PS-INSAR PROTSESSI REALISEERIMINE
KASUTADES VABAVALALISE TEEKE

Magistritöö

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

Kinnitan, et olen koostanud antud lõputöö iseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärit 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 käiteseadavad. Arvatavasti StaMPSile nende andmete ametlikku tuge ei kirjutata, sest projekti edasiarendus lõppenud.

Töö tulemusena kirjutati programmi StaMPS PSInSAR protsessi 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.
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 funktsooni 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] analüüksiks kasutatakse täna 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 töötlmiseks 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 programmeerimiskeel 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


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

Käesolevas lõputöö lõpeaks 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.


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

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 vörreldakse faasimuutu. Mida rohkem on pilte seda täpsem on tulemus. Kahe või enama faasipildi/nihe võrdlemist nimetatakse interfeeromeetriaks [4].

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


2. Tavaliselt moodustad 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


Koherentsust saab välja arvutada valemiga (1):

\[
\gamma = \frac{s_1 s_2^*}{\sqrt{(s_1 s_2^*)^2}} \quad 0 \leq |\gamma| \leq 1
\]

Kus \(s_1\) ja \(s_2\) on, kaks kompleksarvu interferomeetria pildis, tärn tähistab kaaskompleks arvu ja sulud (\(\cdot\)) 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-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 närsked. Andmeid kasutatakse ilmaennustuseks, mere veetaseme monitoorimiseks, merereostustuste avastamiseks ja mereelustiku vaatlemiseks.

ja Sentinel-5 UVNS (Ultraviolet Visible Near-infrared Shortwave) sensoriga. Sentinel-5 on olema 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 laineõiguseid. See aitab ennastada 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 ja moodulid on vabalt allalaetavad ning paigaldatavad arvutisse. Küll aga tuleb mainida, et rakendus tahab väga palju arvutusressurssit, 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 väljalöikamine (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 andmetest 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.
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*).
Viimase sammu tulemusena peaks saama pikslite deformatsiooni, mida saab juba edasi töödelda SNAPis.

StaMPS'i kasutatakse selleks, et leida huvipakkuvu ala deformatsioon. Paraku ei ole praeguses protsessis StaMPS väga stabiilne. Kui ala on liiga väike siis püsivpeegeldajaid ei leita. Ala suurus peab olema vähemalt 3km². 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 veateateeni jõuda, peab mitukümmend minutit StaMPS töötlema andmeid. Peale viga saab küll jätkata seal tammust kust töötlus poolel jäi, aga inimene siiski peab töötluse juures olema ning vajadusel sekkuma.


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

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 tarkvara 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


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 andmete moodul, interferomeetri/ InSAR protsessi moodul, DInSAR ja geokodeerimise moodul, punktide ja püsivpeegeldajate analüüs (*Interferometric Point Target*) moodul, statistika, klassifitseerimine, filtreerimise ja eksportimise moodul ja maapinna geokodeerimise moodul eraldi neile, kes DInSAR protsessi ei kasuta.
Lisaks reklamitakse Gammat kui kasutajasõbralikku vahendit, mida aga StaMPS näiteks ei ole. Tarkvaral on Sentinel-1 töötlemise tugi.

Tegemist on tasulise tarkvaraga ning seepärast ei sobi kässoles kasutamiseks, kuna siin otsitakse vabavaralist alternatiivi. Küll aga saaks selle abil vörrelda kui täpselt ja kui palju kiiremini või aeglasmalt 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


2.7.7 SARPOZ

SARPOZ [26] (The SAR PROcessing tool by periZ) on tasuline tarkvara mis peaks asendama StaMPS. Kirjutatud on see programmeerimiskeelees MATLAB ning lähekood avatud ei ole, aga antud keel olgu 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 kirjeldatse magistritöö raames loodud programmi ja selle jõudlust vörreldes StaMPS-ga. Kirjeldatse 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 abistavates teekideks olid SciPy ja NumPy (versioon 1.13.3).

NumPy on väga populaarne teek MATLABis arendatud rakendust realisatsioonikes Pythonis. Internetis leidub arvestatabal 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 käivitamiseks läheb vaja on failis „env.txt“.

Töös ei ole kasutatud eelmaintud rakendusi PySAR ja GIAnT. Põhjustena, 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


3.2.2 Sättefailid ja parameetrite selgitus

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

Parameetrid failis on järgnevat:

- **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. Kohtu kuju tulemused (.npz failid) salvestatakse

1 Mõnes operatsioonisüsteemis, näiteks Ubuntu, 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 teigi kasutamine


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 korreksioon.

