisaks juhusliku jaotuse vahesalvestamisele ja salvestatu selle kasutamisele on võimalus ka juhuslik massiiv anda parameetrina. Seda saab teha konstruktoris andes sisendparameetrile *outter_rand_dist* soovitud massiiv. Seda kasutatakse testides, et anda ette MATLAB'is loodud juhuarvude massiiv ning siis valideerida protsessi lõpptulemusi StaMPS'i omadega. Lõppkasutajale see parameeter pole, aga nähtav ning seda hetkel kuskil seadistustest ette anda ei saa.

Kui juhusliku jaotuse massiiv on leitud, leitakse *grid_ij* massiiv. See leitakse kasutades *xy* massiivi. *grid_ij* on massiiv, milles on kaks veergu. Massiivi veerud on *xy* omadega vahetuses, see tähendab, et *grid_ij* nullis veerg on *xy* massiivi esimene veerg ja *grid_ij* esimene veerg tehakse *xy* nullindast veerust.

Veerud ei täideta lihtsalt *xy* andmetega. Esmalt arvutatakse kõik massiivi elemendid ümber (koodis on see funktsioon *fill_cols_with_xy_values*) . See tehakse valemiga 6.

$$\left\lceil \frac{x - \min(x) + 10^{-6}}{filter_grid_size} \right\rceil \quad (6)$$

Kus x tähistab massiivi xy veergu. Vähima leidmiseks kasutatakse NumPy funktsiooni *amin*, mis leiab massiivist kõige väikseima elemendi. *filter_grid_size* on konstant, mis väärtustati konstruktoris. Funktsiooni ümardatakse üles mille jaoks kasutatakse NumPy funktsiooni *ceil*. Tähele tuleb panna, et valemi nimetajas olev on koodis viidud tüüpi float32. Vaikimis on selleks float64, mis oli aga liiga täpne ning tekitas ümardamise käigus olukorra, kus vastus tuli ühe võrra suurem kui StaMPSis.

Selle tulemusest otsitakse NumPy *amax* funktsiooni kasutades element (või elemendid) mis olid kõige suuremad. Nendest elementidest lahutatakse üks.

Sellega lõppeb *grid_ij* veeru andmete leidmine. Lõpptulemusena tagastatakse massiiv milles on täisarvud (tüübiks *numpy.float32*).

Seejärel leitakse kaalud massiivist *da*. Selleks leitakse elementide väärtused. Seda tehakse NumPy funktsiooniga *divide,* kus jagatav, ehk esimene parameeter funktsioonis, on üks ja jagaja on *da* massiv. Saadud tulemus tehakse veerumaatriksiks, mille jaoks kasutatakse funktsiooni *to_col_matrix,* mis asub klassis *ArrayUtils*. See salvestatakse klassimuutujatesse nimega *weights_org*.

Seejärel leitakse *ph_patch, k_ps, c_ps, coh_ps, n_opt, ph_res, ph_grid* ja *low_pass* massiivid. Need kõik leitakse samas funktsioonis, *sw_loop.* Siin funktsioonis leitakse gamma muutus interferogrammidele. Seda tehakse *while*-tsüklis ja nii kaua kuni gamma muutuse absoluutväärtus võrreldes eelmisega on suurem kui parameetri *gamma_change_convergence* väärtus. Võrdlus tehakse funktsioonis *is_gamma_in_change_delta,* kus võrreldakse kas delta on väiksem eelmainitud klassimuutujast ja *while* tingimuseks on selle negatsioon (Python *not*).

Kõigepealt leitakse *grid_ij* massiivist maksimaalsed väärtused kasutades NumPy funktsiooni *max*. Need leitakse nullindast ja esimesest veerust ning salvestatakse vastavalt muutujatesse *nr_i* ja *nr_j*. Neid kasutatakse ph_grid massiivi kuju määramisel. Massiivi kuju on nr_i * nr_j * nr_ifgs, kus viimane on *load_ps_params* funktsioonis leitud üks parameetritest. See kuju salvestatakse muutujasse *PH_GRID_SHAPE*. See on justkui konstant muutuja, et seda leitud massiivi kuju saaks kasutada mitmete NumPy massiivide loomisel siin funktsioonis.

Näiteks kasutatakse seda *ph_grid* massiivi leidmisel mis tehakse sisemises funktsioonis *make_ph_grid.* Selleks võetakse järjest *grid_ij* massiivist indekseid kuhu väärtused panna, millest on lahutatud maha üks, et olla võrdne Python keele indeksitega. Väärtused tulevad funktsioonist *pydsm.relab.shiftdim*, kus esimene parameeter on rida *weights* massiivist. See tehakse *for*-tsüklis, millest tuleb grid_ij maatriksist võetavad indeksid ja massiivist *weights* võetav rida. Tsükli pikkus tuleb parameetrist *nr_ps*, mis tähistab pikslite arvu.

Teine koht kus *PH_GRID_SHAPE* kuju kasutatakse on massiivi *ph_filt* loomisel. See tehakse *make_ph_filt* funktsioonis. Seal kutsutakse *for*-tsüklis välja funktsioon *clap_filt*, kus *ph_grid* massiivist võetakse tsükli indeksiga teisest veerust elemendid. Tsükli pikkuseks on inteferogrammide arv ehk muutuja *nr_ifgs*.

Seejärel leitakse *ph_path* massiiv. Sarnaselt *ph_grid* leidmisega, kasutatakse ka siin *grid_ij* massiivi, et leida indeksid (sama moodi ka lahutatakse üks). Küll aga kasutatakse neid indekseid selleks, et selekteerida õiged elemendid massiivist *ph_filt*. Selekteeritud elementidele funktsioon *squeeze* võtab massiivist ära üherealised massiivid [33]. Saadud tulemus lisatakse tsükli *ph_patch* massiivi, kus reaindeks tuleb tsüklist ja veeru indeksid on inteferogrammide arvuni. See tähendab siis, et koodis on see kirjutatud selliselt *ph_patch[i, :nr_ifgs]*. Tsükli pikkuseks on interferogrammide arv.

Seejärel leitakse *PsTopofit* funktsiooniga *ps_topofit_loop* funktsiooniga *k_ps, c_ps, coh_ps, n_opt, ph_res* massiivid.

PsTopofit puhul luuakse kõigepealt objekt, mille käigus initsialiseeritakse massiivid, mis funktsioonist *ps_topofit_loop* tagastatakse. Funktsioonis kutsutakse *for*-tsüklis välja *PsTopofit* klassis olev funktsioon *ps_topofit_fun*. Selle tulemusena tagastatakse massiivid mille kaudu täidetakse eelnevalt mainitud tulemus massiivid. Tsükli pikkuseks on pikslite arv.

Kui *ps_topofit_loop* on lõppenud siis võetakse topofit objektist NumPy massiivi funktsiooniga *copy* tulemused ja salvestatakse need eraldi muutujatesse. Peale seda kustutatakse objekt, et säästa mälu. Ilma kopeerimiseta ei pruugi kustutamine täielikult õnnestuda, kuna viited objektile on veel alles.

Siis leitakse koherentsuse muutus. See arvutatakse välja valemiga 7.

$$gamma_change_rms = \sqrt{\sum \frac{(coh_ps - coh_ps_result)^2}{nr_ps}} \quad (7)$$

Kus ruutjuure, summa ja astendamise jaoks kasutatakse vastavalt NumPy funktsioone *sqrt, sum, power. coh_ps_result* on eelmise tsükli korra *coh_ps* muutuja tulemus. Valemi tulemusena tagastatakse murdarv.

Selle valemi tulemusest lahutatakse eelmise tsükli korra tulemus ja sellest saab *gamma_change_delta.* Selle tulemuse absoluutväärtuse järgi vaadatakse kas *while*-tsüklit võib jätkata. Kui võib siis enne uuesti alustamist tehakse veel lisategevused.

Lisategevustes leitakse uuesti kaalud ehk massiiv *weights*. Selle jaoks leitakse kõigepealt histogramm, kasutades *MatlabUtils* klassis olevat funktsiooni hist, mida kasutasime ka juhuslike jaotuste leidmisel. Funktsiooni esimene parameeter on *coh_ps* ja vahemikud on konstruktoris leitud *coherence_bins,* nagu ka *rand_distr* massiivi täitmise juures.

Funktsioon *hist* tagastab kaks parameetrit [32], aga siin on tarvis ainult jaotusi, ehk esimest parameetrit. Selle tulemusest võtame kuni massiivist read kuni reani

low_coh_thresh_ind (kaasaarvatud) ning summeerime need NumPy funktsiooniga *sum*. Sama teeme ka juhusliku jaotuse massiiviga, *rand_dist*. Need kaks summat jagame omavahel ja seejärel saame arvu millega korrutada läbi kogu juhusliku jaotuse massiiv. Programmi StaMPS koodifailist „ps_est_gamma_quick.m“, kust kogu see loogika pärineb, oli selle kohta kommentaariks kirjutatud, et juhuslik jaotus tehakse reaalseks.

Seejärel võetakse *hist* funktsiooni tulemus ja asendatakse kõik nullid ühtedega. Põhjusena, et see saab jagajaks (teiseks parameetriks) funktsioonile NumPy *divide*, kus jagaja on juhuslike jaotuste massiiv. Tegelikult pole tarvis siin kasutada NumPy eraldi jagamise funktsiooni, sest kui tegemist on massiividega siis on jagamine nagunii elementide vaheline tegevus [27]. Aga tehtud on see selliselt seepärast, et kui juhuslikult peaks kunagi muutma ühe neist massiividest maatriksiteks siis tavaline jagamine oleks juba maatrikstehe ja see toob kaasa juba vale tulemuse.

Sellist mustrit, kus StaMPS koodis oli elementide vaheline tehe (enamasti korrutamine või jagamine) siis on see selles tehtud NumPy elementide põhiste funktsioonidena (näiteks *divide, multiply*) tehtud, et andmetüüpi muutes massiivist (tüüp *numpy.ndarray*) maatriksiks (tüüp *numpy.matrix*) ei peaks koodi olulisel määral ümber kirjutama.

Jagatise tulemuses määratakse algusest kuni arvuni, mis on näidatud klassimuutujas *low_coherence_thresh*, üheks. Sama muutujat kasutati ka osas kus tehti juhuslik jaotus reaalseks. Seejärel pannakse samas massiivis indeksist, mis on määratud klassimuutujas nimega *nr_max_nz_ind*, kuni lõpuni elementideks nullid. Kõige lõpuks asendatakse kõik ühest suuremad väärtused ühega.

