ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(СПбГУ)
Кафедра картографии и геоинформатики

Кузнецова Василина Михайловна

Разработка алгоритмов обработки относительных фазовых определений. Их программная реализация.

Выпускная квалификационная работа бакалавра
по направлению 021300 «Картография и геоинформатика»

Научный руководитель:
к.т.н., доцент С. В. Тюрин
_____
«____»_____2016

Заведующий кафедрой:
к.г.н., доцент Е. Г. Капралов
_____
«____»_____2016

Санкт-Петербург
2016

Оглавление

Введение

Развитие науки влечет за собой изменения в методологической базе любой дисциплины. Так и в геодезии традиционные способы все чаще уступают свое место использованию спутниковых измерений в определении координат. Такие современные способы измерений позволяют с высокой точностью определять положения объектов, при этом исчезла необходимость в зрительной видимости между определяемыми объектами. Кроме того, расстояния между пунктами во время геодезических работ также может быть увеличено по сравнению с традиционными измерениями.

В геодезии в основном используют относительные измерения, так как с их помощью достигается максимальная точность определения координат. Они заключаются в определении сдвига по фазе между колебаниями сигнала, принятого от спутника, и собственными колебаниями. При этом фазовые измерения должны производиться одновременно на нескольких станциях (базовой с известными координатами и определяемых).

Обработка таких измерений есть в существующем программном обеспечении. Однако, к сожалению, в основном все они является зарубежными, дорогими и закрытыми и российских аналогов не имеют. Используемые в их основе математические алгоритмы неизвестны, что является большим недостатком для каких-либо научных, исследовательских, а также образовательных целей. Поэтому возникла необходимость в создании собственного продукта, совершающего обработку геодезических спутниковых измерений.

Спутниковые измерения могут быть использованы в частности для выполнения работ по мониторингу деформаций природных и инженерных объектов. Этот вид геодезических работ очень часто применяется в мире, так как позволяет предотвратить серьезные повреждения гидротехнических сооружений, архитектурных памятников и других инженерных объектов. Программное обеспечение для этого вида работ в настоящее время активно создается и развивается в разных странах как в исследовательских центрах, так и в коммерческих организациях. В России геодезический мониторинг тоже является востребованным, поэтому есть необходимость в создании нового, удобного для отечественных специалистов программного обеспечения, отвечающего всем требованиям и использующего новейшие технологии. Так как для мониторинга необходимы измерения с максимальной точностью, то актуальной проблемой является создание программного обеспечения, основанного на относительных фазовых методах.

Таким образом, целью работы является разработка алгоритмов обработки относительных фазовых измерений.

Для достижения поставленной цели необходимо было решить следующие задачи:

- разработать и реализовать алгоритм для чтения RINEX-файлов версии 3.0;
- разработать и реализовать алгоритм для вычисления координат спутника по бортовым эфемеридам;
- разработать и реализовать алгоритм для вычисления координат местности относительным фазовым методом.

Структура работы представлена 5 главами. В первой главе приведен обзор литературы и известных решений. Вторая глава посвящена формализации задачи и содержит математические выкладки и уравнения для вычисления координат пункта относительным фазовым методом. В третьей главе приведены алгоритмы вычисления координат спутника по бортовым эфемеридам. Четвертая глава содержит описания алгоритмов в виде блок-схем и реализации программных модулей. В пятой главе приведен анализ результатов, получаемых при обработке измерений с помощью созданного программного продукта. В заключении приведены выводы работы и указаны направления дальнейшего развития. В приложение включен исходный код разработанных программных модулей

Глава 1. Обзор существующей литературы и программного обеспечения

1.1. ОБЗОР ЛИТЕРАТУРЫ

В качестве материалов по теме работы использовались научная литература, учебно-методические пособия, а также нормативно-правовая документация Российской Федерации.

Главным источником, в котором подробно раскрыты теоретические основы спутниковых технологий в геодезии, является монография Антоновича К.М. В двух томах этого труда объясняется устройство спутниковых радионавигационных систем, а также методы наблюдений и основы обработки таких измерений.

Математические алгоритмы фазовых относительных определений подробнее рассмотрены у Коугия В.А., где уделено внимание вычислению одинарных, двойных и тройных разностей. Также по теме работы был рассмотрен ГОСТ Р 53608 — 2009 «Глобальная навигационная спутниковая система. Методы и технологии выполнения геодезических и землеустроительных работ. Разрешение неоднозначности фазовых измерений псевдодальности. Основные положения».

При проведении вычислений возникла необходимость в определении координат спутника по бортовым эфемеридам (Кеплеровым элементам). Алгоритм вычислений был рассмотрен по учебному пособию по астрономической практике «Небесные и земные координаты» Санкт-Петербургского государственного университета.

Кроме того, при выполнении работы использовались труды таких отечественных авторов, как Губанов В.С. и Маркузе Ю.И. При разработке использовалась рассмотренная ими теория математической обработки геодезических измерений.

1.2. ОБЗОР СУЩЕСТВУЮЩЕГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ МОНИТОРИНГА ДЕФОРМАЦИЙ

В связи с тем, что разрабатываемый алгоритм по определению координат относительными фазовыми методами предполагается использовать для такого вида геодезических работ, как мониторинг деформаций, то перед разработкой и созданием алгоритма необходимо ознакомиться с крупнейшими из существующих на рынке продуктами для данного вида работ.

Одним из хорошо известных пакетов программ является GOCA — (GNSS/LPS/LS-based Online Control and Alarm Systems), разрабатываемый в Институте прикладных научных исследований города Карлсруэ в Германии. Данный продукт является мультисенсорной системой, не зависящей от источников используемых

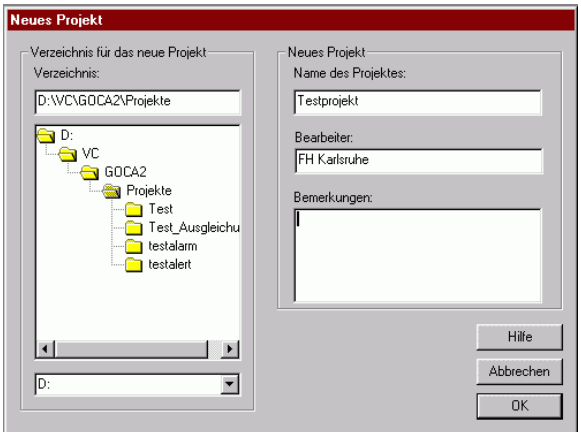

Рисунок 1. Интерфейс ПО GOCA (http://goca.info/index_ru.html)

данных. Данная система обрабатывает спутниковые и другие измерения и анализирует полученные результаты, фильтрует данные и составляет оповещения, если положение объекта изменилось и достигло критического состояния, которое задает пользователь.

Другой рассмотренный программный продукт — GeoMoS (Leica), разработанный в Швейцарии. В основном он использует данные с электронных тахеометров, GPS приемников, цифровых нивелиров и других геодезических оборудований. Состоит из трех модулей: Monitor — приложение, отвечающее за сбор данных; Analyzer — приложение по визуализации и пост-обработке данных; Adjustment — приложение, в котором производится уравнивание сетей и анализ деформаций.

