Computations of the geoidal height over Norway using procedure-oriented programming with Fortran

Alexander Nossum, I-IKT, alexanno@stud.ntnu.no, 674107
Martin Vitsø, I-IKT, martinv@stud.ntnu.no, 671848

May 9, 2007

Abstract

This document describes the project assignment in the course TKT4185 held on NTNU. The project assignment is to compute the geoidal heights over Norway, do an analysis based on GPS measured data and implement using procedure-oriented practices in Fortran. The document contains several color images and references to the program and results computed from it. Therefore it is recommended to download the digital version with complete source-code, input-data and results. The complete package can be found at http://folk.ntnu.no/alexanno/skole/GeoideHeights/

Contents

1 Introduction

The goal for this project is to develop a program, using FORTRAN, to solve problems within the scientific field of geodesy. Focus is on the programming part itself and not the scientific aspect of it related to geodesy, so the results of our computations will not be discussed much further than statistical comparison to "expected" values. This project is a part of applied procedure-oriented programming, so architecture-wise we will try to take advantage of the strengths provided by FORTRAN as opposed to many other programming languages, that being fast numerical calculations, simple handling of matrixes and the use of sub-routines and functions.

Computations of geoidal heights are what we will be programming towards. To explain what geoidal heights are, we need to look at the geoid - the equipotential surface of the earth's gravity field which best fits, in a least squares sense, the global mean sea level. In short terms, the geoid is used to relate orthometric heights, physical - on the earth's surface, to ellipsoidic heights - the mathematical surface, an ellipsoid, used to model the earth. The geodial height N anywhere on the earth's surface can be described in relation to the orthometric height h and ellipsoidic height H with the following formula:

$$N = h - H$$

N in the above formula is called $N_{Geometric}$. We will also be using a gravity field model for a more complex computation of N, called $N_{Gravimetric}$, where we use physical constants derived from the gravity field model to define the referrence ellipsoid. We will be using two different global gravity models for our gravimetric calculations, GGM02S and EGM96. We will discuss our programming work, program structure and coding issues. We will also analyze our numerical results, by the help of statistical analysis, to the extent possible given the information we have on expected values.

2 Architecture

The chosen architecture is driven by performance and modifiability attributes. The implementation language was restricted to Fortran in the project assignment - therefore we chose to make the architecture easy to understand and to embrace seperation of functionality. We chose this tactic so the system could easily be modified and the source-code readable. We realize this hits performance slightly but we think it is a well invested tradeoff in a project like this. To gain performance for this tradeoff one can apply better algorithms and coding strategies as discussed further in the implementation part 3. For visualisation we have chosen to use an

adaption of the UML-class diagram (figure 1). Since the implementation language is procedure-oriented the classes represents functions and Fortran specific subroutines, later refered to as functions. The communication between the functions in the system is visualised with dependencies arrows indicating direction and with a keyword for explanation.

The main part of the system is the *computeN* function and its underlaying functions. This is the implementation of the algorithm used to calculate the gravimetric height for a specified point on the globe. The architecture supports and assumes that Dynamic Programming (*Wikipedia* 2007.05.08) is applied to the calculation of P. The algorithm for calculating P consists of several subproblems that will repeat itself with the exact same solution several times during one calculation of the gravimetric height. Not applying Dynamic Programming or another form of caching for the subproblems would hit performance extremely hard and probably break any normal desktop computer. The theoretical part of the algorithm is not discussed in this part and assumed known to the reader.

computeGrid is a function for calculating gravimetric heights in the area $\phi \in (0°, 35°)$ and $\lambda \in (55°, 75°)$ with $0.5°$ increments. In other words a grid with discrete points every $0.5°$ in each direction covering Norway.

GeoideHeights is the main entry point for the application. Its main task is to take user input, read and fetch data from files and to call necessary functions for the computations. The logic for task II and III (Nahavandchi 2007) is also found here since we think of these tasks as supplementary tasks in the assignment and found them to consist of a mixture of reading, writing and calculating data. Therefore we do not seperate them into a seperate function. However, this could easily be achieved with a minimum of effort if one found it interesting or necessary. *GeoideHeights* is also the communication link between the user console and the program.

3 Implementation

The implementation phase was combined with the design phase of the system since we found the project assignment was focused on programming and not on design. The fact that none of the project members had any experience in the Fortran language made us prototype a lot before any architectural decision was made. Implementation of the main algorithm in task I was easy when no concideration where made. After some testing on performance we found that our straight forward approach was insufficient as the runtime was extremely high just for a couple of points. This made us reconsider our approach and search for the main problem

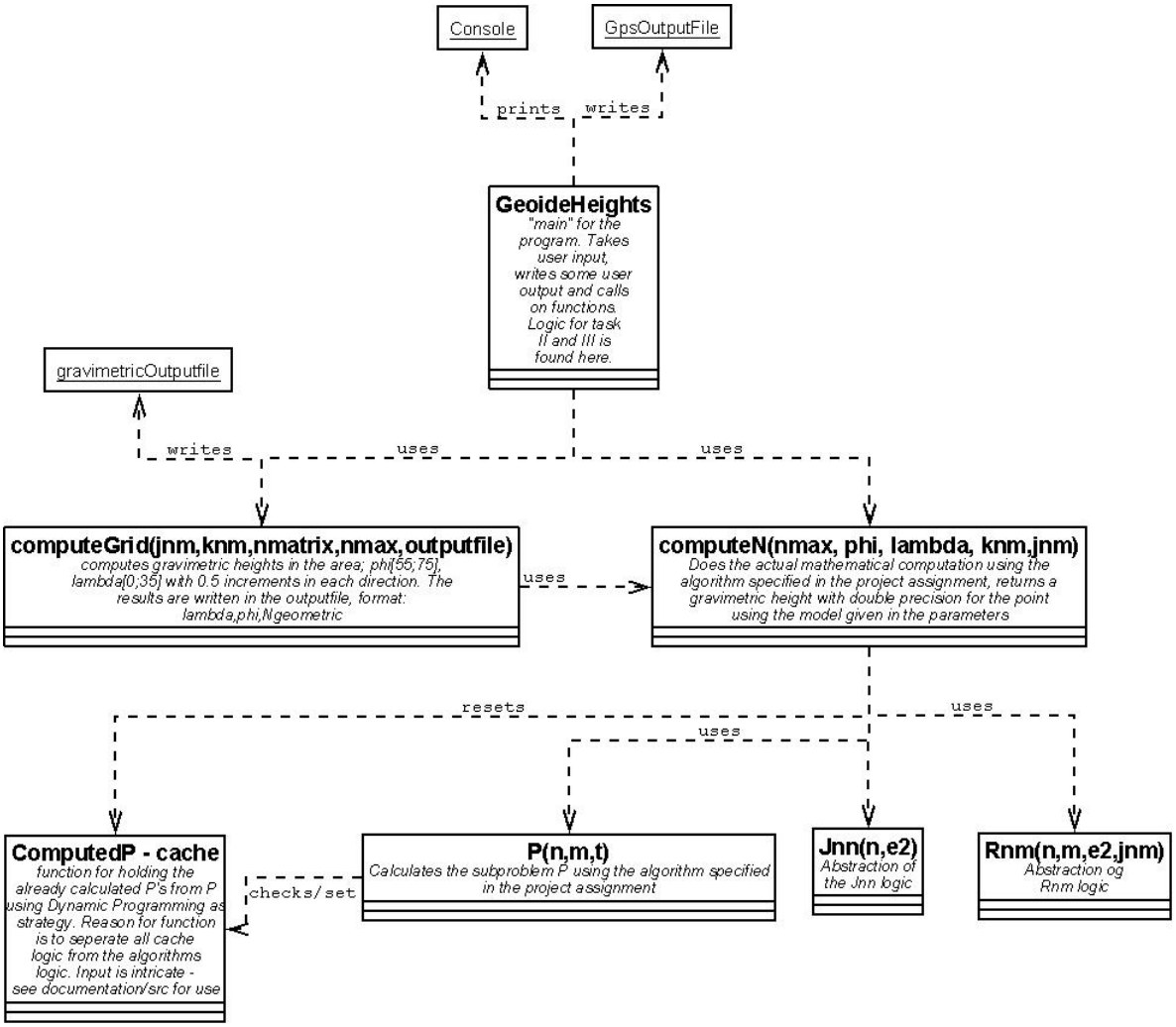

Figure 1: Architecture of the system. UML-classes equals functions and subroutines