Seejärel lisatakse massiivi algusesse seitse number ühte algusesse juurde. Seda kasutatakse funktsioon *scipy.signal.lfilter* juures kolmanda parameetri juures. Esimeseks parameetriks tehakse Gaussi aken funktsiooniga *gausswin*, mis on klassis *MMatlabUtils*, mille suuruseks on 7. Funktsiooni teiseks parameetriks on list ühe elemendiga, arvuga 1. Funktsiooni tulemus jagatakse sama suure Gaussi akna summaga, kui anti funktsiooni.

Selle tulemusest võetakse massiivi elemendid alates seitsmendast elemendist, et massiivi suuruseks tekiks 100, ja sellega tegutsetakse edasi.

Seda tulemust kasutatakse interpoleerimise funktsioonis mis on järgmine samm. Interpoleerimine toimub funktsioonis *interp* ja mis asub klassis *MatlabUtils*. Funktsiooni sisendparameetriteks on eelmise sammus leitud massiiv mille algusesse on lisatud 1, teiseks parameetriks on 10, mis näitab mitu korda interpoleerida. Katsete tulemusena (peatükis 3.8) kasutame siin liigiks *„quadratic“*, kuna see oli täpsem kui alternatiivid. Funktsiooni tulemusest jätame kõrvale üheksa viimast elementi.

Seejärel tehakse *coh_ps* massiivist indeksite massiiv. Kuna *coh_ps*'is on väärtused nulli ja ühe vahel siis kõigepealt korrutatakse arvud tuhandega ja siis ümaradatakse, NumPy funktsiooni *round* abil. Massiiv elementide tüüp konverteeritakse täisarvudeks, kasutades NumPy *astype* funktsiooni. See on tarvis seepärast, et NumPy indeksid peavad olema seda tüüpi, aga coh_ps massiivis on komaga arvud. StaMPS'is oli ka lisaks ühe juurde liitmine tekkinud massiivile, aga Python keele indeksite eripära tõttu ei ole vaja seda teha. Enne kui indeksite massiiv võetakse kasutusel vaadatakse, kas massiivi mõõtmed on ühemõõtmelise massiivi oma. Kui ei ole siis tehakse kasutades NumPy *squeeze* funktsooni.

Seejärel loodud indeksmassiivi järgi leitakse massiivist, mis tuli interpoleerimise funktsiooni tulemusena, arvud. Need konjugeeritakse ja transponeeritakse.

Saadud tulemusest saadaksegi uued kaalud. Tulemustest lahutatakse arv üks ja astendatakse kahega (funktsiooni *numpy.power* abil). Seejärel viiakse massiiv õigele kujule kasutades *reshape* funktsiooni NumPy teegis. Kuju on muutujas *SW_ARRAY_SHAPE*, mis on nr_ps * 1. Tegemist on justkui massiiviga massiivis.

See peab selliselt olema seepärast, sest mõningad funktsioonid tagastavad taolised massiivid. Selleks, et nende massiividega tehteid teha peavad massiivid olema samade mõõtmetega. Kohati aga võtab rohkem nende massiivide kuju muutmine, mis funktsioonidest tagastatakse, aega rohkem kui ühe massiivi õigesse mõõtu tegemine.

Viimases, kaalude leidmise osas, tuleb tähele panna, et antud implementatsioonis on üks osa puudu. Programmis StaMPS oli eraldi loogika selle jaoks, kui parameeter *filter_weighting,* kaalude leidmise algoritm, on midagi muud kui „P-square“. Siinkirjutaja ei näinud põhjust, et seda oleks vaja antud implementatsioonis realiseerida. See oleks tekitanud liiga suurt lisatööd, eriti just testimise osas.

Lisaks tasub tähele panna, et see funktsioon on väga ajakulukas. Suurem osa ajast kulub *PsTopofit* klassis oleva funktsiooni *ps_topofit_loop* tegemisele. Seda põhjusena, et tegemist on suhteliselt suure tsükliga ning tehtavad arvutused *topofit_fun* funktsioonis ei ole kõige triviaalsemad. Nagu jõudlustestidest, peatükk 3.9, nähtub, et olnud selle funksiooni kiirendamisel abi ka Cython'ist. Lisaks ei andnud esialgsed katsetused Numba teegi [34] toel tsüklite paralleelseks muutmine kohest võitu.

3.6.4 Klass PsSelect

Protessiklassis *PsSelect* valitakse eelmise protsessi, *PsEstGamma*, tulemusest saadud andmete põhjal stabiilseimad. Protsess on tehtud StaMPS faili „ps_select.m“ järgi.

Protsessis leitakse muutujad: *coh_thresh, ph_patch, coh_thresh_ind, keep_ind, coh_ps, coh_ps2, ph_res, k_ps, c_ps, ifg_ind*.

Protsessi initsialiseerimisel antakse klassi konstruktorisse *PsFiles* objekt ja *PsEstGamma* objekt. Sarnaselt *PsEstGamma*’s olnuga tehakse ka siin privaatseid klassimuutujad, mis StaMPS’is sätiti süsteemi *setparam* funktsiooniga. Parameetrid seadistatakse funktsioonis „*set_internal_params*“ ja on järgnevad (vaikeväärtused ja selgitused) [16]:

- select_method = *DESINITY*. Mõnedes kohtades koodis saab eraldi seadistada, et kuidas selekteeritakse piksleid. Antud implementatsioonis kasutatakse selle jaoks andmetüüpi enum, kus on kaks võimalikku väärtust, *DESINITY* ja *PERCENT*.
- gamma_stdev_reject = 0. Kasutatakse, et filtreerida *coh_thresh_ind* massiiv. Massiivi jäetakse kõik väärtused mis on sellest väiksemad. Hetkel on implementeeritud ainult väärtus 0, mis tähendab, et filtreerimist ei toimu. Seda

dokumentatsioonis ei olnud kirjeldatud. Seetõttu on võimalik, et parameetrit enam ei kasutata.

- drop_ifg_index = *numpy_array([])*, ehk tühi NumPy array. Siia pannakse indeksid interferogrammidest, mida ei soovi protsessi võtta.
- gaussian_window = np.multiply(np.asmatrix(MatlabUtils.gausswin(7)), np.asmatrix(MatlabUtils.gausswin(7)).conj().transpose()). See originaalis tehti koodis sees ja siis salvestati süsteemi. Tegemist on Gaussi aknaga mida kasutatakse *clap_filt_for_patch* funktsioonis.

Lisaks on seal *slc_osf, clap_alpha, clap_beta, clap_win*, mis olid ka eelmises protsessis. Kui neid muuta siis nende väärtused peaksid olema mõlemas protsessis samad.

Pikslite valimise protsess hakkab parameetrite laadimisega objektidest, mis anti konstruktoris kaasa.

Kõigepealt laetakse muutujad objektist *PsFiles*. Selle jaoks kasutatakse funktsiooni *get_ps_variables,* nagu tehti faasianalüüsis. Küll aga siin protsessis ei kasutata parameetrit *nr_ps*.

Seejärel valitakse massiividest *ph* ja *bperp* välja kõik andmed, mis on ajaliselt peale ülempilti. See tehakse sisemises funktsioonis *filter_params_based_on_ifgs_and_master*. Seal tehakse kõigepealt NumPy *arange* funktsiooniga arvurida, mis algab 0'ist ja lõppeb numbriga, mis on muutujas *nr_ifgs*. Sellest massivist võetakse kõigepealt välja interferogrammid mida ei vaadelda. Selle jaoks kasutatakse NumPy funktsiooni *setdiff1d*. Mitte vaadeldavate interferogrammide indeksid on massiivis *drop_ifg_index,* mis sätiti konstruktoris. Selle tulemusest võetakse, sama funktsiooniga, välja ülempildi indeks, mis tuleb *PsFiles* objektist.

Seejärel valitakse interferogrammidid, mis ei ole ülempildi omad. Selle jaoks kasutatakse klassis *PsFiles* olevat funktsiooni *get_nr_ifgs_copared_to_master,* kus parameetriks on võrdlusfunktsioon, mis näitab, et võrrelda tuleb selliselt kus ülempildi kuupäev on suurem kui massiivis olevad väärtused. Funktsiooni tulemusest lahutatakse üks, et olla

võrdne indeksitega. Seda seepärast, et funktsioon tagastab arvu mitu inteferogrammi võrreldes ülempildiga, mitte otseselt indeksit. Saadud arvu kasutatakse, et lahutada maha arv üks kõikidest arvudest seal massiivis, mille väärtused on suuremad kui see leitud arv. Sellega täidetakse sisuliselt tühimik mis jäi ülempildi välja viskamisest. Saadud massiivi nimeks on *ifg_ind* ning seal on indeksid.

Seejärel valitakse massiividest *bperp* ja *ph* välja veerud, mis ei olnud ülempildi omad. Selleks tehakse massiiv indeksitest, kust on eemaldatud ülempildi indeks. See tehakse sarnaselt nagu filtreeriti indeksid *drop_ifg_index* massiivi abil, aga massiivi asemel kasutatakse *PsFiles* klassis olevat *master_nr* muutujat millest on lahutatud üks, et saada Python keeles kasutatav indeks.

Lõpptulemusena tagastatakse muutujad *ifg_ind, bpep* ja *ph* (viimased filtreeritult).

Seejärel leitakse massiiv *da* ja *da_max* funktsioonis *get_da_max*. Funktsioonist tagastatakse peale *da_max* muutuja ja *da*. Seda seepärast, et kui massiiv *da* on väiksem kui 10 000 elementi suur siis luuakse *da* massiiv seal funktsioonis. Tehakse massiiv, mis on täis ühtesid. Massiivi suurus on võrdne *coh_ps*'iga.

Kui *da* massiiv on suurem eelnimetatud arvust siis *da* massiiv sorteeritakse kasutades NumPy funktsiooni *sort.* Sorteeritakse veergude järgi, selle jaoks sätitakse parameeter *axis = 0*.

Peale seda leitakse muutuja nimega *bin_size*. Kui *da* massiiv on suurem kui 5000 siis saab selleks 10 000. Kui vähem siis 2000.

Seda kasutatakse selleks, et võtta sorteeritud massiivist elemente. Seda võetakse nii, et massiivi alguspunkt on *bin_size – 1*, lõpp on tagantpoolt *bin_size – 1* ja sammuks on sama *bin_size*. Ehk siis tekib rida *da_sorted[bin_size - 1: -bin_size - 1: bin_size]*. Sedasi tekib 388117 suurusest massivist massiiv mis, on 37 elementi suur. Selle algusse pannakse üks number 0 ja lõppu sorteeritust viimane element. See kõik pannakse kokku NumPy funktsiooniga *concatenate*. Konkatenatsiooni tulemus ongi *da_max*.

Viimasena võetakse *rand_dist PsEstGamma* klassist. See oli Stamps'is nimetatud *Nr_dist*.