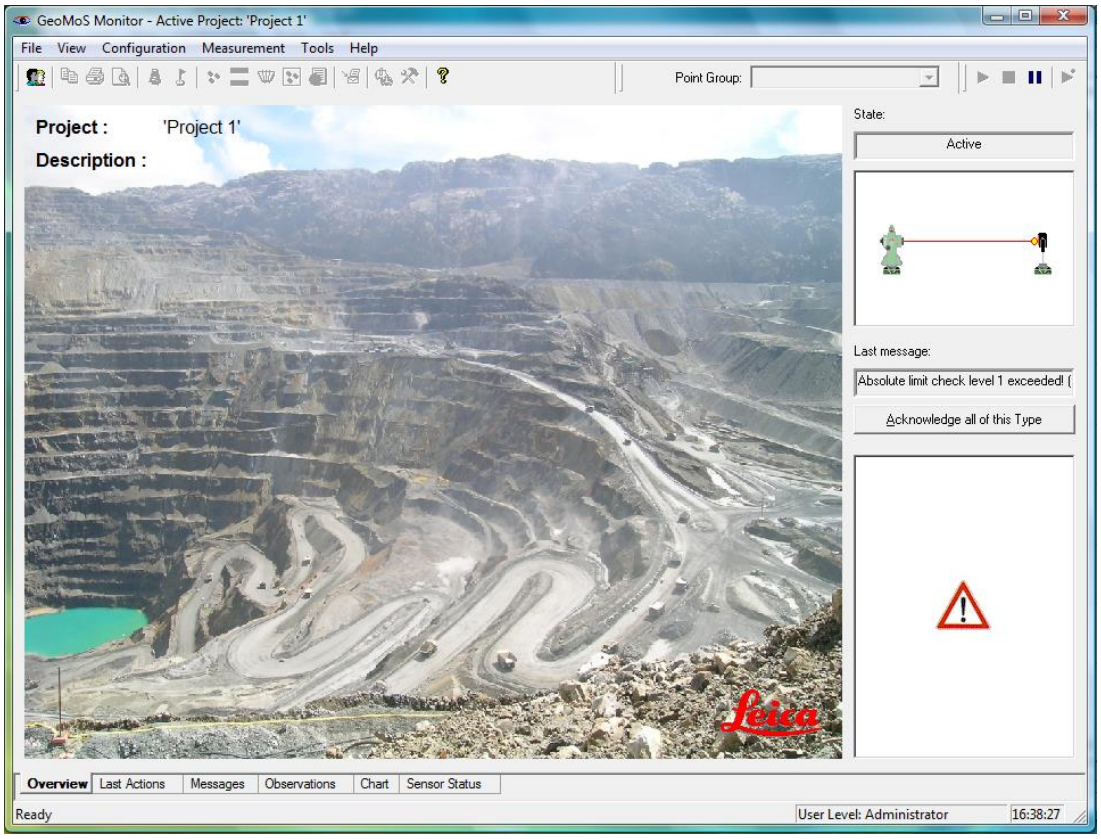

Рисунок 2. Интерфейс ПО GeoMoS модуль Monitor (http://leica-geosystems.com)

Все измерения и результаты, полученные в этом программном продукте, могут записываться в базу данных, из которой с помощью SQL-запросов можно получать необходимую информацию.

Подобным программным обеспечением является Trimble 4D Control, разработанный в США. Этот продукт также обрабатывает данные, полученные с различных геодезических оборудований. У этой программы есть три конфигурации.

1. Возможна только постобработка информации, в том числе: анализ деформаций, создание отчетов и построение графиков.
2. Возможна комбинация из данных с тахеометров, получающих программой в режиме реального времени, и спутниковых наблюдений, которые используются в постобработке и совместном уравнивании. При этом помимо функций первой конфигурации добавляется возможность оповещения при тревожных результатах.
3. Возможна комбинация данных с тахеометров и ГНСС-приемников, обрабатываемых в режиме реального времени, для дальнейшего совместного уравнивания. Кроме того, доступны все функции второй конфигурации.

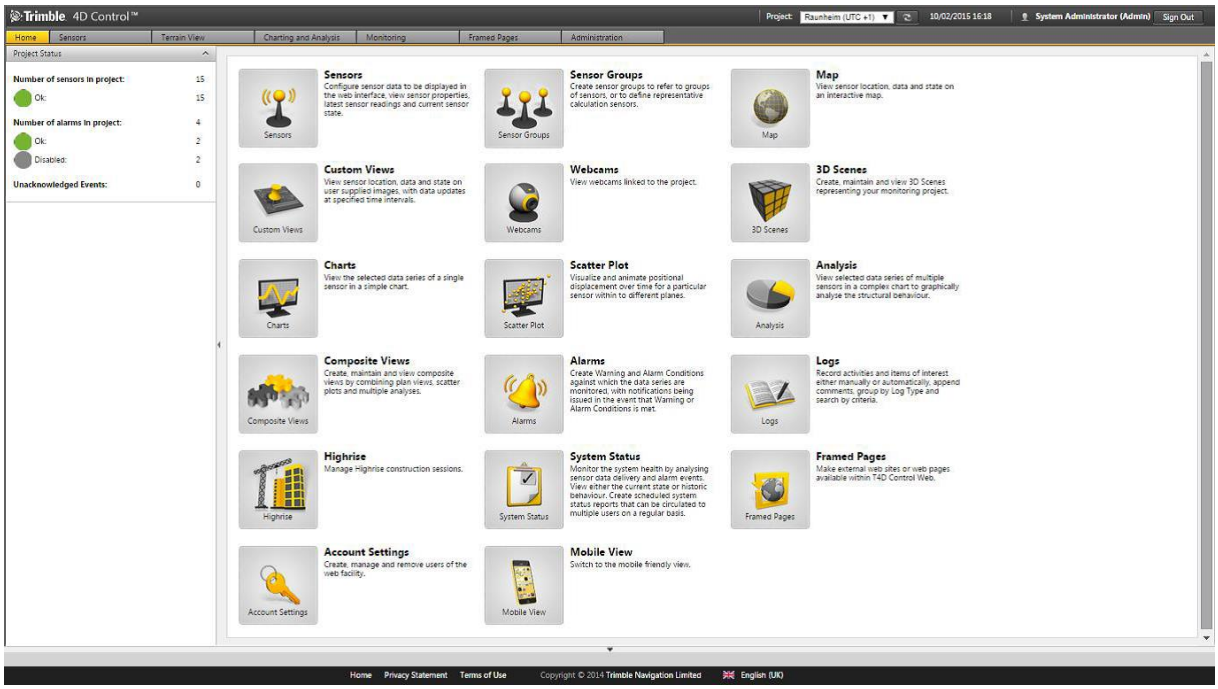

Рисунок 3. Интерфейс ПО Trimble 4D Control (http://www.trimble.com)

У компании Trimble существует еще один продукт — это Trimble Integrity Manager, использующий данные с ГНСС-приемников, а также датчиков наклона (инклинометров). Это программное обеспечение также производит анализ деформаций с дальнейшим созданием отчетов и оповещениями пользователей при тревожных ситуациях.

Таким образом, изучив несколько автоматизированных систем для мониторинга деформаций природных и инженерных объектов, можно сделать некоторые выводы по поводу необходимых функций для нового программного обеспечения.

Во-первых, проектируемый продукт должен быть мультисенсорным и обрабатывать данные из разных источников (тахеометров, ГНСС-приемников, датчиков температуры и прочих приборов) вне зависимости от фирмы производителя. Эта возможность очень развита у программного обеспечения GOCA, что дает ему большое преимущество перед продуктами компаний Leica и Trimble, у которых эта опция есть, но уже как дополнительный модуль.