Järgnevates alampeatükvides 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 lõpukus leitakse muutujad muutukud 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 failide „ph1.mat“, „bp1.mat“, „la1.mat“, „lm1.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 püsivpeegeldajatest 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“ massiivid 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:

### 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.


Abiklassid ja nendes paiknevad abifunktsioonid on kaustas „utils“. Tegemist on funktsioonidega, mis on tihealt 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 abifunksioonid massiividega töötlemiseks. Funktsioonid on statailised.


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

3.4.2 Testide kaust

„tests“ kaustas on kaks alamkaust: „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 polümurfismi 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 filtrreerimine (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:

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


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.

Eelnevad seletatud aitab mõista joonis 2.

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

3.6 StampsReplacer koodi kirjeldus

Järgnevates alampeatükides on kirjeldatud magistritöö käigus loodud programmi koodi, mis on ümberkirjutatud StaMPS. Sellest ka nimi StampsReplacer. Kirjeldatud, 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't on massiivid vaikimisi justkui veeruvektorid, NumPy's on nad 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 alampeatükikes eraldi välja toodud.

Järgnevalt kirjeldavad koodifailid asuvad kaustas scripts/processes.
3.6.1 Klass CreateLonLat

See protsess pole otsekselt programm StaMPS osa. Küll aga tehakse seal edastest 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 leib „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 konfiguratsiooniparameter või on see kuskil alamkaustas. Kui PATCH_1 on alamkaustas siis tuleb see lisada parameetrisse „patch_folder“’s. Vastasel juhul ei leita faili pscands.1.ij.

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

Antud juhul kasutatakse Snappy teeki selleks, et lugeda SNAP’i failidest pikslid. Selleks leitakse SNAP’i failist vertikaalsihi ja ristisihi lainepeikused (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“ sisese kõik read ning pannakse need muutujasse pscands_ji. Väärtusi ei töödelda mitte mingil kujul. Lõpptulemuseks on massiiv, mille veerus on kolm täisarvu (int32) ja mille pikkus sõltub ala suurusest, aga talviselt on see sedades 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 väärused.

Esimese parameetri puhul on failiteega, mis tekib konfiguratsiooni parameetrite path ja patch_folder liitmisel. Näiteks kui parameeter path on säititud „C:\testi_andmed\test_1“ (jutumärkdeta) 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 parameetrikis. 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 teistest 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 algsest faasi lugemise protsessist. Esmalt loetakse failist „rsc.txt” failitee millest lugeda parameetrid. 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.
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 funksioonist leitud väärtused heading ja center_range_slc salvestatakse klassi muutujatesse heading ja mean_range. Tegemist on ujukomaarvudega. StaMPS’is salvestati väärtus „heading” MATLAB keskkonna parameetri parameetritesse (kasutas funktsiooni setparm). Sellist võimalust Python keeles pole ja need salvestatakse klassimuutujatesse. Lisaks leia autor, et selline stil, 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 lainepiikiks. 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 inteferogrammi kuupäev (kuupäeva formaat on YYYYMMDD). Näiteks kataloogitee on „C:\testifailid\test_1\resources\export\diff0\20160614_20160509.diff” siin siin on ülepildi kuupäevaks 14.06.2016 ja inteferogrammi kuupäevaks on 09.05.2016. Leitud järjendit, inteferogrammide failiteedest, kasutatakse pärast, et leida mitu inteferogrammi 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 inteferogrammi on enne ülepildi, peab leidma selle pildi kuupäeva. Selle leiaab 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 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.
jaotada eraldajate järgi (funktsioon split) ning anda õiges järjekorras ette date objektile. Öige parameetrite järjekord on aasta, kuu, päev.

Nüüd kus ülepildi kuupäev on teada, saame leida mitu pilti on enne ülepilti, mille kaudu leiame mitmes on ülepilt interferogrammide kogumiku. 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 teistpäde, mitu pilti on peale ülepilti. Aga selle asemel, et teha eraldi funktsioon selle jaoks antakse funktsioonile ette funktsioon kuidas võrrelda.