Seejärel on kõik muutujad olemas ning me paneme need vaheobjekti nimega *DataDTO.* See parandab kaasa koodi loetavust, sest nii on kolme või nelja sisendparameetri asemel üks või kaks. DataDTO'sse panema ainult siin funktsioonis leitud muutujad.

Funktsiooni lõpus tagastame DataDTO objekti leitud muutujatega.

Seejärel leitakse murdarv *max_rand*, mis StaMPS'is oli nimetatud *max_percent_rand*. Seda tehakse funktsioonis *get_max_rand,* kuhu antakse parameetriteks massiivid *da_max* ja *xy*. Seal kasutatakse ka privaatset parameetrit *select_method.* Kui *select_method* on *DESINITY* siis leitakse see järgnevalt. *xy* massiivist võetakse suurim rida ja väikseim rida (kasutades *MatlabUtils* klassis olevaid funktsioone *max* ja *min*) ning leitakse nende vahe. Lahutamise tulemusena saadakse massiiv ja selle elemendid korrutatakse omavahel kasutades NumPy funktsiooni *prod*. Funktsiooni tulemus jagatakse 10^6. Selle tulemusena saadakse muutuja *patch_area*, mis StaMPS'i koodis olevate kommentaaride alusel peaks olema ala kilomeetrites. *max_rand* saadakse sellest kui see korrutatakse konstantiga, milleks hetkel on 20, ja jagatakse *da_max* massiivi pikkusega millest on lahutatud üks.

Kui aga valimismeetod ei ole DESINITY (SelectMethod enumi kohaselt saab selleks olla vaid PERCENT) siis *max_rand*'i väärtus on juba nimetatud konstant.

StaMPS'is oli eelmainitud konstant *setparam*'iga süsteemi laetud ning *PERCENT* ja *DESINITY* jaoks olid erinevad muutujad (vastavalt „*percent_rand*" ja „density_rand"). Siin aga on tehtud üks muutuja, kuna korraga saab vaid üks selekteerimise meetod olla valitud. Lisaks on see siin implementatsioonis tehtud kohaliku muutujana funktsioonis *get_max_rand*, sest mujal seda väärtust ei kasutata.

Seejärel leitakse *min_coh, da_mean ja is_min_coh_nan_array*. Need leitakse kasutades funktsiooni *get_min_coh_and_da_mean.* Leidmiseks kasutatakse *PsEstGamma* klassist massiive *coh_ps, coherence_bins ja rand_dist*. DataDTO objektist kasutatakse *da* ja *da_max* ning lisaks kasutatakse eelnevalt leitud *max_rand*'i. *is_min_coh_nan_array* näitab tegelikult, kas min_coh'i sees on väärtuseid mis on nan'id. Kuna seda kasutatakse mitmetes kohtades siis on parem see leida siin ja ühe korra.

Neid on vaja, et leida koherentsuse lävi ehk *coh_thresh* funktsioonist *get_coh_thresh*. StaMPS'is oli natuke teisemalt see tehtud, tagastati ka *coh_thresh_coffs* maatriks, mida siin ei tagastata. Seda põhjusena, et seda kasutati hiljem joonestamiseks, aga antud implementatsioonis sellist funktsionaalsust ei ole.

Juhul kui eelmisest funktsioonis muutuja *is_min_coh_nan_array* tõene siis *coh_thresh* massiivi pannakse üks väärtus, milleks konstant nimega *DEF_COH_THRESH* ja väärtuseks on 0,3.

Kui aga *is_min_coh_nan_array* on väär siis esmalt vaadatakse kui suur on *min_coh*. Kui see on tühi siis pannakse see sama tühi massiiv. Kui massiiv on täidetud siis leitakse koherentsuse piirmäära koefitsient (muutuja nimega *coh_thresh_coffs*) mis leitakse NumPy funktsiooniga *polyfit*. Selle massiivi nullinda elemendi järgi vaadatakse, kas tegemist on positiivse kaldega (ingise keeles *positive slope)* või mitte. Kui on siis leitakse NumPy *polyval* funktsiooniga *coh_thresh* massiiv, mis ka funktsioonist tagastatakse. Funktsiooni parameetriteks on coh_thresh_coffs ja teiseks parameetriks on massiiv *da*. Kui esimene element pole suurem nullist siis on *polyval*'i teine parameeter on arv 0,35.

Seejärel leitakse massiiv *coh_thresh_ind* äsja leitud massiivist *coh_thresh*. Seda tehakse funktsioonis *get_coh_thresh_ind*. Lisaks kasutatakse massiive *coh_ps* ja *ph_res* *PsEstGamma* klassist. Seal leitakse *coh_ps* massiivist elemendid, mis on suuremad kui *coh_thresh* massiivis olevad elemendid. Selleks kasutatakse NumPy funktsiooni *where,* millest saadakse indeksid, kus ülalnimetatud tingimus kehtib.

Juhul kui *gamma_stdev_reject* privaatmuutuja on suurem nullist tehakse lisategevusi. Küll aga on siin implementatsioonis need tegemata.

Kui eelnimetatud muutuja on null siis tagastatakse *where* funktsiooniga leitud massiiv indeksitest.

Seejärel leitakse massiiv *ph_patch*. See leitakse funktsiooniga *get_ph_path.* Siin nii ei tehta ja need salvestatakse kohalikult siia klassimuutujatena. Antud massiiv leitakse *for*-tsüklis, mille pikkuseks on *coh_thresh_ind* massiivi pikkus.

Enne tsükli algust tehakse tühi massiiv *ph_filt* ja leitakse *nr_i* ja *nr_j* mis toimivad indeksitena. Sarnaselt nagu *PsEstGamma* protessis. *PsEstGamma* protsessis leitud *grid_ij* massiivist leitakse maksimaalsed väärtused, nullinda veeru suurim väärtus saab *nr_i*'iks ja esimese veeru maksimaalne väärtus saab *nr_j*'iks.

Muutuja *ph_filt* tehakse kasutades NumPy funktsiooni *zeros*, mis teeb parameetriks antud kujuga massiivi nullidest [35]. Selle kuju jaoks kasutatakse *clap_win* parameetrit, mis sätiti funktsioonis *set_internal_params*, ja inteferogrammide arvu, *nr_ifgs* muutujat *DataDTO* objektist. Kujuks saab *clap_win * clap_win * nr_ifgs*. StaMPS'is oli tehtav massiiv nimetusega *ph_filt2*.

Tsükli sees leitakse *PsEstGamma grid_ij* massiivist elemente. Selekteerimiseks kasutatakse *coh_tresh_ind* massiivi, kus on indeksid, millest peab valima. Viimasest võetakse tsüklis järjest. Need salvestatakse muutujasse *ps_ij*.

Seda kasutatakse järgmises sammus, et leida *PsEstGamma* protsessis leitud massiivist *ph_grid* selekteeritav vahemik. Vahemiku indeksid leitakse kasutades privaatset funktsiooni *get_max_min*, kuhu parameetriteks läheb kaks väärtust. Esimene *ps_ij* rida millest on lahutatud üks ja teine maksimaalne lubatud suurus, ehk *nr_j* või *nr_i*. *ph_grid* massiivist selekteeritakse kahe telje järgi (massiiv on ise kolmemõõtmeline) ehk siis tekib kaks paari indekseid, *i_min, i_max* ja *i_min, j_max*. Nende järgi leitakse *ph_grid* massiivist väärtused ja salvestatakse *ph_bit* nimelisse muutujasse. Tähele tuleb panna,

et selekteerides tehakse koopia kasutades NumPy funktsiooni *copy.* Seda seepärast, et kui selekteeritud väärtuseid muuta siis muudetakse neid ka *PsEstGamma* objektis.

Sealt saame leida massiivi mille abil leida *ph_filt* massiiv. See leitakse funktsiooni *clap_filt_for_patch* abil. Selle parameetriteks on *ph_bit* ja *PsEstGamma* objektist saadud massiiv *low_pass.* See tehakse for-tüsklis mille pikkuseks on interferogrammide arv.

Selle tulemus ongi peale NumPy *squeeze* funktsiooni ja selekteerimist ph_patch.

Tähele tuleb panna, et tegemist on üpris ajakuluka protsessiga seepärast see tulemus vahesalvestatakse nagu näiteks *PsEstGamma* protsessis salvestati juhuslik jaotus. Salvestatakse see nime alla „tmp_ph_patch.npz“. Lisaks *ph_patch* massiivile salvestatakse ka *coh_thres_ind* massiiv. Viimane seepärast, et kontrollida kas vahesalvestatu on tehtud samade andmete alusel, kui see, mis on hetkel protsessis. Kuna siin kontrollitakse, kas protsessi tulemus on juba olemas, siis seda vahesalvestuse loogikat läbi konfiguratsiooni seadistada ei saa, nagu sai *PsEstGamma* oma.

Seejärel saadakse *PsTopofit* klassis oleva *ps_topofit_loop* funktsiooni uus koherentsus. Seda tehakse funktsioonis *PsSelect* klassi funkstioonis *topofit*. *ps_topofit_loop* parameetriteks antakse *DataDTO* objektis olev massiiv *ph* ja *bperp PsFiles* objektist. Mõlemast valitakse välja *coh_thresh_ind* massiivi alusel elemendid. Lisaks antakse üheks parameetriks eelmises funktsioonis leitud *ph_patch* massiivi.

PsTopofit protsessist tehakse koopia *coh_ps* massiivist. Kopeeritud massiivis asendatakse *topofit* tulemusena saadud *coh_ps* elemendid, kus asendatavate elementide indeksid tulevad *coh_thresh_ind* massiivist. See sama, *coh_ps*, massiiv ka tagastatakse funktsioonist. Lisaks sellele tagastatakse ka *PsTopofit* objekt.

StaMPS'is siin osas kirjutati eelmises protsessis saadud *coh_ps* massiivi elemendid üle.

Leitud uue *coh_ps*'iga tehakse uuesti funktsioonid *get_min_coh_and_da_mean* ja *get_coh_thresh.* Nende andmetega leitakse *keep_ind* massiiv, mis StaMPS'is oli

nimetatud kui keep_ix. Seal on indeksid, mis on protsessist väljavalitud, aga need ei ole veel püsivpeegeldajad.

Protsessi viimase sammuna salvestatakse leiud klassimuutujatesse. Teeme selle lõpus seepärast, et päris paljud neist leitakse mitu korda (coh_ps'i uuesti arvutamisega).

3.6.5 Klass PsWeed

PsWeed protsessis valitakse selgeimad pikslid, mis leiti eelmisest protsessis. StaMPS'is oli see failis „ps_weed.m".

Protsessi initsialiseerimisel antakse *PsWeed* klassi konstruktorisse tee PATCH_1 kaustale, *PsFiles, PsEstGamma* ja *PsSelect* objektid.