Во-вторых, все существующие системы обладают функцией автоматического оповещения клиентов о серьезных деформациях. Это позволяет своевременно отреагировать на трудную ситуацию и оперативно принимать решения. Кроме того, необходимо, чтобы пользователь мог сам устанавливать допустимые значения смещений объекта.

В-третьих, немаловажным является графическое представление движения объектов во времени. Это дает возможность наглядно показать сдвиги точек, что может отразить определенные закономерности в их смещениях. Эта функция имеется у всех существующих программных продуктов.

В-четвертых, так как это программное обеспечение нужно для точной обработки и уравнивания сетей, состоящих из опорных пунктов и марок, важно, чтобы алгоритмы были оптимизированными и производили обработку данных в кратчайшие сроки. У программного обеспечения GOCA кроме уравниваний есть несколько вариантов фильтрации данных, которые облегчают работу с ГНСС-данными и выявляют общие закономерности смещений точек.

В-пятых, автоматические системы GeoMoS и Trimble 4D Control имеют базу данных, данные из которой можно использовать в других программных продуктах. Это также является неоспоримым достоинством, так как можно составлять SQL-запросы и получать информацию вне зависимости от используемой программы.

Таким образом, программные обеспечения очень схожи между собой по функциональности. Поэтому при создании нового продукта необходимо добиться наличия всех существующих опций, чтобы была возможность конкурировать с зарубежными аналогами.

Глава 2. Определения координат по фазовым измерениям.

2.1. УРАВНЕНИЯ ИЗМЕРЕННЫХ ФАЗ

Согласно теории (Коугия, 2012) фазовые определения координат основываются на измерении сдвига по фазе Ф между колебаниями принятого от спутника сигнала и собственными колебаниями частоты L_1. Эти измерения выполняют на несущей частоте L_1, а в современных двухчастотных приемниках – также на частоте L_2.

Между формированием некоторой фазы частоты L_1 на приемнике и приемом такой же фазы от спутника содержится интервал времени, который можно выразить как:

$$\Delta t = (t_{SYS} + \delta_i) - (t^{SYS} + \delta^S)$$

Через число периодов колебаний T это равенство можно опредедить так:

$$\Delta t = (N + \Phi)T$$

где N - целое число; Ф - дробь, измеренный сдвиг по фазе.

Теперь приравниваем правые части обоих выражений и умножаем их на скорость распространения сигнала c, учтя при этом, что длина волны $\lambda = cT$, тогда запишем

$$c(t_{SYS} - t^{SYS}) - c(\delta^S - \delta_i) = (N + \Phi)\lambda$$

В соответствии с выражением

$$D = c(t_{SYS} - t^{SYS}),$$

получим

$$D = (N + \Phi)\lambda + c(\delta^S - \delta_i) \qquad (2.1)$$

Так как целое число длин волн N неизвестно в измеряемом расстоянии, то результат фазовых измерений считается неоднозначным, поэтому число N называют числом неоднозначности.

Измеряемый сдвиг по фазе Ф непрерывно изменяется по мере движения спутника. При этом число N считают неизменным, поэтому все изменения относят на дробную часть Ф. Таким образом, измерив Ф, подсчитывают число ее переходов через ноль. Число переходов добавляют к дробной части Ф, в виду этого Ф становится неправильной дробью, положительной или отрицательной – в зависимости от направления переходов.

Учитывая поправку часов спутника и задержку сигнала в ионосфере и тропосфере, разделим (2.1) на λ и примем во внимание следующее: с $\lambda^{-1} = T^{-1} = f$, наблюдаются несколько спутников и в несколько эпох. Напишем уравнение для фазы сигнала, связывающей положения спутника s и пункта i в эпоху t:

$$\Phi_i^S(t) = \frac{1}{\lambda} D_i^S(t) - N_i^S + f\delta_i(t), \quad (2.2)$$

где

$$D_i^S = \sqrt{(X^S(t) - X_i)^2 + (Y^S(t) - Y_i)^2 + (Z^S(t) - Z_i)^2}, \quad (2.3)$$

$X^S(t)$, $Y^S(t)$, $Z^S(t)$ – координаты спутника *s* в эпоху *t*, X_i, Y_i, Z_i – координаты пункта *i*.

Число уравнений (2.2) равно числу измерений, т.е. n^s n_t, где n^s — число наблюдаемых спутников; n_t - число эпох с измерениями. Неизвестными здесь являются координаты X_i Y_i Z_i из уравнения (2.3), n_t смещений часов приемника и n^s чисел неоднозначности.

Длины волн несущих частот L_1 и L_2 равны соответственно 19 и 24 см. Фазы измеряют с точностью около 1% фазового цикла, что соответствует средним квадратическим погрешностям около 2 мм (Антонович, 2006).

Однако результаты измерений содержат еще значительные погрешности, причиной которых являются ошибки определения орбит спутников, ионосферная и тропосферная рефракция. Если измерения в двух пунктах А и В выполнены синхронно в эпоху t, то названные значительные погрешности в фазах $\Phi^S_A(t)$ и $\Phi^S_B(t)$ являются близкими по значению для обеих измеренных фаз. Эти общие погрешности устраняют путем формирования и использования в дальнейшем разностей фаз. В разностях фаз общие погрешности взаимно уничтожаются, остаются только малые погрешности, которые полагают случайными и статистически независимыми. В соответствии с этим для синхронно измеренных фаз $\Phi^S_A(t)$ и $\Phi^S_B(t)$ корреляционная матрица.

$$K_\Phi = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix} = \sigma^2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad (2.4)$$

где σ — среднее квадратическое отклонение.

При выполнении измерений в пунктах А и В в течение ряда эпох наблюдают 8-12 спутников и получают избыточное число измерений. Для их уравнивания выражения (2.2) линеаризуют и составляют уравнения поправок.

Уравнение поправок фазы Φ^S_A, измеренной в пункте А, с учетом разложения уравнения (2.3) в ряд

$$D_i^S = D_{i0}^S + \frac{dD_i^S}{dX_i} dX_i + \frac{dD_i^S}{dY_i} dY_i + \frac{dD_i^S}{dZ_i} dZ_i$$

и равенств

$$\frac{D_i^S}{dX} = \frac{-X^S - X_i}{D_i^S}; \frac{D_i^S}{dY} = \frac{-Y^S - Y_i}{D_i^S}; \frac{D_i^S}{dZ} = \frac{-Z^S - Z_i}{D_i^S}$$

имеет вид

$$\frac{-\Delta X_A^S}{\lambda D_A^S}\delta X_A - \frac{\Delta Y_A^S}{\lambda D_A^S}\delta Y_A - \frac{\Delta Z_A^S}{\lambda D_A^S}\delta Z_A + N_A^S + f\delta_A + \left(\frac{1}{\lambda}D_{A0}^S - \Phi_A^S\right) = \upsilon_A^S$$

где D^S_{A0} — приближенное расстояние от пункта A до спутника s.

Введем обозначения a^S_X, a^S_Y, a^S_Z для коэффициентов составленного уравнения и отметим существование следующих соотношений:

$$a_X^S = \frac{\Delta X_A^S}{\lambda D_A^S} = \frac{cos\xi^S}{\lambda}; a_Y^S = \frac{\Delta Y_A^S}{\lambda D_A^S} = \frac{cos\psi^S}{\lambda}; a_Z^S = \frac{\Delta Z_A^S}{\lambda D_A^S} = \frac{cos\zeta^S}{\lambda},$$

где ξ^S, ψ^S, ζ^S — углы, составленные направлением на спутник s и координатными осями X, Y, Z.