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 keelest 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 B₁ paralleelset baasliini. Θ on nurk satelliidi ja vaadeldava objekti vahel (ehk vaatenurk, inglise keelest 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):

$$bc / bn = tcn + baseline_rate \times \frac{(ij_{lon}-mean_{azimuth_line})}{prf}$$ (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'ist (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 saad ki 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 valemist (3) mida kutsutakse välja for-tsüklis.

$$bperp = bc \times \cos(sat_{look\_angle}) - bn \times \sin(sat_{look\_angle})$$ (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.
Seejärele 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).

\[
\text{incidence} = \arccos \frac{\text{sar_to_earth_center}^2 - \text{earth_radius_below_sensor}^2 - rg^2}{2 \times \text{earth_radius_below_sensor} \times rg}
\] (4)

Tähele tuleb panna, et antud koodis leitakse muutujate \( \text{sar_to_earth_center} \) ja \( \text{earth_radius_below_sensor} \) astme väärtused enne kui langemisnurk.

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

Seejärele leitakse muutu ja \( \phi \). 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 tavaliselt NumPy 128bit kompleksarvu sedasi laadida ei saa.

Seejärele leitakse \( ll \), kasutades funktsiooni „get_ll_array“. See leitakse massivist \( \text{lonlat} \), kus \( \text{lonlat} \) massiivi rea maksimaalne ja minimaalne liidetakse omavahel ja jagatakse kahega.

Siis leitakse \( xy \) ja sorteerimise järjekord (massiivi nimetusega \( \text{sort_ind} \)). See tehakse funktsioonis get_xy. \( xy \) massiivi jaoks võetakse nullis ja teine veerg massiivist \( \text{pscands_ij} \). 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 kasutatakse siis saaks selle leida läbi indeksi. See tähendab, et antud implementatsioonis massiiv \( xy \) on suurusega 2 * \( \text{pscands_ij} \) pikkus.

Kui \( \text{pscands_ij} \) 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.

Seejärele 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 järgekorraks 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 järgekorda funktsioonis sort_results, et sorteerida leitud massiivid mis on salvestatud klassimuutujatesse.


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. Andmetüübis on >f4 mis on „big-endian“ 32bit ujukomaga arvud [30].


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. StaMP5`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 vahepunkte.

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 järgevalt kirjeldatse olid MATLAB’is failis „ps_params_default.m“: [16]

- filter_weighting = „P-square“. Kalaude leidmise algoritm (*Weighting scheme*). Kanditaatpikslid korrutatakse läbi kaaludega resampeldamise protsessis vastavalt sellele algoritmile. StaMPS’is oli teiseks vääimaluseks „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 analüüsima.
- clap_low_pass_wavelength = 800. Filter laineepikkusele millest alates enam ei arvestata.
- 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’i 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 lahtutati maha üks. Aga StaMPS’is tehti seda vaid siis kui small_baseline_flag oli väärtusega „N“. Siin protsessis ja kõikidest teistes on kood tehtud seda, 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 funktsooni \texttt{delete} NumPy teekist, kus eemaldatava rea indeks on \texttt{PsFiles} klassis olev \texttt{master_nr}. Viimasest parameetrinist on lahatud üks, et olla võrdne massiivi indeksitega. Lisaks tasub tähele panna, et siin me ei kasuta \texttt{MatrixUtils.delete_master_col} funktsooni, sest kustutatavas massiivis on vaid üks rida ja \texttt{MatrixUtils}'is olev kustutab veergudest.

\texttt{PsFiles} klassist loetakse sisse ja muudetakse ka massiivi \texttt{sort_ind}, mis StaMPS’is oli „la1.mat“ failis olev muutuja \texttt{la}. Massiivi keskmistatakse kasutades NumPy funktsooni \texttt{mean} ilma lisaparametreid. 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 ümberkirjutatud selles osas osas natuke erinevate (täpsemate) tulemustega (võimalik näha testides). Tekkinud ujukomaarv salvestatakse muutujasse \texttt{sort_ind_meaned}. StaMPS’is oli see nimetusega \texttt{inc_mean}. StaMPS’is oli ka võimalus leida \texttt{inc_mean}, kui faili „la“ ei olnud. See leiti valemiga 21 * pi / 180.

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

\[ k = \lambda \ast \text{mean_range} \ast \frac{\sin(\text{sort_ind\_meanded})}{4} \ast \pi \]  

Valemis \texttt{lambda} on lainepikkus klassis \texttt{PsFiles} (muutuja \texttt{wavelength}), \texttt{mean_range} on privaatne parameeter mis säititi konstruktoris ja \texttt{sort_ind\_meanded} on parameeter mis tuli \texttt{load_ps_params} funkstioonist ühe tagastusparameetrina.