Konstruktoris initsialiseeritakse järgmised privaatmuutujad

- time_win = 730. Silumine. Ühik päevade arv.
- weed_standard_dev = 1.0. Starndarhälve piirmäär. Iga piksel mille standardhälve on sellest suurem eemaldatakse
- weed_max_noise = sys.maxsize. Piksli müra piirmäär. Iga piksel mille müra on suurem sellest eemaldatakse.
- weed_zero_elevation
- weed_neighbours

Eelmisega võrreldes ka parameeter *drop_ifg_index*, mille tähendus on sama.

Lisaks tehakse loetakse failist „psweed.2.edge" andmed. Seda tehakse siin, mitte *PsFiles* klassis, sest *PsWeed* protsess on ainus, kus seda kasutatakse ning tegemist on eelviimase protsessiga. Kui töötlus siia ei jõua siis *PsFiles* klassis raisatud aega ja ressurssi. Võimalus oleks ka *PsFiles* klassis laadida see „laisalt", ehk kui andmeid küsitakse siis laetakse.

Protsess algab, nagu eelnevad, eelmistest protsessidest andmete laadimisega. See tehakse funktsioonis *load_ps_params* ning lõpptulemusena tagastatakse *DataDTO*

objekt. See *DataDTO* ei oma seost *PsSelect*'is olevaga, peale selle, et see on sisemine klass ja selles hoitakse muutujaid.

Kõigepealt leitakse *PsSelect* objektis massiiv indeksitest *keep_ind*. Seda kasutatakse, et valida õiged pikslid massiividest *coh_thresh_ind*, *c_ps*, *k_ps, coh_ps*. Kui massiiv *keep_ind* on tühi võetakse lihtsalt eelnimetatud massiivid. Samas massiiv nimega *ph_res*, mis tuleb *PsSelect* objektist, selekteeritakse alati *keep_ind* järgi, ka siis kui viimane on tühi. Massiiv *keep_ind* salvestatakse DataDTO klassi nime alla *ind.*

PsFiles objektist võetakse muutujad *pscands_ij, xy, ph, lonlat, hgt*, mis, sarnaselt *PsSelect*'is tehtud loogikaga filteeritakse. Seekord tehakse selekteerimine massiiviga *coh_thresh_ind*. Lisaks võetakse *PsFiles* objektist muutujad *master_nr* ja *ifg_dates master_nr, ifg_dates, bperp_meaned, master_date*.

PsEstGamma objektist võetakse ainult *ph_patch* ja see filtreeritakse samuti *coh_thresh_ind* massiivi järgi. Leitu salvestatakse muutujasse *ph_patch_org*.

Kõik leitud muutujad pannakse *DataDTO* objekti kus neid pärast kasutatakse.

Tähele tuleb panna, et kui *coh_thresh_ind* massiiv on tühi siis paljud massiivid on samuti tühjad. See aga on veaolukord ja programmis tekib viga. Erindit programm ei tagastata, küll aga logitakse see hoiatusena. Kui algavad arvutused siis programm ei saa tööd jätkata.

Seejärel leitakse muutuja ij_shift, mis leitakse pscands_ij maatriksist ja coh_thresh_ind pikkusest. See toimub funktsioonis *get_ij_shift*.

Seda kasutatakse, et teha massiiv *neighbour_ind*. See tuleb funktsioonist *init_neighbours* ja lisaks eelnevalt leitud massiivile kasutatakse ka *coh_thresh_ind* massiivi pikkust. Tegemist on massiiviga, mis näitab püsivpeegeldajate naabrid. Algul täidetakse massiiv -1'tedega ja seejärel täidetakse õigete indeksitega. StaMPS'is täideti nullidega, aga kuna Python'is algavad indeksid nullist siis -1 on siin rakenduses kui tühi.

-1 on klassis privaatse muutujana *DEF_NEIGHBOUR_VAL*, et saaks lihtsalt kontrollida ka mujal funktsioonidest ja kui on tarvis muuta seda „tühja“ väärtust siis saab seda ühest kohast teha.

Seejärel leitakse massiiv *neighbour_ps* funktsioonis *find_neighbours*. Mis on NumPy massiiv, mis sisaldab endas teisi massiive. Eelmises funktsioonis leitud *neighbour_ind* massiivist selekteeritakse *ij_shift* järgi (kus ij_shift'i sees olevatest indeksitest lahutatakse üks). Saadud tulemust kontrollitakse, kas tegemist on tühja väärtusega, *DEF_NEIGHBOUR_VAL*'i järgi. Kui tegemist pole tühjaga siis pannakse see *neighbour_ps* massiivi.

Seda massiivi kasutatakse, et leida parimate pikslite arv. Funktsioon *select_ps* tagastab massiivi *selectable_ps* (StaMPS'is oli 'ix_weed'). *selectable_ps* on massiiv mis on täidetud tõeväärtus tüüpi elementidega (Python tüüp boolean). StaMPS'is oli massiiv täidetud väärtustega 1 ja 0, aga NumPy sellist massiivi käsitleb kui indekseid. Peale seda parandatakse/ täpsustatakse *selectable_ps* massiivi järgnevates funkstioonides.

Esmalt eemaldatakse (märgitakse *False*'ks) selectable_ps massiivis dublikaadid. Dublikaatide leidmiseks kasutatakse *xy* massiivi. Siin funktsioonis ei tagastata *xy* massiivi, mis on selekteeritud selectable_ps massiivi järgi kuna seda ei kasutatud pärast kuskil. Seda muutujat StaMPS'is nimetati „ix_weed_num“.

Seejärel tehakse interferogrammide massiiv. Kasutades NumPy funktsiooni *arange,* kus alguspunkt on 0 ja teine parameeter (lõpppunkt) *DataDTO* objektis olev *nr_ifgs*. Lisaks filteeritakse *drop_ifg_index* alusel, kui seal massiivis on midagi.

Järgnev tehakse siis kui privaatsed parameetrid *weed_standard_dev* ja *weed_max_noise* ei olnud suuremad või võrdsed kui pii. StaMPS'is oli see nimetatud kui *no_weed_noisy*. StaMPS'is oli eraldi loogika selle jaoks kui tingimus ei kehti, aga siin ei ole seda implementeeritud.

Seejärel eemaldatakse müra mille jaoks kasutatakse „psweed.2.edge“ failist saadud andmeid. Lisaks sellele kasutatakse massiivi *selectable_ps*, mille järgi selekteeritakse massiividest *ph, k_ps, c_ps* elemendid. Protsessist tagastatakse massiivid *edge_std*, *edge_max*, mida kasutatakse järgmises funktsioonis.

Järgmises funktsioonis, *get_ps_arrays* leitakse massiivid *ps_std, ps_max*. Funktsiooni sisendparameetriteks peale eelnevalt leitud kahel massiivile on ka massiivi *selectable_ps* tõeste väärtuste arv ja „psweed.2.edge“ loetud massiiv indeksitest. edge_std ja edge_max väärtustest leitakse minimaalsed. Valimiseks kasutatakse „psweed.2.edge“ indekseid. selectable_ps massiivi tõeste väärtuste arvu kasutatakse, massiivide *ps_std, ps_max* loomiseks. See saab nende massiivide pikkuseks.

Nende abil leitakse, mis alad on mürarikkad funktsioonis *estimate_max_noise.* Selleks kõigepealt leitakse massiiv *weeded*, mis pärast salvestatakse klassimuutujatesse nimega *selectable_ps2*. Selle täitmiseks leitakse tõeväärtused valemile *ps_std < self.__weed_standard_dev & ps_max < self.__weed_max_noise*. Selle jaoks kasutatakse NumPy funktsiooni *logical_and,* mis tagastab sama moodi tõeväärtustega täidetud massiivi nagu *selectable_ps* juba on. Funktsioonist tagastatakse eelnevalt mainitud *selectable_ps2* ja *selectable_ps.* Viimase massiivi tõesed väärtused on asendatud massiivi *weeded* väärtustega.

PsWeed klassis on funktsioon, et järgmiste protsesside jaoks selekteerida massiividest elemendid. Neil elementidel on tehtud indeksite (kas siis *keep_ind* või *coh_thresh_ind*) järgi selekteerimine *load_ps_params* funktsioonis ja seejärel *selectable_ps* massiivi järgi. Funktsioonist tagastatakse *coh_ps, k_ps, c_ps, ph_patch, ph, xy, pscands_ij, lonlat, hgt, bperp*. StaMPS'is selle asemel salvestati muutujad uudesse .mat failidesse, mille lõpus olevat numbrit suurendati ühe võrra. Näiteks massiiv *ph* salvestati *ph* „ph2.mat“ faili, enne oli failinimeks „ph1.mat“.

3.6.6 Klass PhaseCorrection

Tegemist on selles töös teostatava viimase sammuga. Selle käigus parandatakse vaatenurga tõttu tekkinud faasiviga. Tehtud on see StaMPS'i faili „ps_correct_phase.m" põhjal.

Siin väga palju loogikat ei olnud, aga mainida tasub, et kas siin oli väike small_baseline_flag'i haru. Siin on implementeeritud vaid osa, kus see muutuja on väärtusega „n".

Funktsiooni initsialiseerimiseks on vaja objekte *PsFiles* ja *PsWeed*.

Esmalt leitakse muutujad *PsFiles* ja *PsWeed* objektidest. *PsFiles* klassist leitakse ainult ülempildi indeks ja interferogrammide arv. *PsWeed* protsessist andmete saamiseks kasutatakse funktsiooni *get_filtered_results.* Sellest funktsioonist kõiki tagastusparameetreid vaja ei ole ja neid me ignoreerime. Vaja läheb *k_ps, c_ps, ph_patch, ph* ja *bperp*, mis kõik on massiivid. Kuna kõik massiivid on juba õigesti selekteeritud ja filtreeritud siis siin funktsioonis rohkem ei tehta midagi.

Leitud muutujad pannakse sisemisse klassi *DataDTO*.