Заметим, что расстояния между геодезическими пунктами по сравнению с расстояниями до спутников малы, поэтому при наблюдениях на разных пунктах общего спутника s коэффициенты a^S_X, a^S_Y, a^S_Z практически не различаются.

Уравнение поправок для фазы Φ^S_A, измеренной на пункте A, принимает вид

$$-a_X^S\delta X_A - a_Y^S\delta Y_A - a_Z^S\delta Z_A + N_A^S + f\delta_A + \left(\frac{1}{\lambda}D_{A0}^S - \Phi_A^S\right) = \upsilon_A^S. \quad (2.5)$$

Аналогично для фазы Φ^S_B, измеренной на пункте B,

$$-a_X^S\delta X_B - a_Y^S\delta Y_B - a_Z^S\delta Z_B + N_B^S + f\delta_B + \left(\frac{1}{\lambda}D_{B0}^S - \Phi_B^S\right) = \upsilon_B^S. \quad (2.6)$$

2.2. МЕТОД ОТНОСИТЕЛЬНЫХ ОПРЕДЕЛЕНИЙ

Высокую точность фазовых измерений достигают за счет применения метода относительного определения положения пунктов. Результаты одновременных наблюдений одного и того же спутника в двух пунктах содержат значительные, но во многих случаях практически одинаковые погрешности. А разность результатов измерений практически свободна от таких погрешностей. Поэтому для получения высокой точности используют разности измеренных фаз. Однако при этом определяют не абсолютные координаты каждого пункта, а координаты одного пункта относительно другого или разности их координат.

2.2.1. Одинарные разности фаз

Пусть в пунктах A и B в эпоху t измерены фазовые сдвиги $\Phi^S_A(t)$ и $\Phi^S_B(t)$ сигнала спутника s (Рисунок 4). Составим для обоих измерений уравнения (2.2) и образуем их разность:

$$\Phi_B^S(t) - \Phi_A^S(t) = \frac{1}{\lambda}[D_B^S(t) - D_A^S(t)] - (N_B^S - N_A^S) + f[\delta_B(t) - \delta_A(t)]$$

Для упрощения записей обозначим каждую разность одним символом с двумя нижними индексами, например, $D^S_{AB}(t) = D^S_B(t) - D^S_A(t)$. Уравнение примет вид

$$\Phi^S_{AB}(t) = \frac{1}{\lambda} D^S_{AB}(t) - N^S_{AB} + f\delta_{AB}(t). \quad (2.7)$$

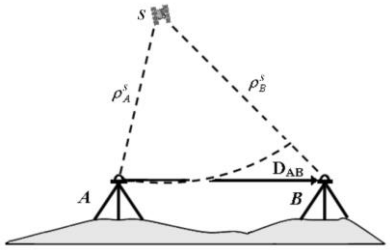

Рисунок 4. Одинарные разности (Антонович, 2006)

Если координаты пункта А считать известными, то неизвестными в уравнении (2.7) будут входящие в $D^S_{AB}(t)$ из формулы (2.3) три координаты X_B, $У_B$, Z_B пункта В, постоянное для спутника s и пунктов А и В целое число N^S_{AB} и новая для каждой эпохи разность $\delta_{AB}(t) = \delta_B(t) - \delta_A(t)$.

Уравнение поправок одинарной разности фаз $\Phi^S_{AB}(t)$ получим, вычтя из уравнения (2.5) уравнение (2.6),

$$-a^S_X\delta X_A - a^S_Y\delta Y_A - a^S_Z\delta Z_A + a^S_X\delta X_B + a^S_Y\delta Y_B + a^S_Z\delta Z_B + N^S_{AB} + f\delta_{AB} + \left(\frac{1}{\lambda}D^S_{A0} - \frac{1}{\lambda}D^S_{B0} + \Phi^S_{AB}\right) = \upsilon^S_{AB}. \quad (2.8)$$

Если считать пункт А исходным, а пункт В определяемым, то выражение (2.8) примет вид

$$a^S_X\delta X_B + a^S_Y\delta Y_B + a^S_Z\delta Z_B + N^S_{AB} + f\delta_{AB} + \left(\frac{1}{\lambda}D^S_{A0} - \frac{1}{\lambda}D^S_{B0} + \Phi^S_{AB}\right) = \upsilon^S_{AB}. \quad (2.9)$$

По результатам измерений на спутник k получим такое же уравнение:

$$a^k_X\delta X_B + a^k_Y\delta Y_B + a^k_Z\delta Z_B + N^k_{AB} + f\delta_{AB} + \left(\frac{1}{\lambda}D^k_{A0} - \frac{1}{\lambda}D^k_{B0} + \Phi^k_{AB}\right) = \upsilon^k_{AB}. \quad (2.10)$$

В уравнениях (2.9) и (2.10) свободные члены

$$l^S_{AB} = \left(\frac{D^S_{A0}}{\lambda} - \frac{D^S_{B0}}{\lambda} + \Phi^S_{AB}\right) \text{ и } l^k_{AB} = \left(\frac{D^k_{A0}}{\lambda} - \frac{D^k_{B0}}{\lambda} + \Phi^k_{AB}\right).$$

Определим корреляционную матрицу уравнений (2.9) и (2.10). Вектор ΔΦ двух одинарных разностей фаз $\Phi^S_{AB}(t)$ и $\Phi^k_{AB}(t)$ вычисляется по формуле

$$\Delta\Phi = \begin{bmatrix}\Phi^S_{AB}(t)\\ \Phi^k_{AB}(t)\end{bmatrix} = \begin{bmatrix}-1 & 1 & 0 & 0\\ 0 & 0 & -1 & 1\end{bmatrix}\begin{bmatrix}\Phi^S_A(t)\\ \Phi^S_B(t)\\ \Phi^k_A(t)\\ \Phi^k_B(t)\end{bmatrix}.$$

По формуле $K_{\Delta\Phi} = CK_\Phi C^T$, где $C = \begin{bmatrix}-1 & 1 & 0 & 0\\ 0 & 0 & -1 & 1\end{bmatrix}$ и $K_\Phi = \sigma^2 E$ (см. корреляционную матрицу (2.4)), находим

$$K_{\Delta\Phi} = \sigma^2\begin{bmatrix}2 & 0\\ 0 & 2\end{bmatrix}. \quad (2.11)$$

2.2.2. Двойные разности фаз.

Если в пунктах A и B выполнены измерения на два спутника (s и k) (Рисунок 5), то можем написать два уравнения одинарных разностей фаз вида (2.7)

$$\Phi_{AB}^{S}(t) = \frac{1}{\lambda} D_{AB}^{S}(t) - N_{AB}^{S} + f\delta_{AB}(t),$$

$$\Phi_{AB}^{k}(t) = \frac{1}{\lambda} D_{AB}^{k}(t) - N_{AB}^{k} + f\delta_{AB}(t).$$

Вычитая из второго уравнения первое и обозначая разности символами с двумя верхними индексами, запишем уравнение двойной разности фаз

$$\Phi_{AB}^{Sk}(t) = \frac{1}{\lambda} D_{AB}^{Sk}(t) - N_{AB}^{Sk}. \qquad (2.12)$$

Здесь, в частности, обозначено $D_{AB}^{Sk}(t) = D_{AB}^{k}(t) - D_{AB}^{S}(t)$.