area. We expected that the recursive computation of the coefficient P could be optimized with known programming patterns. As the computation of P is a set of subproblems that repeat themselfs we landed on the Dynamic Programming practice (*Wikipedia* 2007.05.08). In its essence this practice holds already computed values in an hash structure which reduces the computational overhead to constant time. When implemented, we were able to run the program and get results.

After a couple of primitive bugfixes we found that the choice of Fortran REAL as precision primitive was not accurate enough. We therefore refactored to the Fortran DOUBLE PRECISION precision primitive. DOUBLE PRECISION gives us the most accurate representation of the number available in Fortran.

Since we had limited experience in programming computational programs in a procedure oriented matter we decided to not tweak the memory use so its fullest extent. This led to a very high memory overhead since we are locating way to much memory for our matrices and arrays. The rationale is the fact that the actual memory use for this overhead is a very small amount of kb. Every computer that would ever run the program has at least 16 mb of free memory and makes the effort of implementing dynamic allocation rather useless.

Other than the approach discussed over we did not apply any other tactics to increase performance. This is because we wanted a readable code and the runtime was acceptable - with a maximum of 10 minutes on a rather slow laptop computer. Implementation tactics that could increase performance is; quicker algorithms, improved datastructures in the caching of P and avoidance of any intrinsic functions such as sin, cos and $\sqrt{x}$ and replace them with their mathematical equivalents. As an example sin would be replaced with a series that could be abrupted after obtaining satisfying accuracy. The intrinsic $\sqrt{x}$ could be replaced with the equivalent $x^{0.5}$. Other tactics can also be applied but we found that applying those tactics would make the code seem very cryptic for a human reader and therefore hit modifiability and not increase performance that much.

As the implementation phase moved forward we refactored the architecture quite heavily. This was a reaction to the fact that we only sketched the initial architecture and did not stated a detailed architecture before implementation. Before implementation we only stated our main focus areas to performance and modifiability. The result of this was a seperation of logic into functions and subroutines and the use of dynamic programming.

3.1 Issues

The resulting program is a mixture of both performance and modifiability with lacks in both directions. This lacks probably comes from our limited experience in this programming style and our limited knowledge about performance tactics in a computational program. The fact that we chose not to implement several

performance tactics probably also hits performance for the program.

The group used Intel Fortran Compiler 9.1 with MS Visual Studio as IDE. This choice was based on advise by the course staff and proved to be ok to develop the program in. On another project of this kind we would reconsider this development environment and stribe to use open source compiler and IDE since they traditionally are better integrated with each other.

4 Numerical investigations

When looking at the figures 2 and 3 over our computed geoide model for both EGM96 and GGM02S, we immediately see that they fit the height line structure and approximate height values we saw in diagrams for this exact geographical area prior to this project.

We have no "correct" values to do a point for point comparison, so we have to assume our model to be correct as we have found no obvious faults in our programming. The basis for our analysis will therefore be the GPS-points and orthometric height calculations based on the data provided in *GPS-Levelling_Stations_TKT4185.dat*. The GPS-data is not provided with additional information about accuracy such as which receivers that are used and which measurement method that was used. These are factors that impact how good the points are. We will assume them to be standard industrial GPS-measurements with a fully determined point accuracy of 1 mm. There are also possible sources for errors in our implementation and computation of $N_{Geometric}$, like bugs and memory issues, but due to the very low complexity in these computations, especially in comparison to $N_{Gravimetric}$, we will, as indicated, use $N_{Geometric}$ as our "correct" values. With this basis, we compare our $N_{Gravimetric}$ to $N_{Geometric}$ to determine how good our $N_{Gravimetric}$ is. See figures 4 and 5.

As we see in *gpsAnalysis.dat*, our results for $N_{Gravimetric}$ show heights above what could be expected from $N_{Geometric}$, especially in the northern area of Norway where the geoidal heights are the lowest. The reasons for this can be numerous. As mentioned, there is a possibility that the values in $N_{Geometric}$ are incorrect. The likelyhood of the same beforementioned issues to apply to $N_{Gravimetric}$ must be considered a lot higher due to the complexity and size of the implementation. The chance for errors to occur in the memory allocation is also higher, considering the implementation of allocated memory space. This is discussed further in the Implementation part (3). In addition to this, the TKT4185_Project.doc states that this project neglects some correction terms and some numerical considerations in the computation of the geoidal height model, especially in the procedure for determination of the "Gravimetric geoidal heights". We do not know how these correction terms and numerical considerations manifest in our calculations, but if