Seejärel leitakse \texttt{bperp\_meanded} massiivist maksimaalse ja minimaalse vahem. See on nimetatud koodis kui \texttt{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).

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


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 imaginaararvukuks ja siis võetud eksponent sellest (funktsoon 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 ajakukas protsess.

Seewäärle 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 koolis MATLAB võetakse histogrammi vahemikud servadest, aga NumPy teegis ja Python’is...
võetakse keskelt [32]. Funktsioon tagastab kaks parameetrit milles 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 ei eelnevalt kirjeldatud ning salvestatakse. Tähele tuleb panna, et kui vahesalvestatud juhummassiivi 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 sisendparameetritele `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 veeru. 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.

$$\left\lfloor \frac{x-\text{min}(x)+10^{-6}}{\text{filter\_grid\_size}} \right\rfloor (6)$$

Kus $x$ tähistab massiivi $xy$ veergu. Vähima leidmiseks kasutatakse NumPy funktsiooni `amin`, mis leiab massiivist kõige väikseima elemendi. `filter\_grid\_size` on konstant, mis väärutusteta konstruktoris. Funktsiooni ümardataks üles mille jaoks kasutatakse NumPy funktsiooni `ceil`. Tähele tuleb panna, et valemi nimetajas olev on koodis viidud tüüp $\text{float32}$. Vaikimis on selleks $\text{float64}$, mis oli aga liiga täpne ning tekitas ümardamise käsikus olukorra, kus vastus tuli ühe vörre suurem kui StaMPSis.

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

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


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. Vördlus tehakse funktsioonis `is_gamma_in_change_delta`, kus võrreldakse kas delta on väiksem eelmainitud klassimuutujust ja while tingimuseks on selle negatsioon (Python `not`).
Kõigepealt leitakse grid_ij massiivist maksimaalsed väärtused kasutades NumPy funktsooni 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 funktsoonis 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 funktsoonis.


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


Seejärel leitakse PsTopofit funktsooniga ps_topofit_loop funktsooniga k_ps, c_ps, coh_ps, n_opt, ph_res massiivid.
PsTopofit puhul luuakse kõigepealt objekt, mille käigus initsialiseeritakse massiivid, mis funksioonist ps_topofit_loop tagastatakse. Funktsioonis kutsutakse for-tsüklis välja PsTopofit klassis olev funksioon 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 funksiooniga copy tulemused ja salvestatakse need eraldi muutujatesse. Peale seda kustutatakse objekt, et säästa mälul. Ilma kopeerimiseta ei pruugi kustutamine täielikult õnnestuda, kuna viited objektile on veel alles.

Siis leitakse koherentsuse muutus. See arvutatakse välja valemiga 7.

\[
\text{gamma\_change\_rms} = \sqrt{\sum \frac{(\text{coh\_ps} - \text{coh\_ps\_result})^2}{nr\_ps}}
\]  (7)

Kus ruutjuure, summa ja astendamise jaoks kasutatakse vastavalt NumPy funksioone sqrt, sum, power. coh_ps_result on eelmise tsüklili korra coh_ps muutuja tulemus. Valeni tulemusena tagastatakse murdarv.

Selle valemist lahtatakse eelmise tsüklili korra tulemus ja sellest saab gamma_change_delta. Selle tulemuse absoluutväärtuse järgi vaadatakas kas while-tsükit võib jätkata. Kui võib siis enne uuesti alustamist tehakse veel lisategevusid.

Lisategevustes leitakse uuesti kaalud ehk massiiv weights. Selle jaoks leitakse kõigepealt histogramm, kasutades MatlabUtils klassis olevat funksiooni 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 parameetreid [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 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
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
ühhest suuremad väärtest 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 MMMatlabUtils,
mile 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 üheks viimast elementi.


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


Viimases, kaalude leidmise osas, tuleb tähele panna, et antud implementatsioonis on üks osa puudu. Programmis StaMPS oli eraldi loogika selle jaoks, kui parameeter \textit{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 funksioon on väga ajakulukas. Suurem osa ajast kulub \textit{PsTopofit} klassis oleva funksiooni \textit{ps\_topofit\_loop} tegemisele. Seda põhjusena, et tegemist on suhteliselt suure tsükliga ning tehtavad arvutused \textit{topofit\_fun} funksioonis ei ole kõige triviaalsemad. Nagu jõudlustestidest, peatükk 3.9, nähtub, et olnud selle funksiooni kiirendamisel abil ka Cython’ist. Lisaks ei andnud esialgsete katsetused Numba teegi [34] toel tsükliste paralleelseks muutmise kohest võitu.