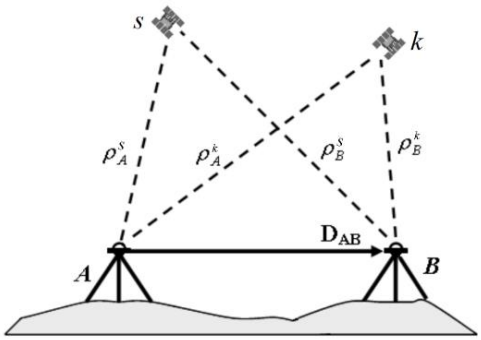

Рисунок 5. Двойные разности (Антонович, 2006)

Число неизвестных уменьшилось, потому что исчезли ошибки смещений часов приемников A и B. Полагая пункт A исходным, определяемыми неизвестными в уравнении (2.12) будем считать координаты X_B, Y_B, Z_B пункта B и число N^{Sk}_{AB}.

При числе наблюдаемых спутников n_s и числе эпох n_t, можно составить $(n_s\text{-}1)\ n_t$ независимых двойных разностей фаз и столько же уравнений (2.12). При этом число неизвестных будет равно $3 + (n_s\text{-}1)$. Чтобы число уравнений оказалось не меньше числа определяемых неизвестных, при наблюдении четырех и более спутников число эпох должно быть не меньше двух. На практике учитывают значительно большее число эпох измерений, результаты которых уравнивают.

Уравнение поправок двойной разности фаз $\Phi^{Sk}_{AB}(t)$ получим, вычтя из уравнения (2.10) уравнение (2.9),

$$a_X^{Sk}\delta X_B + a_Y^{Sk}\delta Y_B + a_Z^{Sk}\delta Z_B + N_{AB}^{Sk} + f\delta_{AB} + \left(\frac{1}{\lambda} D_{A0}^{Sk} - \frac{1}{\lambda} D_{B0}^{Sk} + \Phi_{AB}^{Sk}\right) = \upsilon_{AB}^{Sk}. \qquad (2.13)$$

Здесь, как и выше, пункт A считаем исходным, а пункт B определяемым.

Для наблюдений спутника l, принимая, как и в (2.13) спутник s за референцный, напишем такое же уравнение

$$a_X^{Sl}\delta X_B + a_Y^{Sl}\delta Y_B + a_Z^{Sl}\delta Z_B + N_{AB}^{Sl} + f\delta_{AB} + \left(\frac{1}{\lambda} D_{A0}^{Sl} - \frac{1}{\lambda} D_{B0}^{Sl} + \Phi_{AB}^{Sl}\right) = \upsilon_{AB}^{Sl}. \qquad (2.14)$$

В уравнениях (2.13) и (2.14) свободные члены

$$l_{AB}^{Sj} = \frac{1}{\lambda}(D_{A0}^{j} - D_{A0}^{S} - D_{B0}^{j} + D_{B0}^{S}) + \Phi_{AB}^{Sj},$$

где *(j = k, l).*

Определим корреляционную матрицу для уравнений (2.13) и (2.14). Вектор $\Delta_2\Phi$ двух двойных разностей фаз $\Phi^{Sk}{}_{AB}(t)$ и $\Phi^{Sl}{}_{AB}(t)$ вычисляется по формуле

$$\Delta_2\Phi = \begin{bmatrix} \Phi_{AB}^{Sk}(t) \\ \Phi_{AB}^{Sl}(t) \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Phi_{AB}^{S}(t) \\ \Phi_{AB}^{k}(t) \\ \Phi_{AB}^{l}(t) \end{bmatrix}.$$

По правилу переноса ошибок, учитывая (2.11), найдем

$$K_{\Delta_2\Phi} = \sigma^2 \begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix} \qquad (2.15)$$

Видим, что двойные разности фаз, сформированные с использованием измерений на общий спутник (спутник s), коррелированы.

2.2.3. Тройные разности фаз

Выполнив в две эпохи t_1 и t_2 измерения в пунктах A и B на два спутника s и k (Рисунок 6), получим два уравнения, аналогичных (2.12)

$$\Phi_{AB}^{Sk}(t_1) = \frac{1}{\lambda} D_{AB}^{Sk}(t_1) - N_{AB}^{Sk},$$

$$\Phi_{AB}^{Sk}(t_2) = \frac{1}{\lambda} D_{AB}^{Sk}(t_2) - N_{AB}^{Sk}.$$

Вычитая из второго уравнения первое, получим уравнение тройной разности фаз

$$\Phi_{AB}^{Sk}(t_{12}) = \frac{1}{\lambda} D_{AB}^{Sk}(t_{12}), \qquad (2.16)$$

где $\Phi_{AB}^{Sk}(t_{12}) = \Phi_{AB}^{Sk}(t_2) - \Phi_{AB}^{Sk}(t_1)$ и $D_{AB}^{Sk}(t_{12}) = D_{AB}^{Sk}(t_2) - D_{AB}^{Sk}(t_1)$.

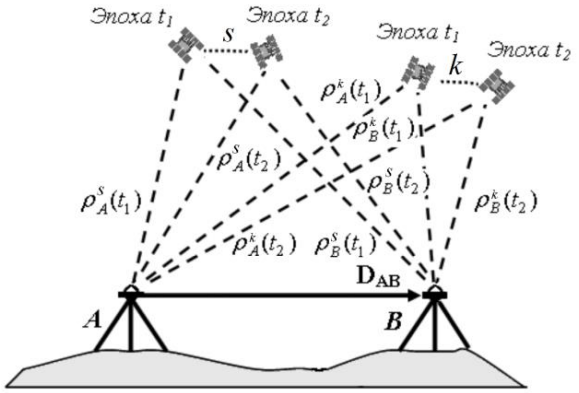

Рисунок 6. Тройные разности (Антонович, 2006)

Заметим, что в уравнении тройной разности фаз отсутствует число неоднозначности N и ошибки часов. Однако ошибки моделирования ионосферы и тропосферы остаются, как и уменьшенное влияние ошибок эфемерид (Hofmann-Wellenhof et al., 2001). Определяемыми неизвестными здесь являются только координаты X_B, Y_B, Z_B пункта B.

Напишем аналогичное (2.16) уравнение для пары спутников s и l

$$\Phi_{AB}^{Sl}(t_{12}) = \frac{1}{\lambda} D_{AB}^{Sl}(t_{12}). \qquad (2.17)$$

Уравнениям (2.16) и (2.17) соответствуют линеаризованные уравнения поправок. Примем, как и прежде, за исходный пункт А. Двойные разности будем формировать относительно спутника s. Тройные разности будем формировать относительно эпохи t_1 из одинаковых комбинаций спутников.