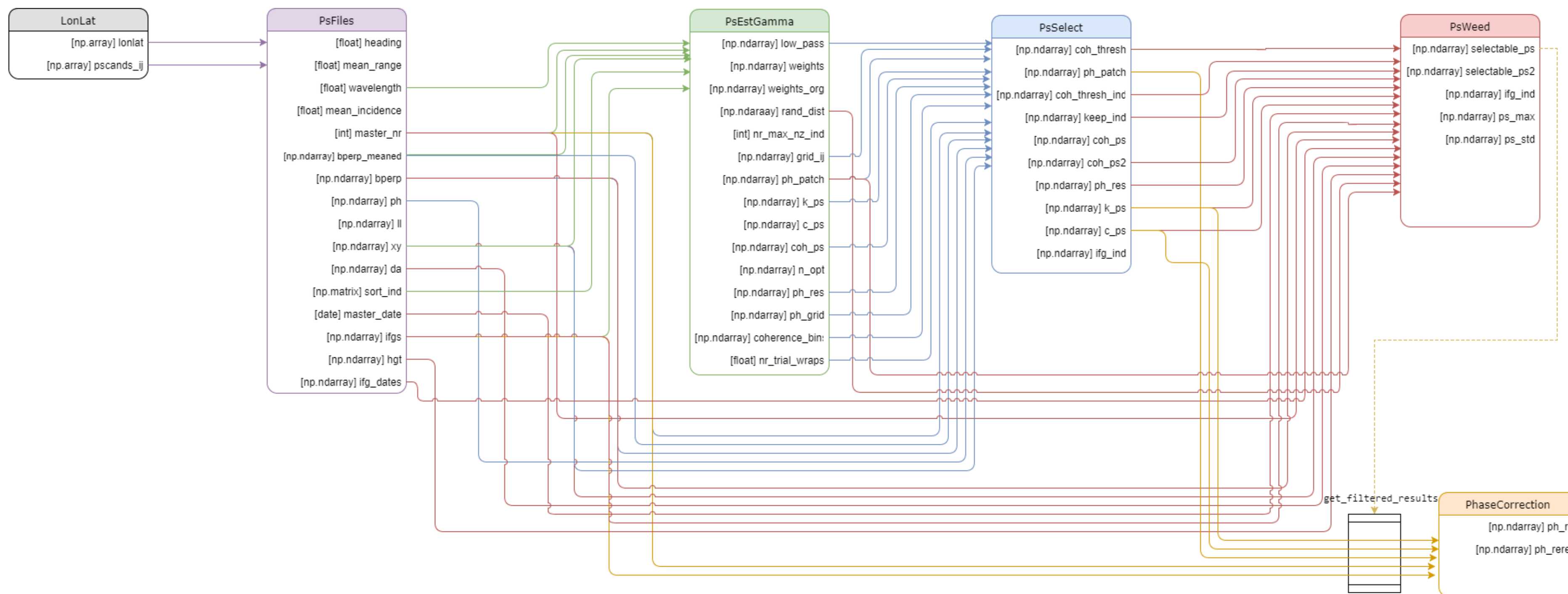
Esmalt leitakse massiiv *ph_rc*. Selle leidmiseks lisatakse *bperp* massiivi ülempildi indeksi veerg, mis täidetakse nullidega. Seda ja massiive *k_ps, c_ps* ja *ph* kasutatakse selleks, et leida muutuja *ph_rc*.

Viimasena leitakse massiiv *ph_reref*, mis leitakse funktsiooniga *get_ph_reref*. See on *ph_patch*, kuhu on lisatud masteri veerg, sarnaselt *bperp* massiiviga. Erinevuseks on, et *bperp*'is uus veerg täideti nullidega, siin ühtedega.

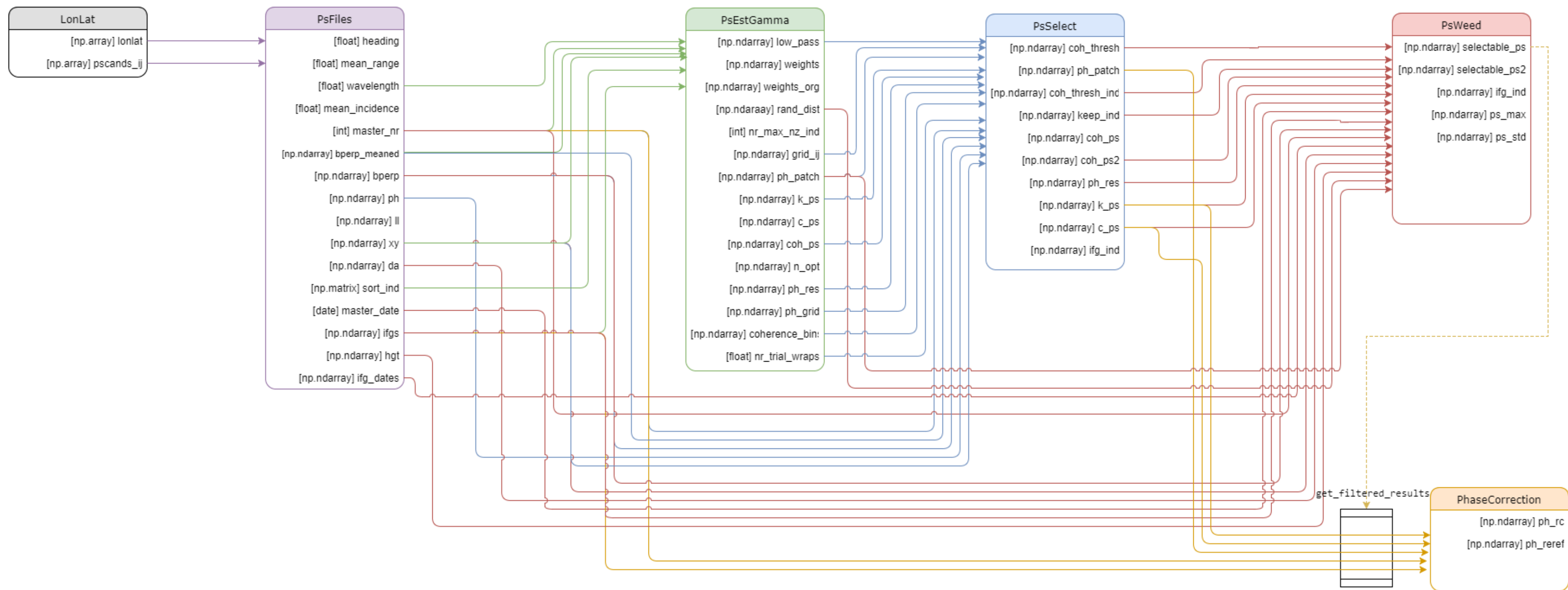
Need kaks massiivi, *ph_reref* ja *ph_rc*, lisatakse klassimuutujatesse.

Tähele tuleb panna, et StaMPS'is oli üks alamsamm, mida siin koodis pole tehtud, aga kuulus viienda sammu juurde. Seda ei tehtud failis fails „ps_correct_phase.m", vaid kutsuti välja eraldi, kui StaMPS'i töö lõppes viienda sammuga. Tegemist on sammuga kus leitakse standardhälbed interferogrammidele. Loogika asub failis „ps_calc_ifg_std.m".

3.7 Andmeskeem



Joonis 4 näitab, kus klassides erinevad muutujad tehakse ja kus neid kasutatakse.



Joonis 4. Andmemudel.

3.8 Pythonis ja Matlabis ekvivalentse koodi loomise probleemidest interpoleerimise funktsiooni näitel

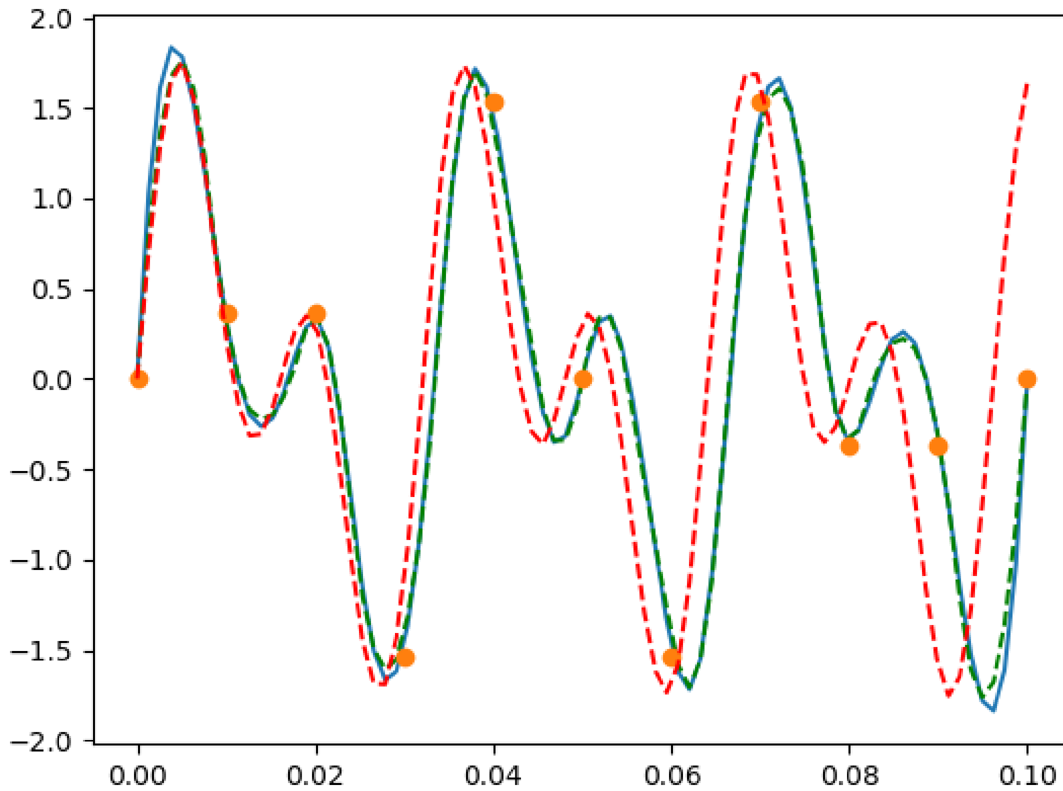
MATLABis kasutatakse madalpääs-interpoleerimise algoritmi [36] SciPy kasutab aga pisut teisiti implementeeritud interpoleerimise algoritmi. Proovides erinevaid lahendusi leidsime sellise lahenduse nagu on tehtud klassi MatlabUtils funktsioon interp.

Funktsioon mida näites interpoleerisime oli järgnev:

$$y = \sin(2\pi * 30t) + \sin(2\pi * 60t)$$

Kus t on arvujada 0'st 1'ni mille kasvamise vahemikku muutsime vastavalt kui palju punte me vajame. Lähtekood millega joonis on tehtud on lisades (Lisa 1).

Jooniselt 5 on näha kuidas erinevad Matlab'is ja Python'is realiseeritud interpoleerimise tulemused. Python'is kasutasime interpoleerimisel teist järku polünomiaalset interpoleerimist ((funktsioonis parameeter *kind*) on „quadratic" (joonisel roheline kriipsjoon)). Funktsiooni vaikeväärtus liigile „qubic" andis küll väga sarnase tulemuse, aga natuke vähem laugemate nurkadega, kui eelmine (joonisel roheline punktjoon). SciPy *iterp1d* funktsioon andis meile eelmistega üpris sarnase tulemuse, aga ka selle „quadratic" on natuke liiga terav võrreldes MatlabUtils klassi tehtuga (sinine pidevjoon). Töös on näidatud ka keele MATLAB tulemus sellele funktsiooni interpoleerimisele (punane punktjoon). Siin on näha, et see on algsega võrreldes (joonisel kollased punktid) lõpus üpris ebatäpne. Kattuvust on väga vähe. Seega on töös leitud funktsioon isegi täpsem kui algne programmis StaMPS kasutatu.