3.6.4 Klass \textit{PsSelect}

Protsessiklassis \textit{PsSelect} valitakse eelmise protsessi, \textit{PsEstGamma}, tulemusest saadud andmete põhjal stabiilseimad. Protsess on tehtud StaMPS faili „ps\_select.m” järgi.

Protsessis leitakse muutujad: \textit{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 \textit{PsFiles} objekt ja \textit{PsEstGamma} objekt. Sarnaselt \textit{PsEstGamma}'s olnuga tehakse ka siin privaatseid klassimuutujad, mis StaMPS'is sätiti süsteemi \textit{setparam} funksiooniga. Parameetrid seadistatakse funksioonis „\textit{set\_internal\_params}” ja on järgnevad (vaieväärtsed ja selgitused) [16]:

- \textit{select\_method} = \textit{DESINITY}. Mõnedes kohtades koodis saab eraldi seadistada, et kuidas selekteeritakse piksleid. Antud implementatsioonis kasutatakse selle jaoks andmetüüpi enum, kus on kaks võimalikku väärust, \textit{DESINITY} ja \textit{PERCENT}.
- \textit{gamma\_stdev\_reject} = 0. Kasutatakse, et filtreerida \textit{coh\_thresh\_ind} massiiv. Massiivi jätakse kõik väärtsed mis on sellest väiksemad. Hetkel on implementeeritud ainult väärtsus 0, mis tähendab, et filtreerimist ei toimu. Seda
dokumentatsioonis ei olnud kirjeldatud. Seetõttu on võimalik, et parameetrit enam ei kasutata.

- drop_ifg_index = \texttt{numpy.array([])}
  ehk tühi NumPy array. Siia pannakse indeksid interferogrammidest, mida ei soovi protsessi võtta.

- \texttt{gaussian\_window} = \texttt{np.multiply(np.asmatrix(MatlabUtils.gausswin(7))},
  \texttt{np.asmatrix(MatlabUtils.gausswin(7)).conj().transpose()}. See originaalis tehti koodis sees ja siis salvestati süsteemi. Tegemist on Gaussi aknaga mida kasutatakse \texttt{clap\_filt\_for\_patch} funktsioonis.

Lisaks on seal \texttt{slc\_osf}, \texttt{clap\_alpha}, \texttt{clap\_beta}, \texttt{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 \texttt{PsFiles}. Selle jaoks kasutatakse funktsiooni \texttt{get\_ps\_variables}, nagu tehti faasianalüüsis. Küll aga siin protsessis ei kasutata parameetrit \texttt{nr\_ps}.

Seejärel valitakse massiividest \texttt{ph} ja \texttt{bperp} välja kõik andmed, mis on ajaliselt peale ülepilti. See tehakse sisemises funktsioonis \texttt{filter\_params\_based\_on\_ifgs\_and\_master}. Seal tehakse kõigepealt NumPy \texttt{arange} funktsiooniga arvurida, mis algab 0’ist ja lõppeb numbriga, mis on muutujas \texttt{nr\_ifgs}. Sellest massivist võetakse kõigepealt välja interferogrammid mida ei vaadelda. Selle jaoks kasutatakse NumPy funktsiooni \texttt{setdiff1d}. Mitte vaadeldavate interferogrammide indeksid on massivis \texttt{drop\_ifg\_index}, mis sätiti konstruktoris. Selle tulemusest võetakse, sama funktsiooniga, välja ülepildi indeks, mis tuleb \texttt{PsFiles} objektist.

Seejärel valitakse interferogrammidid, mis ei ole ülepildi omad. Selle jaoks kasutatakse klassis \texttt{PsFiles} olevat funktsiooni \texttt{get\_nr\_ifgs\_copared\_to\_master}, kus parameetriks on võrdlusfunktsoon, mis näitab, et võrrelda tuleb selliselt kus ülepildi kuupäev on suurem kui massiivis olevad väärtused. Funktsiooni tulemusest lahutatakse üks, et olla
võrdne indeksitega. Seda seepärast, et funksioon tagastab arvu mitu interferogrammi 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 jää 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 massiivi 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 lahatatud üks, et saada Python keeles kasutatav indeks.