Для эпох t_1 и t_2 и спутников s и k уравнение поправок двойной разности (2.13) примет вид соответственно

$$a_X^{Sk}(t_1)\delta X_B + a_Y^{Sk}(t_1)\delta Y_B + a_Z^{Sk}(t_1)\delta Z_B + N_{AB}^{Sk} + l_{AB}^{Sk}(t_1) = v_{AB}^{Sk}(t_1)$$

$$a_X^{Sk}(t_2)\delta X_B + a_Y^{Sk}(t_2)\delta Y_B + a_Z^{Sk}(t_2)\delta Z_B + N_{AB}^{Sk} + l_{AB}^{Sk}(t_2) = v_{AB}^{Sk}(t_2)$$

Вычитая из второго уравнения первое, получим уравнение поправок тройной разности фаз

$$a_X^{Sk}(t_{12})\delta X_B + a_Y^{Sk}(t_{12})\delta Y_B + a_Z^{Sk}(t_{12})\delta Z_B + N_{AB}^{Sk} + l_{AB}^{Sk}(t_{12}) = v_{AB}^{Sk}(t_{12})$$

Аналогично для спутников s и l напишем

$$a_X^{Sl}(t_{12})\delta X_B + a_Y^{Sl}(t_{12})\delta Y_B + a_Z^{Sl}(t_{12})\delta Z_B + N_{AB}^{Sl} + l_{AB}^{Sl}(t_{12}) = v_{AB}^{Sl}(t_{12})$$

Определим корреляционную матрицу для последних двух уравнений. Вектор $\Delta_3\Phi$ тройных разностей фаз $\Phi^{Sk}{}_{AB}(t_{12})$ и $\Phi^{Sl}{}_{AB}(t_{12})$ вычисляется по формуле

$$\Delta_3\Phi = \begin{bmatrix} \Phi_{AB}^{Sk}(t_{12}) \\ \Phi_{AB}^{Sl}(t_{12}) \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & -1 & 1 & 0 \\ 1 & 0 & -1 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Phi_{AB}^{S}(t_1) \\ \Phi_{AB}^{k}(t_1) \\ \Phi_{AB}^{l}(t_1) \\ \Phi_{AB}^{S}(t_2) \\ \Phi_{AB}^{k}(t_2) \\ \Phi_{AB}^{l}(t_2) \end{bmatrix}$$

Пользуясь правилом переноса ошибок и учитывая (2.15), получим корреляционную матрицу для рассмотренных тройных разностей фаз $\Phi_{AB}^{Sk}(t_{12})$ и $\Phi_{AB}^{Sl}(t_{12})$:

$$K_{\Delta_3\Phi} = \sigma^2 \begin{bmatrix} 8 & 4 \\ 4 & 8 \end{bmatrix}$$

Следует обратить внимание на то, что расположенные вне главной диагонали элементы последней матрицы могут иметь различные четные значения от -4 до +4 в зависимости от сочетания символов s, k, t_1, t_2 и s, l, t_1, t_2 Модуль элемента равен 2 (q - 1), где q - число одинаковых символов. Знак элемента изменяется на обратный, если какой-либо символ изменяет позицию.

2.2.4. Решение уравнений, составленных по результатам фазовых измерений

Предварительное решение заключается в следующих этапах.

- Систему уравнений, составленных по результатам измерений на базовой линии AB, решают методом наименьших квадратов. Координаты пункта A считают известными. Решая уравнения поправок тройных разностей, находят координаты X_B, Y_B, Z_B пункта B. Решая уравнения поправок двойных разностей, определяют координаты X_B, Y_B, Z_B пункта B и числа N. При верном составлении корреляционных матриц оба решения приладят к одинаковым значениям координат X_B, Y_B, Z_B. Полученное решение является предварительным, так как полученные числа N оказываются нецелыми, что некорректно.
- Отметим, что уравнения одинарных разностей для получения предварительного решения не подходят, так как матрица коэффициентов таких уравнений имеет линейно-зависимые столбцы (столбцы с коэффициентами λ и c).
- Рассмотрим пример матрицы коэффициентов A, вектора определяемых параметров $\bar{X}$, вектора свободных членов L и корреляционной матрицы системы уравнений двойных разностей фаз, составленной по результатам измерений на пунктах A и B. Наблюдения четырех спутников (s, k, l, m) выполнены в две эпохи (t_1 и t_2). Получены следующие величины:

$$A = \begin{bmatrix} a_X^{Sk}(t_1) & a_Y^{Sk}(t_1) & a_Z^{Sk}(t_1) & 1 & 0 & 0 \\ a_X^{Sl}(t_1) & a_Y^{Sl}(t_1) & a_Z^{Sl}(t_1) & 0 & 1 & 0 \\ a_X^{Sm}(t_1) & a_Y^{Sm}(t_1) & a_Z^{Sm}(t_1) & 0 & 0 & 1 \\ a_X^{Sk}(t_2) & a_Y^{Sk}(t_2) & a_Z^{Sk}(t_2) & 1 & 0 & 0 \\ a_X^{Sl}(t_2) & a_Y^{Sl}(t_2) & a_Z^{Sl}(t_2) & 0 & 1 & 0 \\ a_X^{Sm}(t_2) & a_Y^{Sm}(t_2) & a_Z^{Sm}(t_2) & 0 & 0 & 1 \end{bmatrix}; \quad \acute{X} = \begin{bmatrix} \delta X_B \\ \delta Y_B \\ \delta Z_B \\ N_{AB}^{Sk} \\ N_{AB}^{Sl} \\ N_{AB}^{Sm} \end{bmatrix}$$

$$L = \begin{bmatrix} l_{AB}^{Sk}(t_1) \\ l_{AB}^{Sl}(t_1) \\ l_{AB}^{Sm}(t_1) \\ l_{AB}^{Sk}(t_2) \\ l_{AB}^{Sl}(t_2) \\ l_{AB}^{Sm}(t_2) \end{bmatrix} \quad K = 2\sigma^2\lambda^2 = \begin{bmatrix} 2 & 1 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 \\ 1 & 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 1 \\ 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 1 & 2 \end{bmatrix}$$

- Решение системы уравнений выполняют методом наименьших квадратов (Губанов, 1997):

$$P = \mu^2 K^{-1};\ R = A^T A;\ \acute{X} = -R^{-1}A^T PL;\ K_{\acute{X}} = \mu^2 R^{-1}, \qquad (2.18)$$

где P — весовая матрица, μ — априорно принятая средняя квадратическая ошибка единицы веса; R — матрица коэффициентов нормальных уравнений; $\bar{X}$ — вектор определяемых параметров, $K\bar{X}$ — корреляционная матрица вектора параметров.

- Используя вычисленные поправки δX_B, δY_B, δZ_B, исправляют приближенные координаты пункта В и находят предварительные, или, как их часто называют, «плавающие», координаты и числа N.

Разрешение неоднозначностей совершается следующим образом:

- Перебирая целые числа, близкие к предварительным значениям чисел N, составляют различные их комбинации и подставляют в уравнения (2.13), (2.14). Получают столько вариантов систем уравнений, сколько составлено комбинаций чисел N. Каждую систему решают методом наименьших квадратов по формулам (2.18), вычисляя вектор поправок к предварительным координатам $[\delta X_B, \delta Y_B, \delta Z_B]$.
- Из возможных векторов поправок выбирают тот, который получен из лучшей комбинации чисел N. Лучшей комбинацией считают ту, в которой минимальной окажется значение целевой функции $V^T PV$.

Существуют и иные методы разрешения неоднозначностей.

Решение, полученное после разрешения неоднозначностей, в отличие от предварительного, «плавающего» решения, называют «фиксированным».