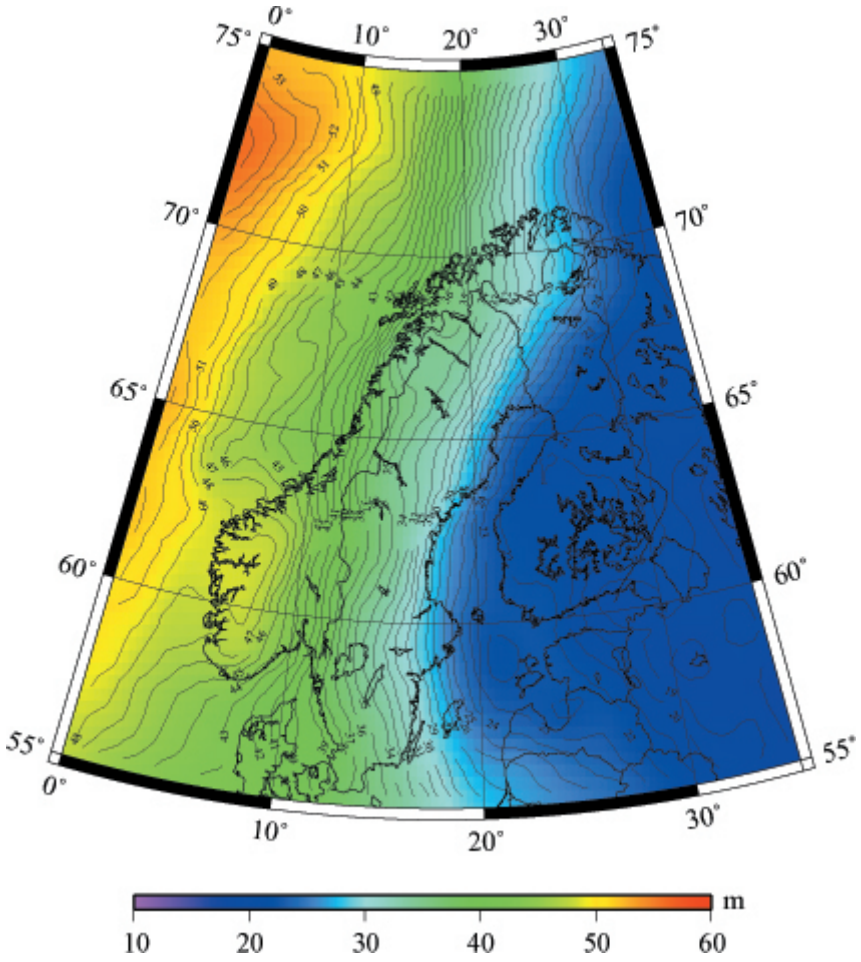

Figure 2: Contour map over the computed heights using *EGM96* as model

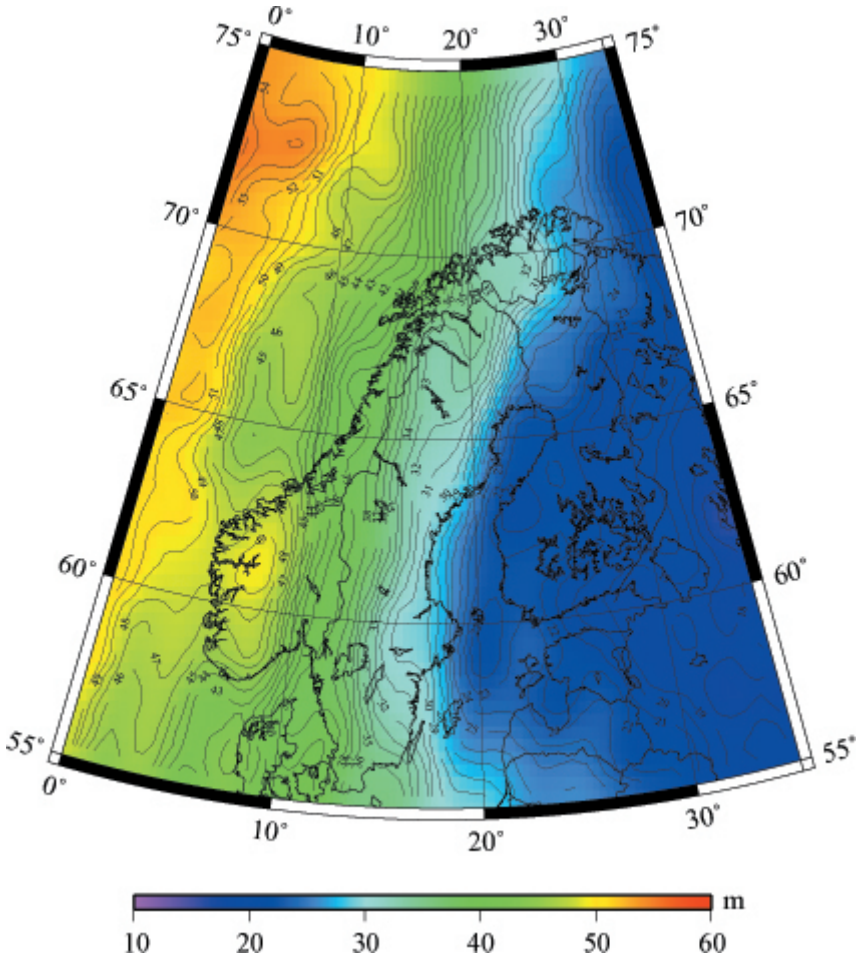

Figure 3: Contour map over the computed heights using *GGM02S* as model

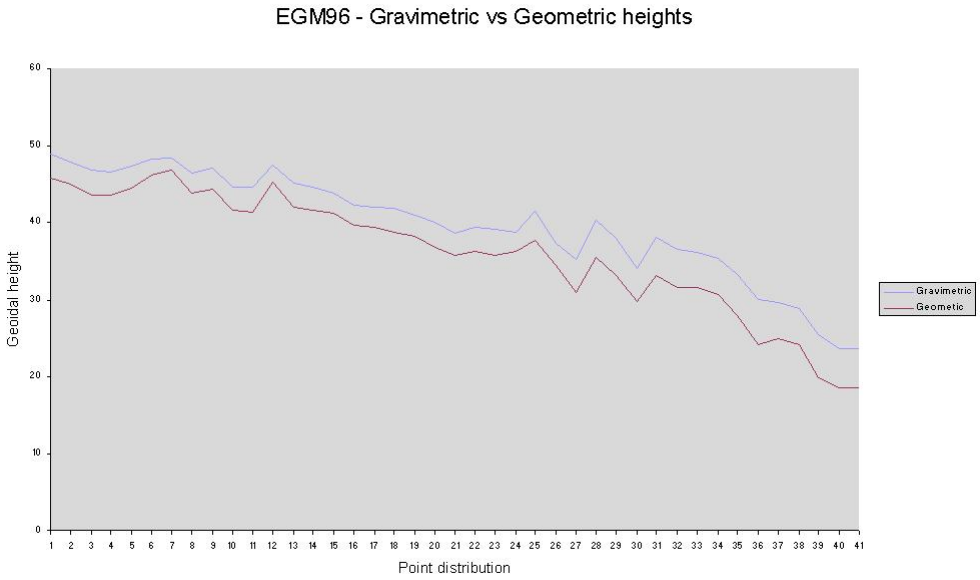

Figure 4: Graph displaying the difference between the geometric and gravimetric height computations using the EGM96 model and measured GPS data

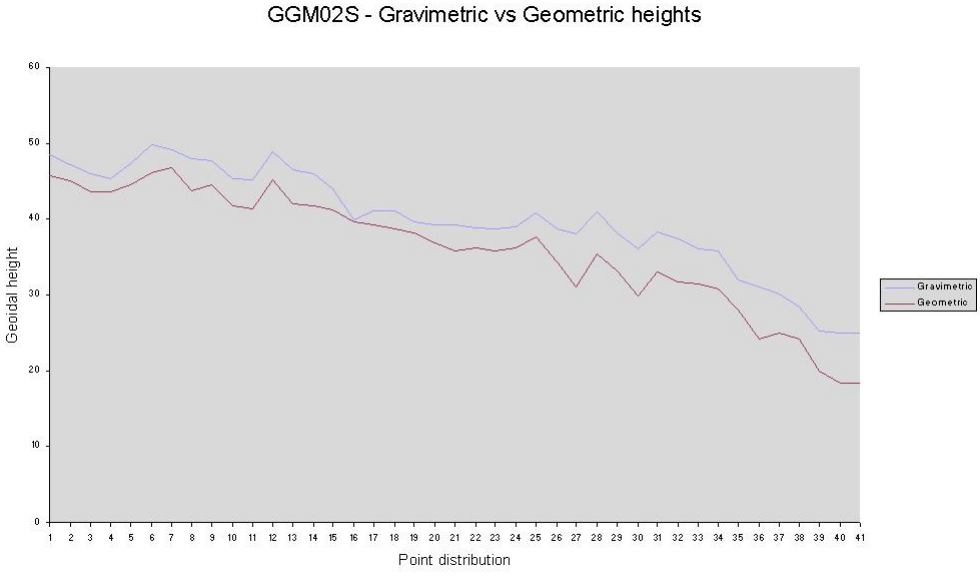

Figure 5: Graph displaying the difference between the geometric and gravimetric height computations using the GGM02S model and measured GPS data

we assume our implementation to be correct, a "calculated" guess from our side is that these terms and conditions deal with the difference between ellipsoid and terrain according to where on the earth's surface one is. This might explain why we experience higher differences in the northernmost areas, but we have no idea if, or how, this culminates in an overall, too high, $N_{Gravimetric}$ or if this is due to a programming error at some level.