Joonis 5. Interpoleerimise tulemused. Roheline kriipsjoon – algne funktsioon. Punane kriipsjoon – Matlab'i *interp* funktsiooni tulemus. Sinine pidevjoon - Python'is realiseeritud teist järku polünomiaalse interpoleerimise funktsiooni tulemus.

Sellised erinevused matemaatiliste funktsioonide realiseerimisel illustreerivad antud töö käigus kogunenud ohtralt ajakulu, et leida adekvaatset viisi, kuidas Python'is kirjutatud koodi testida. Käesoleva näite lähtekood on Lisas A.

Töö käigus esines ka mitmeid teisi raskesti avastatavaid probleeme. Näiteks muutus töö tegemise käigus Python 3 versiooniuuendusega Python'is kasutatav vaikimisi täisarvuvorming 32 bitiselt täisarvult 64 bitisele. See viis vearaporti esitamiseni SNAP'i foorumi[1].

[1] http://forum.step.esa.int/t/correction-needed-in-snappy-ndvi-py-example-file/6660

3.9 Jõudlustestid tehtud tööle

Järgnevates testitakse ja võrreldakse tehtud töö jõudlust StaMPS'i omaga.

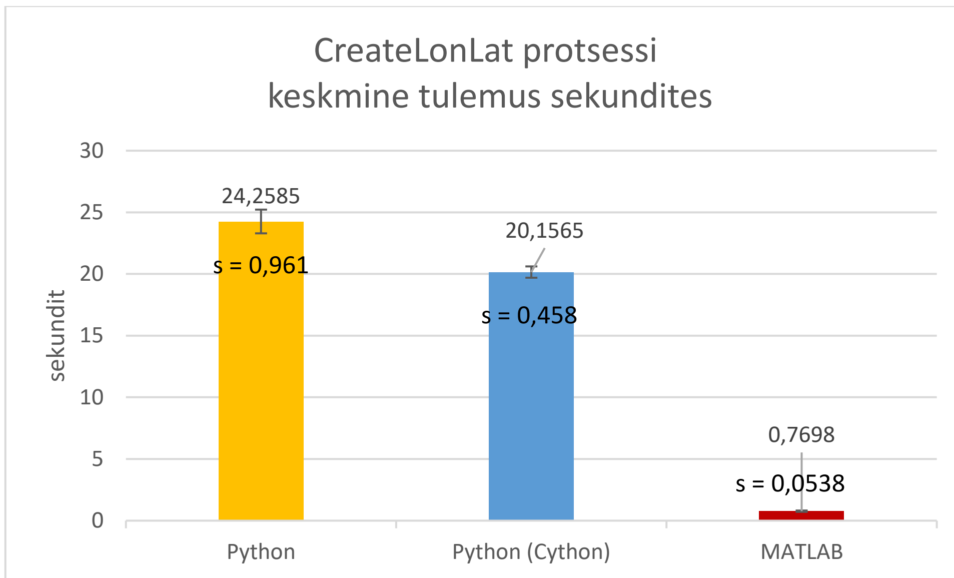
Testid on läbi viidud kasutades StampsReplacer'i testiklasse. Kulunud aja mõõtmiseks kasutati logikirjeid. Võeti alguse (logis „Start“) ja lõpu (logis „End“) vahe. Nii saame ainult töötlusele kulunud aja ja ei võta arvesse kui kaua test ise kokku töötas.

Python tulemused on leitud kahel viisil. Tavaline Python ja Python'i kood kompileeritud kasutades Cython'i. Mõnes protsessis on ka tulemuste vahesalvestamise funktsionaalsus. Neid on testitud vahesalvestatud tulemi laadimisega kui ka ilma.

MATLAB tulemused on saadud kasutades selles keeles olevat kiiruse mõõtmise käske *tic*/ *toc* [37]. Kus käsk *tic* tähistab algust ja *toc* lõppu. Lõpus tagastatakse tulemus. Mõõtmise jaoks lisas autor StaMPS'i koodifailidesse need käsud ning mõõtis kõiki protsesse eraldi.

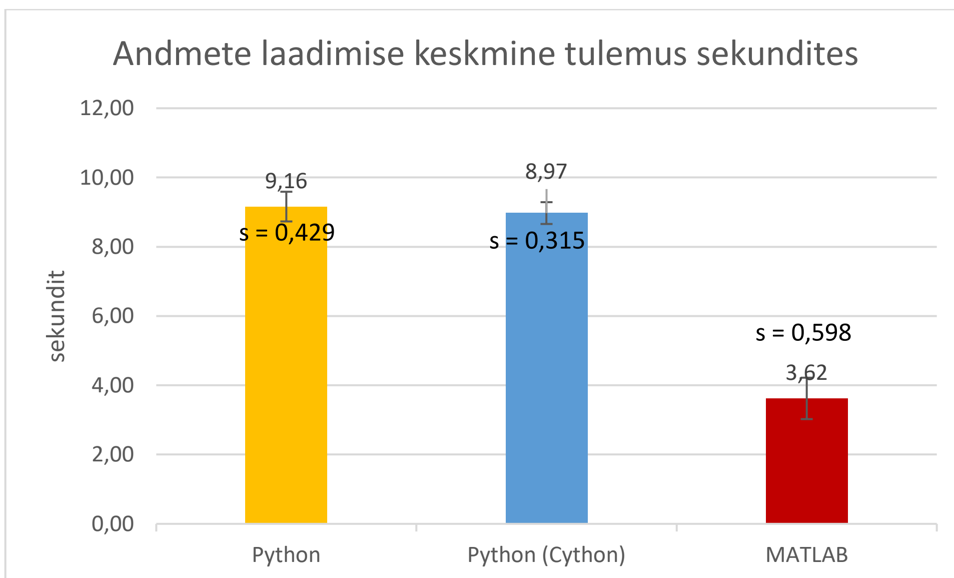
Kõiki viite protsessi on käivitatud kümme korda iga eelmainitud viisiga. Joonistel on kümne korra keskmine tulemus ja tulemuste standardhälve (joonistes tähistatud sümboliga s).

Esimene on *CreateLonLat*. Python's kasutati selle jaoks Snappy teeki, mis omakorda kasutab programmi SNAP Java koodi. Teistes protsessides seda ei kasutata. StaMPS'is leiti see natuke teisemini.



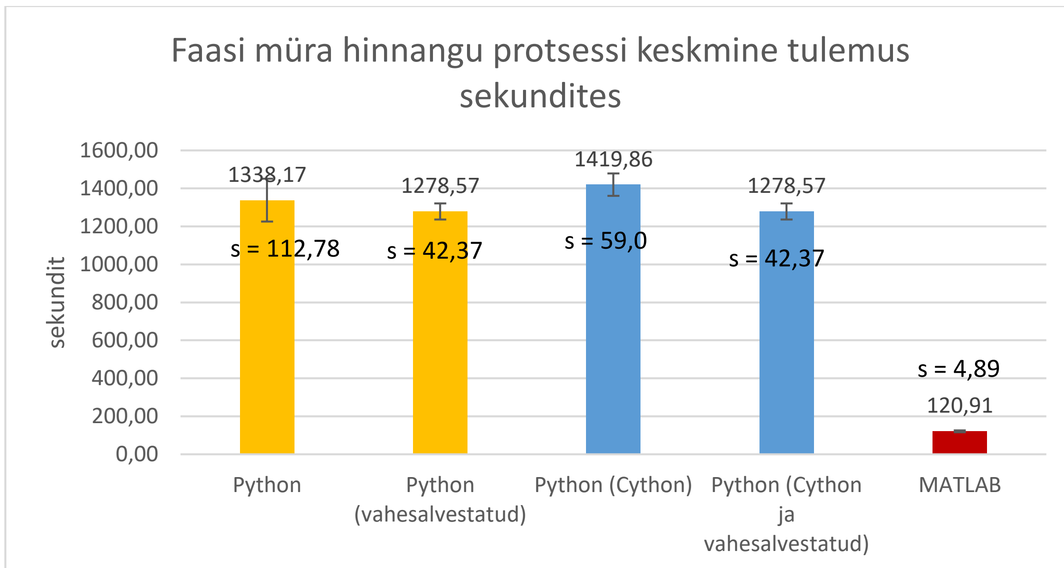
Joonis 6. *CreateLonLat* protsessi keskmine tulemus.

Esimene StaMPS'i samm, on andmete laadimine.



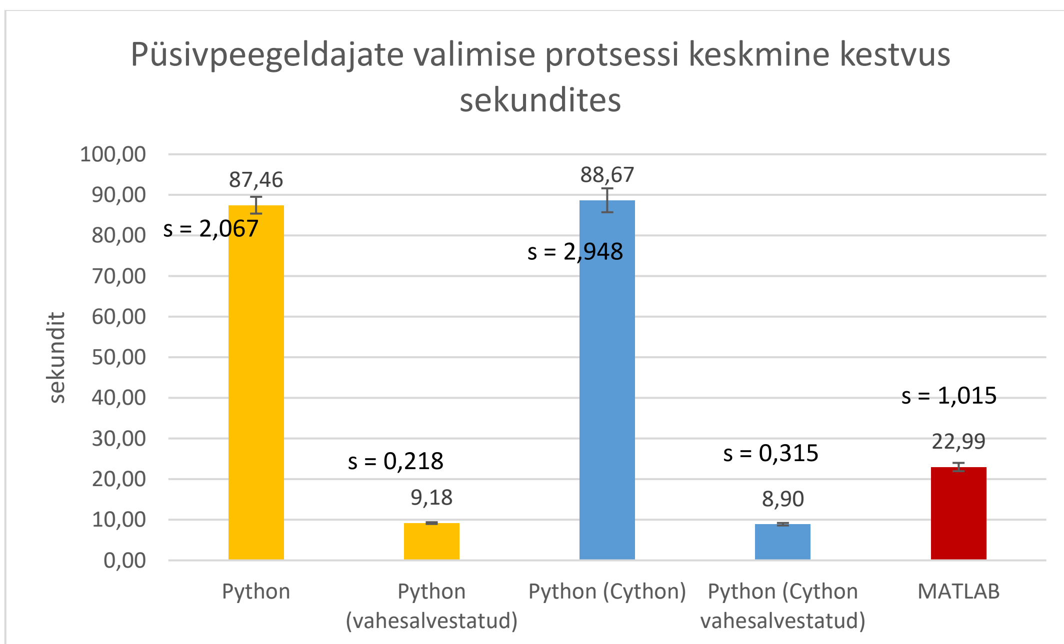
Joonis 7. Andmete laadimise keskmine tulemus.