Глава 3. Определение координат спутника по бортовым эфемеридам

По бортовым эфемеридам вычисляются координаты спутника на определенный момент наблюдения. К необходимым эфемеридам, содержащимся в навигационном RINEX-файле, относятся:

- t_{oe} — эпоха;
- WN — неделя;
- e — эксцентриситет орбиты;
- $A^{1/2}$ — квадратный корень из большой полуоси орбиты;
- Ω_0 — долгота восходящего узла плоскости орбиты на время эпохи;
- i_0 — наклонение орбиты на время эпохи;
- ω — аргумент перигея;
- M_0 — средняя аномалия на начало эпохи;
- Δn — отклонение значения среднего движения;
- $\acute{\Omega}$ — скорость изменения долготы восходящего узла плоскости орбиты;
- IDOT — скорость изменения наклонения орбиты;
- C_{uc} — амплитуда квадратурной поправки аргумента широты;
- C_{us} — амплитуда синфазной поправки аргумента широты;
- C_{rc} — амплитуда квадратурной поправки радиуса орбиты;
- C_{rs} — амплитуда синфазной поправки радиуса орбиты;
- C_{ic} — амплитуда квадратурной поправки наклонения орбиты;
- C_{is} — амплитуда синфазной поправки наклонения орбиты.

Координаты рассчитываются на определенный момент наблюдения (t_{obs}), на который мы знаем псевдодальность до спутника (P) по следующему алгоритму (Витязев, Гусева, 2011).

Первым этапом является вычисление среднего движения

$$n_0 = \sqrt{\frac{\mu}{A^3}}, \tag{3.1}$$

где μ — геоцентрическая гравитационная постоянная, равная 3,986004418 * 10^{14} $м^3/с^2$.

Затем определяем момент излучения сигнала спутником

$$t_{em} = t_{obs} - \frac{P}{c}, \tag{3.2}$$

где c — скорость света, равная 299792458,0 м/с.

Разница между моментом излучения сигнала спутника и эпохой определяет время от начала эпохи, значение которого по модулю не должно превышать 302400.

$$t = t_{em} - t_{oe}. \quad (3.3)$$

Таким образом, если значение больше 302400 с, то и него вычитается 604800, а если меньше, то добавляется.

По данным из бортовых эфемерид вычисляем исправленное среднее движение

$$n = n_0 + \Delta n. \quad (3.4)$$

Необходимое в расчетах значение средней аномалии находится по формуле

$$M = M_0 + nt. \quad (3.5)$$

Ее значение нужно для расчета эксцентрической аномалии, производимое по уравнению Кеплера итерационным методом.

$$\begin{aligned} M &= E - esinE, \\ E_i &= M + esinE_{i-1}. \end{aligned} \quad (3.6)$$

При вычислении выполняется такое количество итераций, которое дает разницу между E_i и E_{i-1} меньше, чем точность данных числового формата float (а именно 0,000001). Вычисленная таким образом эксцентрическая аномалия используется в расчетах истиной аномалии (3.7), которая необходима для определения аргумента широты (3.8).

$$tg\frac{\nu}{2} = \sqrt{\frac{1+e}{1-e}}tg\frac{E}{2}, \quad (3.7)$$

$$\phi = \nu + \omega. \quad (3.8)$$

Для дальнейших вычислений необходимо исправить значения аргумента широты, радиуса и наклонения

$$u = \phi + \delta u, \quad (3.9)$$

$$r = a\,(1 - e\,cosE) + \delta r, \quad (3.10)$$

$$i = i_0 + \delta i + IDOT\,t, \quad (3.11)$$

где $\delta u = C_{us}\,sin2\,\phi + C_{uc}\,cos2\phi,$

$\delta r = C_{rs}\,sin2\,\phi + C_{rc}\,cos2\,\phi,$

$\delta i = C_{is}\,sin2\,\phi + C_{ic}\,cos2\,\phi.$

В результате всех этих вычислений можно получить координаты спутника в орбитальной плоскости по формулам

$$X_{orb} = r\,cos\,u, \quad (3.12)$$

$$Y_{orb} = r\,sin\,u. \quad (3.13)$$

Однако, в описываемом в главе 1 алгоритме используются координаты в земной системе координат. Для этого необходимо вычислить исправленную долготу восходящего узла

$$\Omega = \Omega_0 + (\acute{\Omega} - \acute{\Omega}_e)t - \acute{\Omega}_e t_{oe}, \quad (3.14)$$

где $\acute{\Omega}_e$ — угловая скорость вращения Земли, равная 7,2921151467 * 10^{-5} рад/с.

Таким образом, определяем положение спутника в земной системе координат

$$X = X_{orb} \cos \Omega - Y_{orb} \cos i \sin \Omega, \quad (3.15)$$

$$Y = X_{orb} \sin \Omega - Y_{orb} \cos i \cos \Omega, \quad (3.16)$$

$$Z = Y_{orb} \sin i. \quad (3.17)$$

В результате с помощью бортовых эфемерид спутника мы можем определить его координаты, используемые в дальнейшем для расчета предварительной разности координат спутника и места в алгоритме относительных фазовых измерений.

Глава 4. Программная реализация алгоритмов определения координат относительным фазовым методом

Для создания программы, выполняющей алгоритмы определения координат, был выбран язык программирования Python. Выбор был обусловлен несколькими факторами:

- язык прост и дружелюбен в обучении, что упростило процесс подготовки к разработке программы;
- язык позволяет программировать сложные операции простыми языковыми конструкциями: блоки кода разделяются величиной отступа, упор в синтаксисе идет на слова, а не символы;
- язык поддерживает объектно-ориентированное программирование;
- существует большое количество библиотек, которые упрощают написание собственного кода;
- по сравнению со строго типизированными или компилирующими языками Python повышает производительность написания кода за счет меньшего его объема.

Реализация алгоритмов заключалась в создании модулей, выполняющих отдельные этапы всего алгоритма. Таким образом были созданы следующие модули:

- по чтению RINEX-файлов версии 3.0 (файлы данных наблюдений и навигационные файлы);
- по вычислению координат спутника по бортовым эфемеридам;
- по вычислению координат относительным фазовым методом;
- для загрузки входных RINEX-файлов и вывода результата.

4.1. РЕАЛИЗАЦИЯ АЛГОРИТМА ДЛЯ ЧТЕНИЯ RINEX-ФАЙЛОВ ВЕРСИИ 3.0

Первый модуль создавался на основе спецификации RINEX 3.0, в которой описаны все форматы и размеры данных, содержащихся в файле (Рисунок 7). Тем самым написание этого модуля упрощалось, так как формат записи данных в файле строго регламентирован.