5 Conclusion

We encountered problems in testing the result from the final program to assure that they were correct. As mentioned above (4) we know we have values in the correct value area, other than that we can not say anything for sure. This is a problem which leads us to the conclusion that results produced by the program must be considered unreliable since we cannot say anything about the accuracy achieved.

The assignment is done in teams using Fortran. The team used Microsoft Visual Studio as development tool which proved to be badly suited for Fortran development in general and had no intuitive tools for version control such as CVS/SVN. Culminating in development on one computer at a time. This was not a great problem since the amount of coding is limited in this project. In a bigger project we would recommend to use another development environment.

Fortran as a programming language proved to live up to its reputation as a high-performance computational language. That said, this is probably all it can do well. It is a very low-level language that is highly suitable for pure computation tasks. In a practical situation one would probably use a high-level language to work as a wrapper to make input, output and formatting better. Fortran is very old and probably not highly popular which reflects in the developer community. We found no big communities or documentation on the internet and it seems that there are not much development being done on libraries other than commercial packages. With that in mind we still think it is a great language for pure mathematical computations.

The final program reflects some of the issues mentioned above. The program lacks quality in user input and output and usability in general. Possible functionality that could have been implemented is user-input validation and better output to both console (GUI) and files. Wrapping the computation in for example a Java application would made it easier to achieve this. Since this is out of the scope for both the course and project objective we chose to not concentrate much on usability.

References

Bell, K.: 2006, Fortran 90, *Konstruksjonsteknikk* .

Nahavandchi, H.: 2007, Project assignment, *NTNU* .

Wikipedia: 2007.05.08, *http://en.wikipedia.org/wiki/Dynamic_programming* .

A Appendix

A.1 Example input files

A.1.1 Gravity model file

```
The GGM02s model is based on the analysis of 363 days of GRACE in-flight data,
spread between April 4, 2002 and Dec 31, 2003.
GGM02S - complete to harmonic degree 160 - is derived purely from
GRACE satellite data, and is unconstrained by any other information.


product_type                gravity_field
modelname                   GGM02S
max_degree                  160

url                 http://www.csr.utexas.edu/grace/gravity/

    n    m           J_nm                    k_nm
end_of_head ================================================
    2    0  -4.8416970738820E-04   0.0000000000000E+00
    2    1  -2.3983249954865E-10   1.4248881632684E-09
    2    2   2.4393210265716E-06  -1.4002777840038E-06
    3    0   9.5718917114588E-07   0.0000000000000E+00
    3    1   2.0304750265902E-06   2.4817248179590E-07
    3    2   9.0480196240900E-07  -6.1900577831986E-07
    3    3   7.2129356212815E-07   1.4143659403729E-06
    .    .         .                       .
```

A.1.2 GPS inputfile

```
Longitude(degree)  Latitude(degree)    H(ortho. Height)(meter)  h(ellip. Height)(meter)
end_of_head ================================================================================
4.88116282029059   61.0876024514533                32.547                   78.29
5.7250060967603    62.2295051197329               197.3605                  242.373
5.67187933107317   58.7256397016947               54.12                     97.709
```