Teisena tehakse faasi müra hindamine. Siin on kaks Python'i tulemust, üks kus juhuslike arvude massiiv arvutatakse välja (joonisel Python) ja teine kus kasutatakse varem leitud tulemust (Joonisel Python (vahesalvestatud) ja Python (Cython vahesalvestatud)).



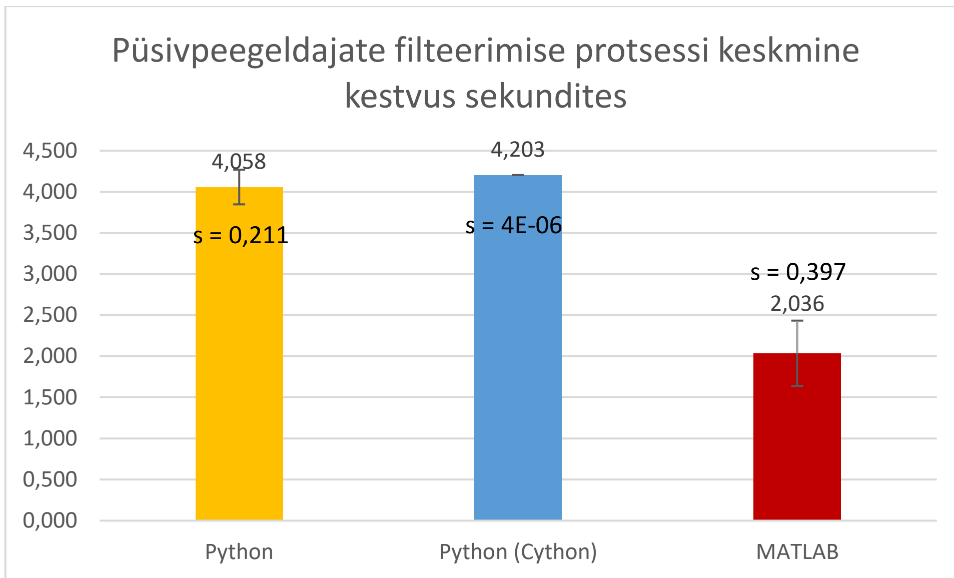
Joonis 8. Faasi müra hindamise protsessi keskmine tulemus.

Kolmandaks leitakse püsivpeegeldajad. Nagu eelmises, on ka siin kaks Python'i tulemust, esimene kus arvutatakse *ph_patch* massiiv välja ja teine kus see loetakse vahesalvestatud failist.



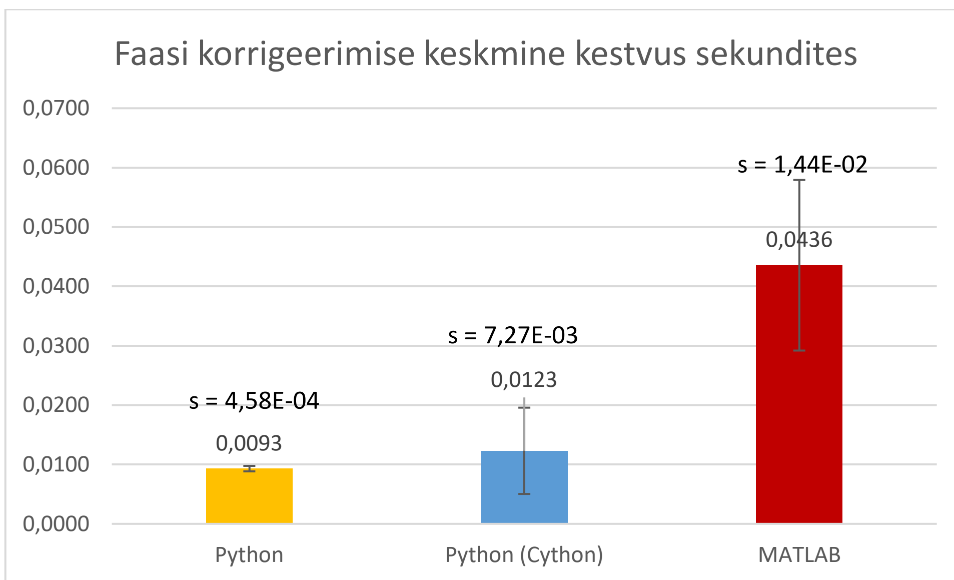
Joonis 9. Püsivpeegeldajate valimise protsessi keskmine tulemus.

Neljanda sammuna filtreeritakse püsivpeegeldajad. Siin vahepealseid tulemusi ei salvestata.



Joonis 10. Püsivpeegeldajate filtreerimise keskmine tulemus.

Viimane, viies samm, protsessis on faasi korrigeerimine.



Joonis 11. Faasi korrigeerimise keskmine kestvus sekundites.

Tulemustest on näha, et enamasti on Python ikkagi oluliselt aeglasem kui MATLAB. Küll aga kui protsessi käivitada korduvalt siis peaks faasi müra hindamine ja püsivpeegeldajate valik töötama kiiremini kui esimene kord, tänu mõningate tulemuste vahesalvestusele.

Samas on ka näha, et lihtsamad protsessid on väga lähedal MATLAB’i tulemustele. Näiteks püsivpeegeldajate valik on vaid sekund aeglasem MATLAB’i omast ja faasi korrigeerimine on isegi kiirem MATLAB’ist. Samas viimase puhul tasub tähele ka panna, et protsess võtab mõlemas keeles aega alla sekundi ning arvatavasti pole vahe siiski märgatav.

Märgatav kiiruse vahe on faasi müra hindamise juures, mis on peaaegu 13 korda aeglasem (StaMPS’is olevast).

4 Edasiarenduse võimalused

Loodud rakenduses on küll tehtud rakenduse StaMPS esimesed viis sammu, aga sellega pole arendus veel lõppenud. Tegemist on algusega, et püsivpeegeldajate leidmist teha efektiivsemaks. Lisaks on kood ümberkirjutatud nii, et edasiarendamine oleks lihtne. Testid, mis töö käigus tehti, tagavad selle, et edasiarendused ei rikuks juba olemasolevat funktsionaalsust. Kuna ümberkirjutatud lahendus kasutab Python'i siis pole ka enam litsentsi vaja.

Järgnevalt sõnastab autor mõningad edasiarendused, mida arenduse käigus märkas. Esmalt peaks üle vaatama peatüki kolm alampeatükkides kirjeldatud osad, mis on tegemata. Näiteks *PsEstGamma* protsessis on tehtud kaalude leidmine vaid ühe algoritmiga (sisemine parameeter *PsEstGamma*). Teha tuleks ka teiste väärtustega kaalude leidmine. Lisaks sellele oli veel teisi protsesse, milles oli funktsionaalsust, mis *StaMPS*'is oli aga selles programmis ei ole. On ka võimalik, et mõnda neist funktsionaalsustest polegi enam tarvis, kasutati näiteks teiste satelliitide andmete töötluses.

Teisena peaks paljud sisemised parameetrid panema sättefailist loetavaks. Umbes sarnaselt nagu on juba implementeeritud näiteks, mis kaustast lugeda töödeldavaid faile. Põhjusena võib tuua näiteks *PsEstGamma* protsessis on sisemised parameetrid, mis hetkel on need tehtud nii sinna kui ka *PsSelect* klassi.

Kolmandaks peaks parandama arendatud rakenduse jõudlust. Antud töös küll kirjutati paljud kohad selliselt ümber, et kui võimalik siis mõningaid arvutusi tehakse alguses korra, mitte mitu korda tsüklis. Küll aga on Python ikkagi aeglasem kui MATLAB. Seetõttu vajab magistritöös kirjutatud rakendus Python keele spetsiifilist optimeerimist. Arvestades üldist töömahtu, ei mahtunud Pythoni'le optimeerimine käesoleva töö skoopi.

Näiteks, üks koht mida peaks optimeerima, on protsessis nimega *CreateLonLat*. Funktsioon *read_pixel* on üpris aeglane (kirjeldatud peatükis 3.6.1). Samuti on *PsEstGamma* protsessis aeglane *PsTopofit* funktsioon.

Peale selle tehakse veel paljudes kohtades *for*-tsüklites tegevusi, mis Python keeles ei ole väga kiired. Seal peaks arvatavasti kasutama *map* funktsiooni või siis listi genereerimist (inglise keeles *list comprehensions*). Mõlemad on oluliselt kiiremad, kui tavalised *for*-tsüklid. [38]

Lisaks neile võiks uurida ka Python'i parallelismi ja CUDA kiirendi implementeerimist. Viimase uurimine takerdus selle taha, et teek, Copperhead, mis oli lihtsasti kasutatav oli aegunud ning selle arendus oli kirjutamise hetkeks peatunud. [39]

5 Kokkuvõte

Käesolevas magistritöös kirjeldas autor PSInSAR protsessis kasutatava rakenduse StaMPS ümberkirjutamist, kasutades vabavaralisi teeke. Loodud lahendus aitab protsessi kasutada omamata keele MATLAB litsentsi. Lisaks on kood ümberkirjutatud sedasi, et seda saaks hõlpsasti edasi arendada.

Rakendus on kirjutatud keeles Python ja põhiliseks teegiks mida kasutatakse on NumPy. Selline valik on tehtud seepärast, et sellele teegile on lai MATLAB funktsioonide tugi, funktsioonid on kiired ning on saadaval abistavad materjalid, kuidas MATLAB'i koodi ümber kirjutada kasutades antud teeki.

Töös on tehtud viis esimest sammu mis programmi StaMPS töötluses on. Nendeks on: sisendandmete lugemine, faasi müra hinnang, püsivpeegeldajate valik, püsivpeegeldajate filtreerimine ning faasikorrektsioon. Kuna töö eesmärgiks oli säilitada StaMPSi funktsionaalsus, siis püüti igal sammul säilitada võrreldavus vastavate StaMPSi sammudega. Jõudlustestid näitasid, et selliselt realiseeritud algoritmid on Pythonis mõnevõrra aeglasemad kui MATLABis. Samuti ei andnud lihtsad paralleliseerimise lähenemised kohest arvestatavat võitu. Koodi jõudluse parandamine vajab koodi refaktoreerimist ja optimeerimist arvestades Python'i ja Numpy teegi iseärasusi.

Magistritöö autor loodab, et tehtud töö leiab kasutust, et leida püsivpeegeldajaid Sentinel-1 satelliidi tehtud piltidelt. Lisaks loodab, et rakendust arendatakse edasi teiste inimeste poolt.