Lõpptulemusena tagastatakse muutujad \( ifg\_ind, bpep \) ja \( ph \) (viimased filtreeritult).

Seejärel leitakse massiiv \( da \) ja \( da\_max \) funksioonis \( get\_da\_max \). Funksioonist tagastatakse peale \( da\_max \) muutuja ja \( da \). Seda seepärast, et kui massiiv \( da \) on väiksem kui 10 000 elementi suur siis luukse \( da\_massiv \) seal funksioonis. Tehakse massiiv, mis on täis ühtesid. Massiivi suurus on võrdne \( coh\_ps^\prime \)iga.

Kui \( da \) massiiv on suurem eelnimetatud arvust siis \( da \) massiiv sorteeritakse kasutades NumPy funksiooni \( sort \). Sorteeritakse veergude järgi, selle jaoks sättetakse 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 massiivist massiivi mis, on 37 elementi suur. Selle algusse pannakse üks number 0 ja lõppu sorteeritust viimane element. See kõik pannakse kokku NumPy funksiooniga \( concatenate \). 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 funktsoonis leitud muutujad.

Funktsooni lõpus tagastame *DataDTO* objekti leitud muutujatega.

Seejärel leitakse murdarv *max_rand*, mis StaMPS‘is oli nimetatud *max_percent_rand*. Seda tehakse funktsoonis *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 funktsoonid *max* ja *min*) ning leitakse nende vahe. Lahutamise tulemusena saadakse massiiv ja selle elemendid korrutatakse omavahel kasutades NumPy funktsooni *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 väärtsus 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 funktsoonis *get_max_rand*, sest mujal seda väärtust ei kasutata.
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 tõlgendus funktsiooniga `get_ph_path`. Siin nii ei tehta, et vajadusel väljatseekse kohalikult siia klassimuutujatena. Antud massiiv leitakse `for`-tsükli, mille pikkuseks on `coh_thresh_ind` massiivi pikkus.

Enne tsükli algust tehakse täielik massiiv `ph_filt` ja leitakse `nr_i` ja `nr_j` mis toimivad indeksitenega. Sarnaselt nagu `PsEstGamma` protsessis. `PsEstGamma` protsessis leitud `grid_ij` massiivist leitakse maksimaalsed väärtused, nullinda veeru suurim väärtus saab `nr_j`iks ja esimese veeru maksimaalne väärtus saab `nr_j`iks.

Muutuja `ph_filt` tehakse kasutades NumPy funktsiooni `zeros`, mis teeb parameetris antud kujuga massiivi 0-kahuridest [35]. Selle kuju jaoks kasutatakse `clap_win` parameetrit, mis sätiti funktsioonis `set_internal_params`, ja interferogrammide arvu, `nr_ifgs` muutujat `DataDTO` objektit. Kujuks saab `clap_win * clap_win * nr_ifgs`. StaMPS'is oli tehtav massiiv nimetusega `ph_filt2`.


Seda kasutatakse järgmise sammuse, 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 indeksid, `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-tükslis 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 koherentus. Seda tehakse funktsioonis *PsSelect* klassi funktsioonis *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 piirmää. Iga piksel mille standardhälve on sellest suurem eemaldatud
- **weed_max_noise** = sys.maxsize. Piksli müra piirmää. Iga piksel mille müra on suurem sellest eemaldatud.
- **weed_zero_elevation**
- **weed_neighbours**

Eelmisega võrreldes ka parameteer 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 protssessiga. 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 funksioonis load_ps_params ning lõpptulemusena tagastatakse DataDTO.
objekt. See DataDTO ei oma seost PsSelect'i olevaga, peale selle, et see on sisemine klass ja selles hoitakse muutujaid.


PsFiles objektist võetakse muutujad pscands_ij, xy, ph, lonlat, hgt, mis, sarnaselt PsSelect'i tehtud loogikaga filteeritakse. See on märklik, et 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'i tehtud loogikaga filteeritakse. See on märklik, et PsSelect objektist selekteeritakse alati keep_ind järgi, ka siis kui viimane on tühi. Massiiv keep_ind salvestatakse DataDTO klassi nime alla ind.

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 töhjad. See on veaolukord ja programmis tekib viga. Erindit programm ei tagastata, kui algavad arvutused ei järjest tohja.