A.2 Source code

The code listed under are cropped where the lines are bigger than the page width. The code included here is intended for a quick overview of the complete source - not for debugging or specific details. For a complete source-code see http://folk.ntnu.no/alexanno/skole/GeoideHeights/

```
!   GeoideHeights.f90
!
!   REAL FUNCTIONS:
!   GeoideHeights       - Entry point of console application.
!
!
!

!*****************************************************************************
!
!   PROGRAM: GeoideHeights
!
!   PURPOSE:  Calculate geoideheights for the specified area with a specified model
!
!*****************************************************************************

    program GeoideHeights

    implicit none


    !the reference to the file
    INTEGER      ::  file = 1
```

```
!some iterators we can use
INTEGER      ::  i,b,l,pointNo

!vars used in reading the file - trash is just the top-text
INTEGER      ::  ioState
CHARACTER*100    ::  trash
!data from the line just read
INTEGER      ::  n,m,nmax,maxGpsPoint
DOUBLE PRECISION          ::  J,K,phi,lambda,Hort,Hell

! two dim matrix that holds the n,m as indexes and the corresponding jnm value
! the matrixes have indexes from 0 to 360 because of nmax is 360 in one of the dat-files
DOUBLE PRECISION, DIMENSION(2:360,0:360)     ::  jnm
DOUBLE PRECISION, DIMENSION(2:360,0:360)     ::  knm
DOUBLE PRECISION, DIMENSION(55:75,0:35)      ::  nmatrix
! overhead to ensure enough space
DOUBLE PRECISION, DIMENSION(0:1000)          ::  gpsGravimetric,gpsPhi,gpsLambda,gpsGeomHeight,finalDiff
DOUBLE PRECISION     ::  minimum, maximum, sum, average, stdev

DOUBLE PRECISION     ::  tmp
! functions used
DOUBLE PRECISION     ::  computeN

CHARACTER*250    ::  modelFilePath
CHARACTER*250    ::  outputFile
CHARACTER*250    ::  gpsDataFile
CHARACTER*250    ::  gpsOutputFile

WRITE(*,*) '=============================================================================='
WRITE(*,*) '= This program is the result of a school project in the course               ='
WRITE(*,*) '= Applied Procedure Oriented Programming, NTNU spring 2007                  ='
WRITE(*,*) '= The results are not guaranteed what so ever. The source code is available ='
WRITE(*,*) '= in the project report and is released with no warranty.                    ='
WRITE(*,*) '= The program computes geoidal heights in the area                          ='
WRITE(*,*) '= phi=0->35deg, lambda=55->75deg surronding Norway.                         ='
WRITE(*,*) '= Computation method is described in the report that should follow this     ='
WRITE(*,*) '= executable - read this before execution.                                  ='
WRITE(*,*) '= In addition to calculating the geoidal heights in the area, the program   ='
WRITE(*,*) '= also does an analysis based on GPS points supplied in a seperate file    ='
WRITE(*,*) '= supplied in a seperate file.                                              ='
WRITE(*,*) '= User input is a set of filepaths:                                         ='
WRITE(*,*) '= model data file                                                           ='
WRITE(*,*) '= outputfile for the geoidal heights                                        ='
WRITE(*,*) '= GPS datafilemodel data file                                               ='
WRITE(*,*) '= outputfile for GPS analysis                                               ='
WRITE(*,*) '= The filestructure is strict and is specified in the attached report       ='
WRITE(*,*) '=============================================================================='

WRITE(*,*) 'Enter path to the model datafile:'
READ(*,*) modelFilePath

WRITE(*,*) 'Enter path to the outputfile for Norwegian geoidheights'
READ(*,*) outputFile

WRITE(*,*) 'Enter path to the GPS datafile:'
READ(*,*) gpsDataFile

WRITE(*,*) 'Enter path to the outputfile for GPS points analysis'
READ(*,*) gpsOutputFile


!FOR EASY UNIT-TESTING ONLY!
!modelFilePath = 'C:\GGM02S_Model_TKT4185.dat'
!modelFilePath = 'C:\EGM96_Model_TKT4185.dat'
!outputFile = 'C:\ncomputations.dat' ! TESTING ONLY
!gpsDataFile = 'C:\GPS-Levelling-Stations-TKT4185.dat'
!gpsOutputFile = 'C:\gpsAnalysis.dat'

!open the file
OPEN (file, FILE = modelFilePath , FORM='FORMATTED', STATUS='OLD')

! Read the datafile and save it in arrays jnm and knm
!represents the current linenumber
i = 0
DO
    IF (i == 0) THEN
        READ(file,*,IOSTAT=ioState) trash
        IF(trash == 'end_of_head') THEN
            i = 1 ! parsed the header, read the data instead
        END IF
    ELSE
        READ(file,*,IOSTAT=ioState) n,m,J,K
        jnm(n,m) = J
```

```
            knm(n,m) = K
            nmax = n
        END IF

        IF (ioState < 0) THEN ! end of file reached
            exit
        END IF
    END DO

    WRITE(*,*) 'Computing geoidal heights over Norway with 0.5 increments'
    CALL computeGrid(jnm,knm,nmatrix, nmax, outputFile)
    WRITE(*,*) 'finished Computing - results in: ',outputFile

    ! Reads gps data - calculates the difference in heights -> geometric height
    OPEN (file, FILE = gpsDataFile , FORM='FORMATTED', STATUS='OLD')
    i = 0 !stupid iterator
    l = 0 !pointNumber
    DO
        IF(i == 0) THEN
            READ(file,*,IOSTAT=ioState) trash
            IF(trash == 'end_of_head') THEN
                i = 1
            END IF
        ELSE
            READ(file,*,IOSTAT=ioState) phi, Lambda, Hort, Hell
            gpsPhi(l) = phi
            gpsLambda(l) = lambda
            gpsGeomHeight(l) = Hell - Hort ! Task II
            maxGpsPoint = l;
            l = l+1
        END IF

        IF(ioState < 0) THEN ! end of file
            exit
        END IF
    END DO

    ! Process the GPS points - TASK II and III

    ! compute N gravimetric for gps-points
    WRITE(*,*) 'Computing gravimetric height for the GPS point using the selected model'
    OPEN(18,FILE = gpsOutputFile , FORM='FORMATTED', STATUS='UNKNOWN')
    WRITE(18,*) '=============================================================='
    WRITE(18,*) 'Analysis of GPS points given in',gpsDataFile,''
    WRITE(18,*) 'The model used for calculation is:',modelFilePath
    WRITE(18,*) 'At the end of the file statistical analysis can be found'
    WRITE(18,*) 'The list is in the same order as the inputfile'
    WRITE(18,*) 'Gravimetric height, Geometric height, Differance'
    WRITE(18,*) 'end_of_head ======================================================='

    DO pointNo=0,maxGpsPoint,1
        gpsGravimetric(pointNo) = computeN(nmax, gpsLambda(pointNo),gpsPhi(pointNo), knm, jnm)
        finalDiff(pointNo) = abs(gpsGeomHeight(pointNo)-gpsGravimetric(pointNo));
        WRITE(18,*) gpsGravimetric(pointNo),gpsGeomHeight(pointNo),finalDiff(pointNo)
    END DO

    sum = 0.0
    minimum = 999999.0
    maximum = -999999.0
    average = 0.0
    stdev = 0.0
    ! statistical analysis
    DO pointNo=0,maxGpsPoint,1
        IF(minimum > finalDiff(pointNo)) THEN
            minimum = finalDiff(pointNo)
        END IF
        IF(maximum < finalDiff(pointNo)) THEN
            maximum = finalDiff(pointNo)
        END IF

        sum = sum + finalDiff(pointNo)
    END DO

    ! maxGpsPoint starts at 0 and therefore lacks one element
    average = sum/(maxGpsPoint + 1)

    !compute standard deviation
    DO pointNo=0,maxGpsPoint,1
        ! ONLY temporary use
        stdev = stdev + (finalDiff(pointNo) - average)**2
    END DO

    stdev = SQRT(stdev/(maxGpsPoint)) !No need to add one for lack of element since formula is (n-1)
```

```
WRITE(18,*) 'end_of_data ==========================================================='
WRITE(18,*) 'STATISTICAL_ANALYSIS_OF_THE_DIFFERENCE'
WRITE(18,*) 'end_of_head ==========================================================='
WRITE(18,*) 'Minimum:',minimum
WRITE(18,*) 'Maximum:',maximum
WRITE(18,*) 'Average:',average
WRITE(18,*) 'Standard_deviation_based_on_sample:',stdev
WRITE(18,*) 'end_of_data ==========================================================='

WRITE(*,*) 'Finished_GPS_point_analysis_-_results_written_to:',gpsOutputFile
WRITE(*,*) 'Finished_total_computation'
stop
end program GeoideHeights


!=======================================
!
!
! Author: Alexander Nossum
!
!
!
!
!
!