Kasutatud kirjandus

[1] K. Zalite ja K. Voormansik, „Differential and Persistent Scatterer SAR Interferometry,“ Tartu Observatoorium, 2016.

[2] „SNAP,“ [Võrgumaterjal]. Available: http://step.esa.int/main/toolboxes/snap/. [Kasutatud 19 märts 2017].

[3] „StaMPS,“ [Võrgumaterjal]. Available: https://homepages.see.leeds.ac.uk/~earahoo/stamps/. [Kasutatud 19 märts 2017].

[4] A. Ferretti, A. Monti-Guarnieri, C. Prati ja F. Rocca, „InSAR Principles: Guidelines for SAR Interferometry Processing and Interpretation,“ ESA Publications, 2007.

[5] A. Hooper, D. Bekaert, K. Spaans ja M. Arıkan, „Recent advances in SAR interferometry time series analysis for measuring crustal deformation,“ Delft University of Technology, 2011.

[6] A. J. Hooper, „Persistent Scatterer Radar Interferometry For Crustal Deformation Studies And Modeling Of Volcanic Deformation,“ 2006.

[7] „Sentinel 1 tooblox Git - BackGeocodingOp Help,“ [Võrgumaterjal]. Available: https://github.com/senbox-org/s1tbx/blob/c0a5b2c4dfd0d11619672ccfd14d38751aa67649/s1tbx-op-sentinel1-ui/src/main/resources/org/esa/s1tbx/sentinel1/docs/operators/BackGeocodingOp.html. [Kasutatud 22 märts 2017].

[8] „Copernicus,“ ESA, [Võrgumaterjal]. Available: http://www.esa.int/Our_Activities/Observing_the_Earth/Copernicus/Overview4. [Kasutatud 19 märts 2017].

[9] „Introducing Sentinel-1,“ ESA, [Võrgumaterjal]. Available: http://www.esa.int/Our_Activities/Observing_the_Earth/Copernicus/Sentinel-1/Introducing_Sentinel-1. [Kasutatud 19 märts 2017].

[10] L. Reisberg, „Metsa raiealade tuvastamine Sentinel-1 SAR andmete põhjal,“ 2015.

[11] „Introducing Sentinel-2," ESA, [Võrgumaterjal]. Available: http://www.esa.int/Our_Activities/Observing_the_Earth/Copernicus/Sentinel-2/Introducing_Sentinel-2. [Kasutatud 19 märts 2017].

[12] „Introducing Sentinel-3," [Võrgumaterjal]. Available: http://m.esa.int/Our_Activities/Observing_the_Earth/Copernicus/Sentinel-3/Introducing_Sentinel-3. [Kasutatud 19 märts 2017].

[13] „Introducing Sentinels-4, 5, 5P," ESA, [Võrgumaterjal]. Available: http://www.esa.int/Our_Activities/Observing_the_Earth/Copernicus/Sentinels_-4_-5_and_-5P. [Kasutatud 19 märts 2017].

[14] „Introducing Sentinel-6," [Võrgumaterjal]. Available: http://www.esa.int/Our_Activities/Observing_the_Earth/Copernicus/Sentinel-6. [Kasutatud 19 märts 2017].

[15] „Sentinel-1 Toolbox," [Võrgumaterjal]. Available: http://step.esa.int/main/toolboxes/sentinel-1-toolbox/. [Kasutatud 19 märts 2017].

[16] A. Hooper, B. David ja S. Karsten, „StaMPS/MTI Manual," Universityof Leeds, Leeds, 2013.

[17] „Doris," [Võrgumaterjal]. Available: http://doris.tudelft.nl/. [Kasutatud 21 märts 2017].

[18] „Delft Object-oriented Radar Interferometric Software. Version v4.02," [Võrgumaterjal]. Available: http://doris.tudelft.nl/software/doris_v4.02.pdf. [Kasutatud 5 1 2018].

[19] „ROI_PAC," [Võrgumaterjal]. Available: http://roipac.org/cgi-bin/moin.cgi/ROI_PAC. [Kasutatud 21 märts 2017].

[20] „ROI_PAC Sentinel-1," [Võrgumaterjal]. Available: https://github.com/RaphaelGrandin/ROI_PAC-Sentinel1. [Kasutatud 21 märts 2017].

[21] „ISCE," [Võrgumaterjal]. Available: http://winsar.unavco.org/isce.html. [Kasutatud 21 märts 2017].

[22] „GAMMA SAR Software," [Võrgumaterjal]. Available: https://www.gamma-rs.ch/no_cache/software/system-overview.html. [Kasutatud 21 märts 2017].

[23] „GIAnT User manual," [Võrgumaterjal]. Available: http://earthdef.caltech.edu/attachments/download/15/GIAnT_doc.pdf. [Kasutatud 21 märts 2017].

[24] Z. Yunjun ja H. Fattahi, „PySAR Documentation 1.2," [Võrgumaterjal]. Available: https://github.com/hfattahi/PySAR/raw/master/docs/PySAR-1.2.pdf. [Kasutatud 21 märts 2017].

[25] B. Peterson ja Python, „Python 2.7 Release Schedule," [Võrgumaterjal]. Available: http://legacy.python.org/dev/peps/pep-0373/. [Kasutatud 9 1 2018].

[26] „Sarapoz manual," [Võrgumaterjal]. Available: https://www.sarproz.com/software-manual/. [Kasutatud 21 märts 2017].

[27] „NumPy for Matlab users," The Scipy community, [Võrgumaterjal]. Available: https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html. [Kasutatud 17 12 2017].

[28] „NumPy for MATLAB users," [Võrgumaterjal]. Available: http://mathesaurus.sourceforge.net/matlab-numpy.html. [Kasutatud 17 12 2017].

[29] „Intel® Math Kernel Library," Intel® Software, [Võrgumaterjal]. Available: https://software.intel.com/en-us/mkl. [Kasutatud 1 9 2018].

[30] Numpy, „Data type objects," Numpy, [Võrgumaterjal]. Available: https://docs.scipy.org/doc/numpy-1.13.0/reference/arrays.dtypes.html#arrays-dtypes. [Kasutatud 12 25 2017].

[31] Scipy, „numpy.mean — NumPy v1.13 Manual," [Võrgumaterjal]. Available: https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.mean.html. [Kasutatud 9 1 2018].

[32] Scipy, „numpy.histogram — NumPy v1.13 Manual," [Võrgumaterjal]. Available: https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.histogram.html. [Kasutatud 9 1 2018].

[33] n. —. N. v. Manual. [Võrgumaterjal]. Available: https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.squeeze.html. [Kasutatud 9 1 2018].

[34] „Automatic parallelization with @jit,“ [Võrgumaterjal]. Available: http://numba.pydata.org/numba-doc/dev/user/parallel.html. [Kasutatud 9 1 2018].

[35] Scipy, „numpy.zeros — NumPy v1.13 Manual,“ [Võrgumaterjal]. Available: https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.zeros.html. [Kasutatud 9 1 2018].

[36] MathWorks, „MATLAB interp,“ [Võrgumaterjal]. Available: https://se.mathworks.com/help/signal/ref/interp.html. [Kasutatud 9 1 2018].

[37] MathWorks, „MATLAB tic,“ [Võrgumaterjal]. Available: https://se.mathworks.com/help/matlab/ref/tic.html?s_tid=doc_ta. [Kasutatud 9 1 2018].

[38] „PythonSeed/PerformaceTips,“ [Võrgumaterjal]. Available: https://wiki.python.org/moin/PythonSpeed/PerformanceTips. [Kasutatud 5 1 2018].

[39] „Github - Copeprhead,“ [Võrgumaterjal]. Available: https://github.com/bryancatanzaro/copperhead. [Kasutatud 5 1 2018].

Lisa A: Interpoleerimiste vahelise võrdluse joonistamise kood

```
import matplotlib.pyplot as plt
import scipy
import numpy as np

from scripts.utils.MatlabUtils import MatlabUtils

t1 = np.linspace(0, .1, num=20, endpoint=True)
t2 = np.linspace(0, .1, num=11, endpoint=True)
t3 = np.linspace(0, .1, num=80, endpoint=True)

x1 = np.sin(2 * np.pi * 30 * t1) + np.sin(2 * np.pi * 60 * t1)
x2 = np.sin(2 * np.pi * 30 * t2) + np.sin(2 * np.pi * 60 * t2)
x3 = np.sin(2 * np.pi * 30 * t3) + np.sin(2 * np.pi * 60 * t3)

y_quadratic = scipy.interpolate.interp1d(t1, x1, kind='quadratic')
y_utils_interp = MatlabUtils.interp(x1, 4)
y_utils_interp_q = MatlabUtils.interp(x1, 4, 'quadratic')

matlab_actual =
np.array([0,0.696503640040176,1.27711714453952,1.64739005556627,1.7529398049
2121, 1.55336268776565,1.15378939450458,0.657775321121426,0.180049415982107,
-0.150441713008426,-0.313908157437896,-0.304898520098431,-0.160104878924336,
0.0644346239092101,0.263336285748697,0.349024373653394,0.260860582334277,
-0.0578533831714737,-0.513488813788482,-1.01306503534041,-1.44534765897933,
-1.68611595824710,-1.68939845829740,-1.43196789089640,-0.938912181896321,
-0.255801888035542,0.475339458228588,1.12611939152925,1.58361297863231,
1.73487568852383,1.61969025411147,1.28930328691517,0.831989902727521,
0.358816703073747,-0.0396999042430857,-0.290955267527316,-0.361219085226352,
-0.229054685927110,-4.84661257572023e-
16,0.229054685927109,0.361219085226352,
```

```
0.290955267527316,0.0396999042430862,-0.358816703073745,-
0.831989902727519,
                          -1.28930328691517,-1.61969025411147,-
1.73487568852382,-1.58361297863231,
                          -1.12611939152925,-
0.475339458228588,0.255801888035540,0.938912181896319,
1.43196789089640,1.68939845829740,1.68611595824710,1.44534765897933,

1.01306503534041,0.513488813788486,0.0578533831714773,-0.260860582334274,
                          -0.349024373653392,-0.263336285748696,-
0.0644346239092107,0.160104878924335,

0.304898520098429,0.313908157437894,0.150441713008424,-0.180049415982108,
                          -0.657775321121427,-1.15378939450458,-
1.55336268776565,-1.75293980492121,
                          -1.64739005556627,-1.27711714453952,-
0.696503640040178,-2.20436423846224e-15,

0.696503640040173,1.27711714453952,1.64739005556626])

plt.plot(t3, y_utils_interp, '-', t3, y_utils_interp_q, 'g--', t2, x2,
'o', t3, matlab_actual, 'r--')

plt.savefig("interp")
```