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

-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 indeksid. Peale seda parandatakse/ täpsustatakse `selectable_ps` massiivi järgnevates funktsioonides.

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 parametrid `weed_standard_dev` ja `weed_max_noise` ei olnud suuremad või vördsed 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.

55
Seejärel eemaldatavat 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. Prosessist 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 vääruste arv ja „psweed.2.edge“ loetud massiiv indeksitest. edge_std ja edge_max väärustest leitakse minimaalsed. Valimiseks kasutatakse „psweed.2.edge“ indekside. selectable_ps massiivi tõeste vääruste arvu kasutatakse, massiivide ps_std, ps_max loomiseks. See saab nende massiivide pikkuseks.


PsWeed klassis on funktsioon, et järgmist 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 tööttu 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 väärnurka „n“.

Funktsiooni initsialiseerimiseks on vaja objekte PsFiles ja PsWeed.

Esmalt leitakse muutujad PsFiles ja PsWeed objektidest. PsFiles klassist leitakse ainult ülepildi 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 öigesti selekteeritud ja filtreeritud siis siin funktsioonis rohkem ei tehta midagi.

Leitud muutujad pannakse sisemisse klassi DataDTO.


Viimasena leitakse massiiv ph_reref, mis leitakse funktsiooniga get_ph_reref. See on ph_patch, kuhu on lisatud masteri veerg, sarnaselt bperp massiivid. Erinevuseks on, et bperp'i 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 fails „ps_calc_ifg_std.m“.
3.7 Andmeskeem

Joonis 4 näitab, kus klassides erinevad muutujad tehakse ja kus neid kasutatakse.
3.8 Pythonis ja Matlabis ekvivalentse koodi loomise probleemidest interpoleerimise funktsiooni näitel


Funktsioon mida näites interpoleerisime oli järgnev:

\[ y = \sin(2\pi \times 30t) + \sin(2\pi \times 60t) \]

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 polüномiaalset interpoleerimist (funktsioonis parameeter \( \text{kind} \) 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 \( \text{interp1d} \) 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.

Sellised erinevused matemaatiliste funktsioonide realiseerimisel illustreerivad antud töö käigus kogunenud ohtralt ajakulu, et leida adekvaatset viisi, kuidas Python’is kirjutatud koodi testida. Kääsesoleva 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 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.


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.

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)).

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 kestvus sekundites

Neljanda sammuna filtreeritakse püsivpeegeldajad. Siin vahepealseid tulemusi ei salvestata.

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 püsivpeegeeldajate valik on vaid sekund aeglasem MATLAB’i omast ja faasi korrigeerimine on iseägi 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


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.

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 teede. Loodud lahendus aitab protsessi kasutada omamata keele MATLAB lisentsi. 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 kood ümber kirjutada kasutades antud teeki.


Kasutatud kirjandus


Lisa A: Interpoleerimiste vahelise võrdluse joonistamise kood

```python
import matplotlib.pyplot as plt
import scipy
import numpy as np
from scripts.utils import MatlabUtils

# Define time intervals
num1 = 20
num2 = 11
num3 = 80

t1 = np.linspace(0, 0.1, num=num1, endpoint=True)
t2 = np.linspace(0, 0.1, num=num2, endpoint=True)
t3 = np.linspace(0, 0.1, num=num3, endpoint=True)

# Generate data
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, 1.277114453952127, 1.647390055566271, 1.75293980492121, 0.5536268776565, 1.1537893450458, 0.657775321121426, 0.180049415982107, -0.150441713008426, -0.3139081378896, -0.304898520098431, -0.160104878924336, 0.064434623902101, 0.263336285748697, 0.349024373653394, 0.260860582334277, -0.057853381714737, -0.513488813788482, -1.01366503534041, -1.4453476589733, -1.6861595824710, -1.68939845829740, -1.43196789089640, -0.93891218196321, -0.25580188035542, 0.475339458228588, 1.12611939152925, 1.58361297863231, 1.7348756852383, 1.6196925411147, 1.28930328691517, 0.831989902727521, 0.358816703073747, -0.0396999042430857, -0.290955267527316, -0.361219085226352, -0.229054685927110, -0.484661257572023e-16, 0.229054685927109, 0.361219085226352,]
```
```python
plt.plot(t3, y_utils_interp, '-', t3, y_utils_interp_q, 'g--', t2, x2, 'o', t3, matlab_actual, 'r--')

plt.savefig("interp")
```