!=======================================
SUBROUTINE computeGrid (jnm, knm, nmatrix, nmax, outputFile)

IMPLICIT none
!input parameters, does not change during computation
DOUBLE PRECISION, DIMENSION (55:75,0:35)     ::  nmatrix
DOUBLE PRECISION, DIMENSION (2:360,0:360)    ::  jnm
DOUBLE PRECISION, DIMENSION (2:360,0:360)    ::  knm

DOUBLE PRECISION     ::  phi,lambda
!user input
CHARACTER*250    ::  outputFile
INTEGER          ::  nmax

! Functions used
DOUBLE PRECISION :: computeN

nmatrix = 0.0

OPEN(15,FILE = outputFile , FORM='FORMATTED', STATUS='UNKNOWN')

DO phi=55.0,75.0,0.5
    DO lambda=0.0,35.0,0.5
        nmatrix(phi,lambda) = computeN(nmax, phi, lambda, knm, jnm)
        WRITE(15,*) lambda,phi,nmatrix(phi,lambda)
    END DO
END DO

END SUBROUTINE

! ==============
! computes the geoidal height of a point using data from inputparameters
! ==============
DOUBLE PRECISION FUNCTION computeN(nmax,phi,lambda, knm, jnm)
    !constants given for the computation
    DOUBLE PRECISION, INTENT(IN)      :: phi,lambda
    INTEGER, INTENT(IN) ::  nmax
    DOUBLE PRECISION, DIMENSION (2:360,0:360)    ::  jnm
    DOUBLE PRECISION, DIMENSION (2:360,0:360)    ::  knm

    ! Constants given
    DOUBLE PRECISION     ::  e2 = 0.006694280023
    DOUBLE PRECISION     ::  a = 6378137.0000
    DOUBLE PRECISION     ::  b = 6356752.3141
    DOUBLE PRECISION     ::  f1 = 298.257222101
    DOUBLE PRECISION     ::  R = 6371000.7900
    DOUBLE PRECISION     ::  GM = 3986005E08
    DOUBLE PRECISION     ::  g = 9.81

    !tmp var for loop computations
    DOUBLE PRECISION     ::  innerLoop, loopComp

    INTEGER              ::  n,m !iterators
    DOUBLE PRECISION     ::  phiRad, lambdaRad
    DOUBLE PRECISION, DIMENSION(2:360,0:360)     ::  Pnm ! ensures enough space with overhead
    REAL     ::  trashCan
    DOUBLE PRECISION     ::  pi
```

```
      ! Functions used
      DOUBLE PRECISION      ::  Rnm, P, computedP

      !initialize variables
      innerLoop = 0.0
      loopComp = 0.0
      pi = 4.0*ATAN(1.0) ! the mathematical constant

      ! modify phi and lambda to radians
      phiRad = phi * pi/180.0
      lambdaRad = lambda * pi/180.0

      computeN = 0.0
      !reset the cache for P
      trashCan = computedP(0,0,0,1)
      !do the loop computation (the doubleSum)
      DO n=2,nmax
           DO m=0,n
                IF(Pnm(n,m) == 0.0) THEN
                     Pnm(n,m) = P(n,m,sin(phiRad)) !compute and save the P for current phi
                END IF
                innerLoop = innerLoop + (((Rnm(n,m,e2,jnm(n,m)) * cos(m*lambdaRad) + knm(n,m)*sin(m*lambdaRad)) * Pr
           END DO
           loopComp = loopComp + (((a/R)**n) * innerLoop)
           innerLoop = 0.0 ! reset the innerLoop var
      END DO
      computeN = (GM/(R*g)) * loopComp ! the final geoidal height
      ! return gravimetric height N
      RETURN;
END

! private REAL FUNCTION for computing Rnm
!
DOUBLE PRECISION FUNCTION Rnm(n,m,e2,jnm)
      IMPLICIT none
      INTEGER, INTENT(IN) ::  n,m
      DOUBLE PRECISION, INTENT(IN)     ::  e2
      DOUBLE PRECISION      ::  Jnn, jnm

      Rnm = 0.0

      IF (m == 0) THEN
           Rnm = jnm - Jnn(n,e2)
      ELSE
           Rnm = jnm
      END IF

RETURN
END

! Private REAL FUNCTION for computing Jnn
!
DOUBLE PRECISION FUNCTION Jnn(n,e2)
      IMPLICIT none
      INTEGER, INTENT(IN) ::  n
      DOUBLE PRECISION, INTENT(IN)     ::  e2 ! constant defined in header
      DOUBLE PRECISION      ::  J2 = 0.108263E-2 ! constant for the GRS80 referance system
      Jnn = 0.0

      IF(n == 2) THEN
           Jnn = -J2/SQRT(5.)
           RETURN
      ELSE IF(MOD(n,2) == 0) THEN !n is even
           Jnn = ((-1.)**(n/2.)) * (((3.*(sqrt(e2)**n) * (1. - (n/2.) + (5./2.) * (J2/e2) * n)))/((n+1.) * (n+3.)
           RETURN
      ELSE IF(MOD(n,2) .NE. 0) THEN !n is odd
           Jnn = 0.0
           RETURN
      END IF

RETURN
END


! Computes the P in the gravimetric height computation
! addPValue is a new computed p-value, when .NE. 0 it gets inserted
DOUBLE PRECISION FUNCTION computedP(n,m,addPValue,init)
      INTEGER, INTENT(IN)                ::  n,m
      DOUBLE PRECISION, INTENT(IN)                    ::  addPValue
      DOUBLE PRECISION, DIMENSION(0:360,0:360)     ::  tmpComputedP
      INTEGER                            ::  i,j

      IF(init == 1) THEN
           DO i=0,360
```

```
            DO j=0,360
                tmpComputedP(i,j) = 0
            END DO
        END DO
        computedP = tmpComputedP(0,0) !trash
        RETURN !trash
    END IF

    IF(tmpComputedP(n,m) .NE. 0) THEN
        !print *,'notnull',n,m,tmpComputedP(n,m)
        computedP = tmpComputedP(n,m)
        RETURN
    ELSE IF(addPValue .NE. 0) THEN ! new P-computation - add to array
        !print *,'adding p',n,m,addPValue,tmpComputedP(n,m)

        tmpComputedP(n,m) = addPValue
        computedP = tmpComputedP(n,m) ! the return value
    ELSE
        computedP = addPValue
    END IF
RETURN
END

! Acts as a facade for the computation
! The REAL FUNCTION gets called recursively and determines which subREAL FUNCTION to use
! This REAL FUNCTION uses dynamic programming
! it saves already computed values instead of computing them several times
! This is to reduce the complexity, maybe we get linear O(n) on runtime
DOUBLE PRECISION FUNCTION P(n,m,t)
    IMPLICIT none
    ! the input parameters
    INTEGER, INTENT(IN) :: n,m
    DOUBLE PRECISION, INTENT(IN) :: t
    DOUBLE PRECISION :: tmp
    ! Defines REAL FUNCTIONs used by this REAL FUNCTION
    DOUBLE PRECISION :: P1,P2,P3,P4
    DOUBLE PRECISION :: computedP

    !we have already computed P return from the list
    IF(computedP(n,m,0,0) .NE. 0) THEN
        P = computedP(n,m,0,0)
        RETURN
    END IF

    !initialize the return variable
    P = 0.0

    IF(n >= 2 .AND. m == 0) THEN
        P = P1(n,m,t)
    ELSE IF(n >= 3 .AND. m >= 1 .AND. m <= (n-2)) THEN
        P = P2(n,m,t)
    ELSE IF(n >= 1 .AND. m == (n-1)) THEN
        P = P3(n,m,t)
    ELSE IF(n >= 2 .AND. m == n) THEN
        P = P4(n,m,t)
    ELSE IF(n == 0 .AND. m == 0) THEN
        P = 1.
    ELSE IF(n == 1 .AND. m == 0) THEN
        P = t*SQRT(3.)
    ELSE IF(n == 1 .AND. m == 1) THEN
        P = SQRT(3.) * SQRT(1.-(t**2))
    ELSE IF(n == 2 .AND. m == 0) THEN
        P = SQRT(5.) * ((3./2.) * (t**2) - 0.5)
    ELSE IF(n == 2 .AND. m == 1) THEN
        P = t * SQRT(15.) * SQRT(1. - (t**2))
    ELSE
        print *,'SOMETHINGS_WRONG_IN_PComputations',n,m
        pause
        RETURN ! we do not have an expression to calculate P
    END IF
    P = computedP(n,m,P,0) ! save the computed P in the array

RETURN
END

! Covers the area n>=2,m==0
DOUBLE PRECISION FUNCTION P1(n,m,t)
    IMPLICIT none
    ! the input parameters
    INTEGER, INTENT(IN) :: n,m
    DOUBLE PRECISION, INTENT(IN) :: t
    ! defines the REAL FUNCTION used by this REAL FUNCTION
    DOUBLE PRECISION :: P
```

```
      P1 = 0.0

      P1 = -((SQRT(2.*n - 1.)/n) * ((n-1.)/(SQRT(2.*n - 3))) * P(n-2,0,t)) + t * (SQRT(2.*n + 1.)/n)*SQRT(2.*n -

RETURN
END

! Covers the area n>=3,1<=m<=n-2
DOUBLE PRECISION FUNCTION P2(n,m,t)
      IMPLICIT none
      ! the input parameters
      INTEGER, INTENT(IN) ::  n,m
      DOUBLE PRECISION, INTENT(IN)     ::  t
      ! defines the REAL FUNCTION used by this REAL FUNCTION
      DOUBLE PRECISION     ::  P
      P2 = 0.0

      P2 = -SQRT(((2.*n + 1.)*(n + m - 1.)*(n - m - 1.))/((2.*n - 3)*(n + m)*(n - m))) * P(n-2,m,t) + t*SQRT(((2.*
RETURN
END

! Covers the area n>=1,m==n-1
DOUBLE PRECISION FUNCTION P3(n,m,t)
      IMPLICIT none
      ! the input parameters
      INTEGER, INTENT(IN) ::  n,m
      DOUBLE PRECISION, INTENT(IN)     ::  t
      ! defines the REAL FUNCTION used by this REAL FUNCTION
      DOUBLE PRECISION     ::  P
      P3 = 0.0

      P3 = t*SQRT(2.*n + 1) * P(n-1,n-1,t)
RETURN
END

! Covers the area n>=2,m==n
DOUBLE PRECISION FUNCTION P4(n,m,t)
      IMPLICIT none
      ! the input parameters
      INTEGER, INTENT(IN) ::  n,m
      DOUBLE PRECISION, INTENT(IN)     ::  t
      ! defines the REAL FUNCTION used by this REAL FUNCTION
      DOUBLE PRECISION     ::  P
      P4 = 0.0

      P4 = SQRT((2.*n + 1)/(2.*n)) * SQRT(1.-(t**2)) * P(n-1,n-1,t)
RETURN
END
```

A.3 Optional figures

A.3.1 Different views on the geoidal heights

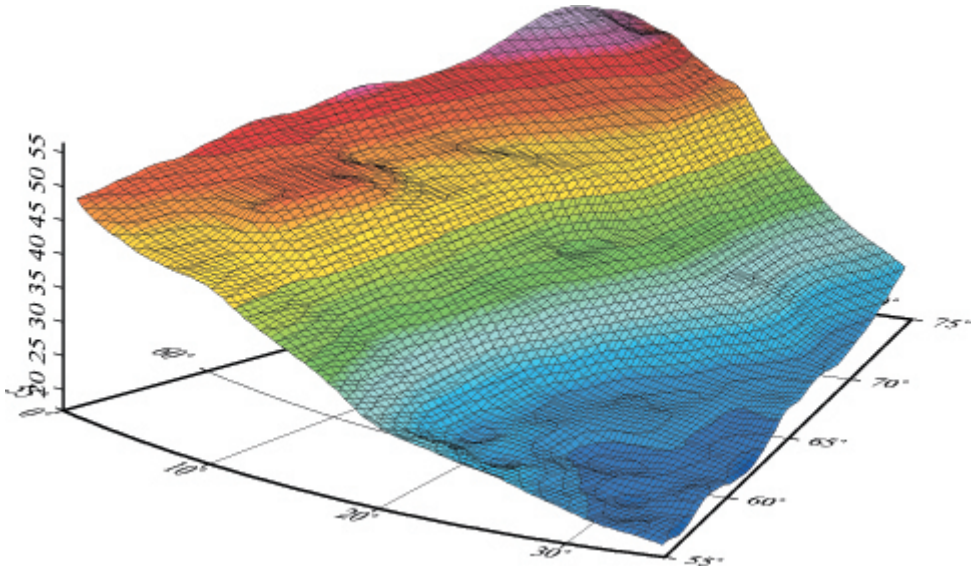

Figure 6: 3D visualisation over the computed geoidal heights using *EGM96* as model

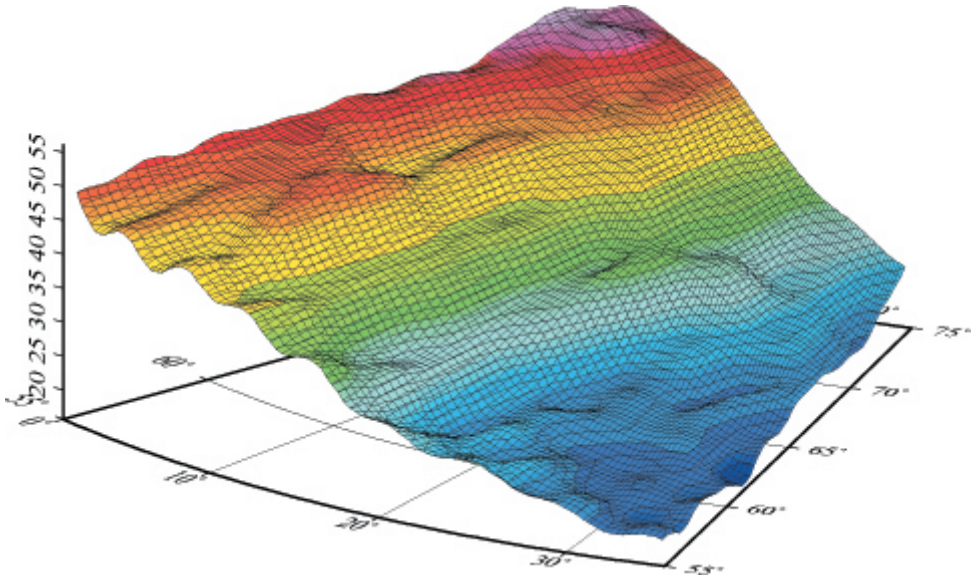

Figure 7: 3D visualisation over the computed geoidal heights using *GGM02S* as model

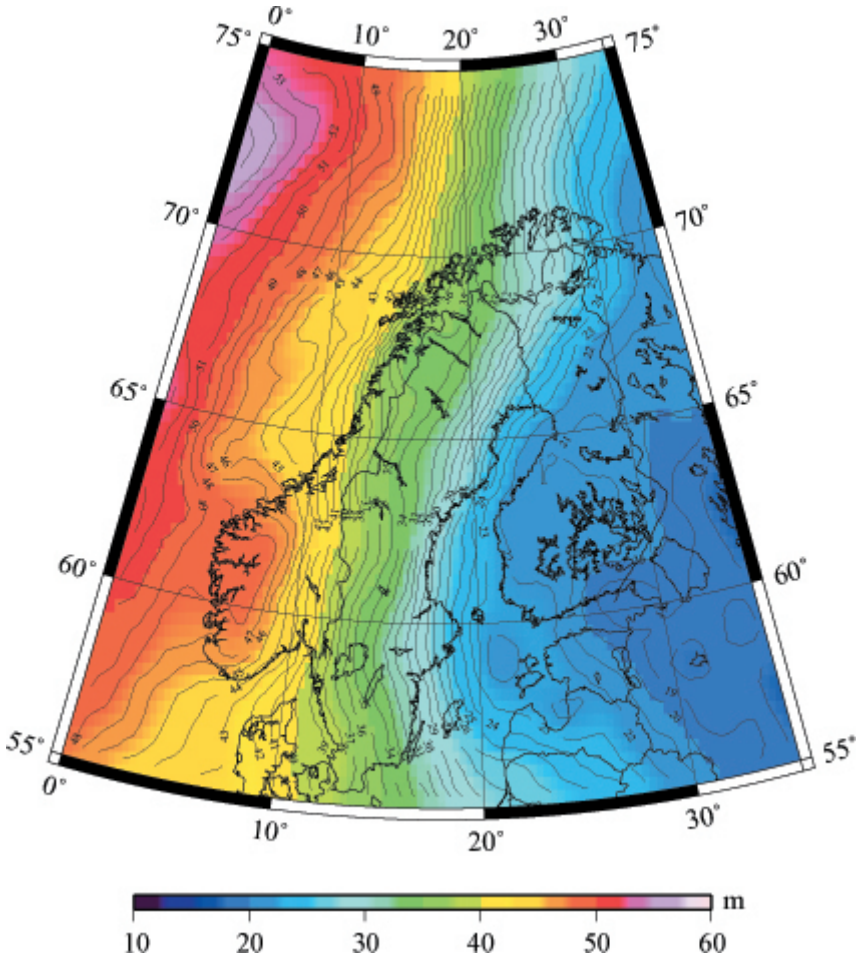

Figure 8: Contour map over the computed heights using *EGM96* as model

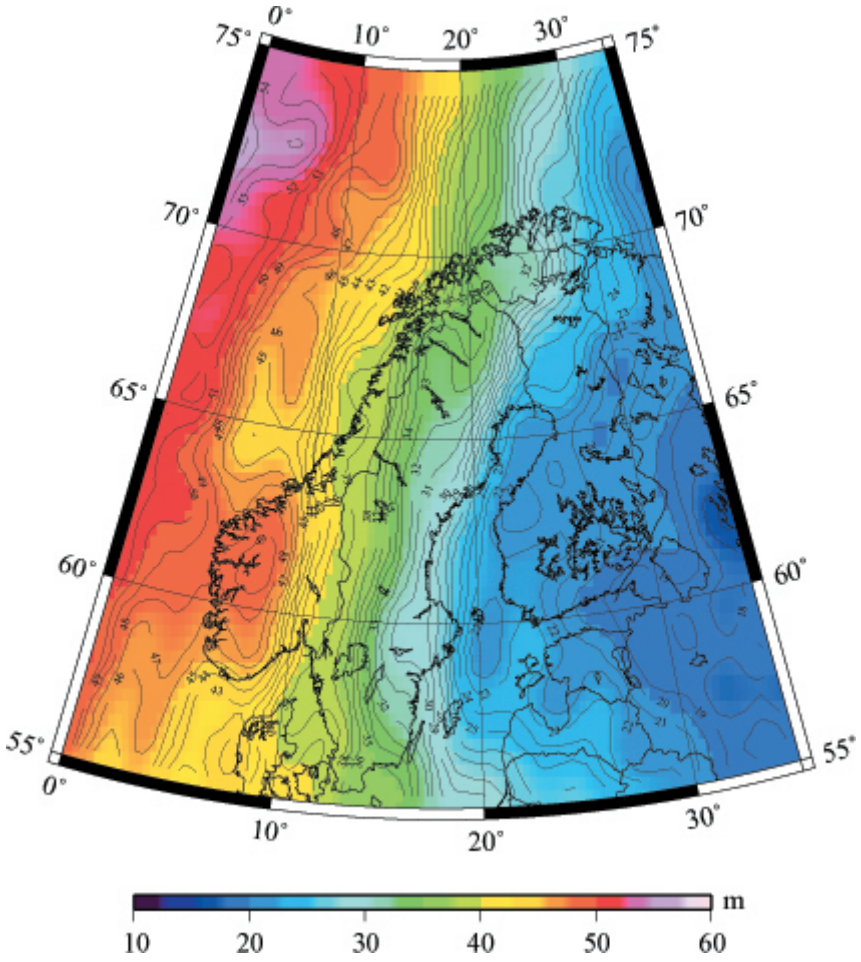

Figure 9: Contour map over the computed heights using *GGM02S* as model

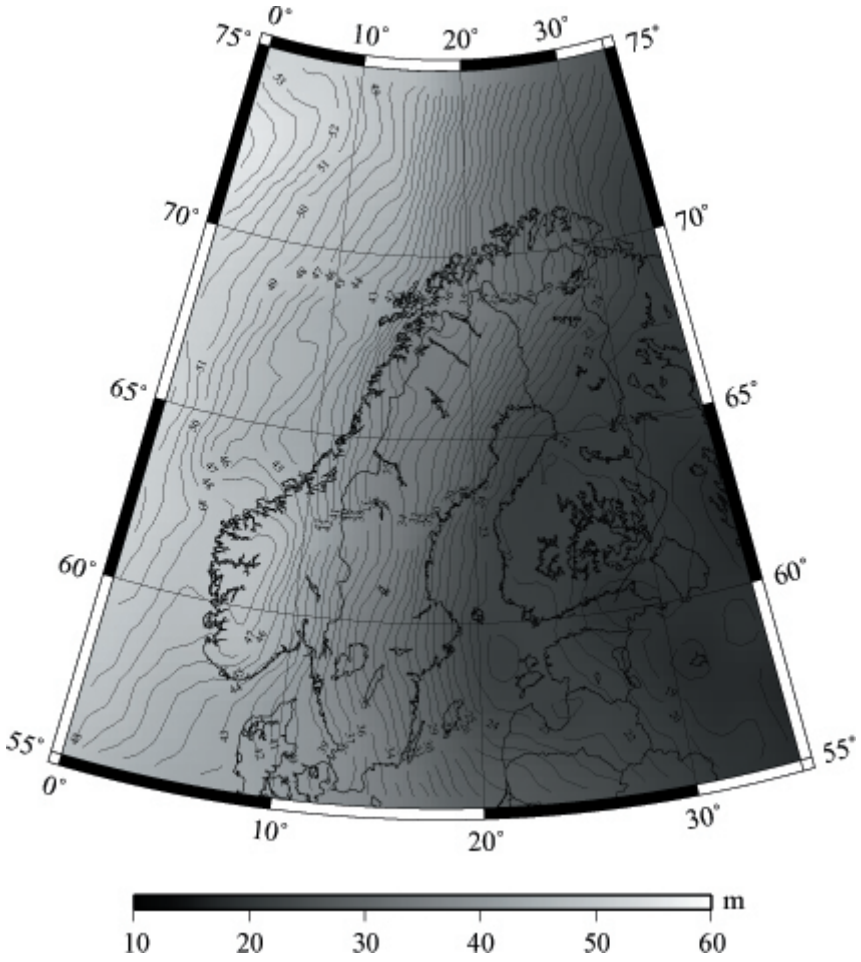

Figure 10: Contour map over the computed heights using *EGM96* as model

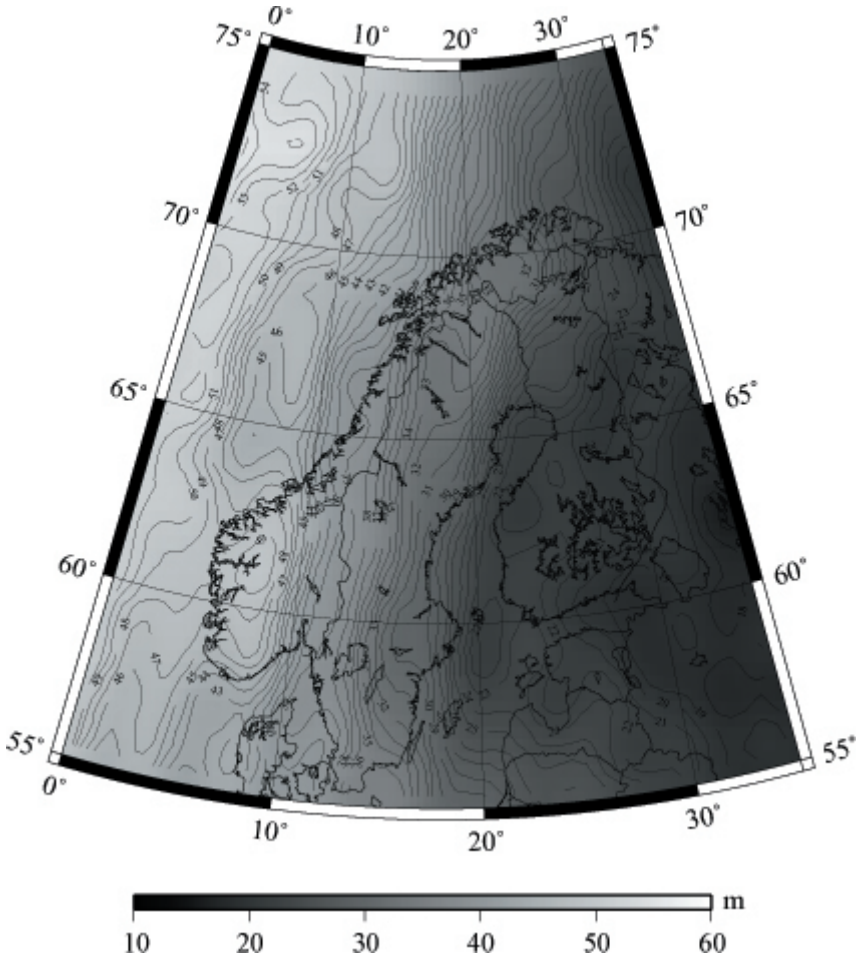

Figure 11: Contour map over the computed heights using *GGM02S* as model