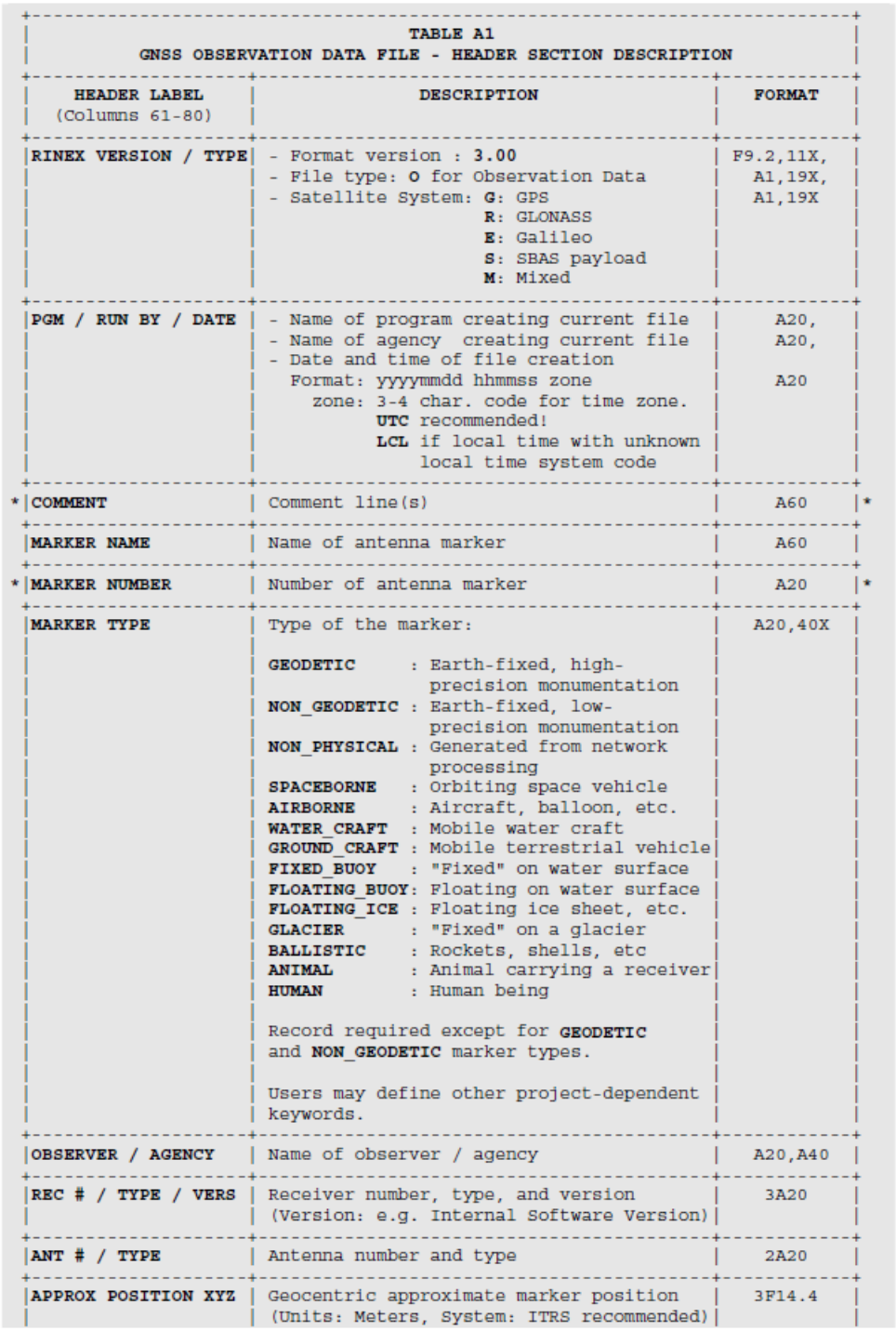

TABLE A1
GNSS OBSERVATION DATA FILE - HEADER SECTION DESCRIPTION

HEADER LABEL (Columns 61-80)	DESCRIPTION	FORMAT
RINEX VERSION / TYPE	- Format version : 3.00 - File type: O for Observation Data - Satellite System: G: GPS R: GLONASS E: Galileo S: SBAS payload M: Mixed	F9.2,11X, A1,19X, A1,19X
PGM / RUN BY / DATE	- Name of program creating current file - Name of agency creating current file - Date and time of file creation Format: yyyymmdd hhmmss zone zone: 3-4 char. code for time zone. UTC recommended! LCL if local time with unknown local time system code	A20, A20, A20
* COMMENT	Comment line(s)	A60 *
MARKER NAME	Name of antenna marker	A60
* MARKER NUMBER	Number of antenna marker	A20 *
MARKER TYPE	Type of the marker: GEODETIC : Earth-fixed, high-precision monumentation NON_GEODETIC : Earth-fixed, low-precision monumentation NON_PHYSICAL : Generated from network processing SPACEBORNE : Orbiting space vehicle AIRBORNE : Aircraft, balloon, etc. WATER_CRAFT : Mobile water craft GROUND_CRAFT : Mobile terrestrial vehicle FIXED_BUOY : "Fixed" on water surface FLOATING_BUOY: Floating on water surface FLOATING_ICE : Floating ice sheet, etc. GLACIER : "Fixed" on a glacier BALLISTIC : Rockets, shells, etc ANIMAL : Animal carrying a receiver HUMAN : Human being Record required except for GEODETIC and NON_GEODETIC marker types. Users may define other project-dependent keywords.	A20,40X
OBSERVER / AGENCY	Name of observer / agency	A20,A40
REC # / TYPE / VERS	Receiver number, type, and version (Version: e.g. Internal Software Version)	3A20
ANT # / TYPE	Antenna number and type	2A20
APPROX POSITION XYZ	Geocentric approximate marker position (Units: Meters, System: ITRS recommended)	3F14.4

Рисунок 7. Страница из спецификации

Модуль состоит из описания четырех классов и функций для них:

1. NavInfoHeader — в нем будет записана информация заголовка навигационного RINEX-файла. Объекты этого класса имеют следующие переменные:

   version – версия формата

   satSystem – система спутников («G» - GPS, «R» - ГЛОНАСС, «M» - Mixed)

   mixed – смешанная система спутников – GPS и ГЛОНАСС

program – название программы (генератора данного файла)

company – имя организации (автора данного файла)

date – время создания файла

gpsIonAlpha – ионосферные параметры альманаха (A0-A3) для GPS

gpsIonBeta – ионосферные параметры альманаха (B0-B3) для GPS

galIonAlpha – ионосферные параметры альманаха(A0-A2) для Galileo

timeCorr – параметры альманаха для пересчета времени в шкалу UTC

leapSec – разница между временем UTC и временем GPS

lineCount – количество строк, занятых заголовком навигационного файла

2. NavInfo — содержит сами данные навигационного файла. Объекты этого класса имеют всего две переменные (Рисунок 2):

   header — переменная класса NavInfoHeader (информация заголовка)

   data — бортовые эфемериды спутников

3. ObservationHeader — класс, описывающий заголовок файла наблюдений. Объекты этого классасодержат переменные:

   version – версия формата

   satSystem – система спутников («G» - GPS, «R» - ГЛОНАСС, «M» - Mixed)

   mixed – смешанная система спутников – GPS и ГЛОНАСС

   program – название программы (генератора данного файла)

   company – имя организации (автора данного файла)

   date – время создания файла

   markerName – имя маркера

   markerNumber – номер маркера

   markerType – тип маркера

   observer – имя наблюдателя

   agency – имя организации

   receiver – параметры приемника

   antenna – параметры антенны

   approxPos – приближенные координаты в системе координат WG84

   centerOfMass – текущее положение центра масс (координаты X,Y,Z)

   satSystemParams – число типов наблюдений, сами типы наблюдений

signalStrenght – сила сигнала

interval – интервал между наблюдениями в секундах

timeFirst – время первой записи наблюдений в файл

timeLast – время последней записи наблюдений в файл

clockOffs – смещение времени в приемнике

leapSec – число добавленных секунд с 6 января 1980 г.

lineCount - количество строк, занятых заголовком навигационного файла

4. Observation — в нем содержатся данные наблюдений. Две переменные описывают класс (Рисунок 8):

   header — переменная класса ObservationHeader (информация заголовка)

   data — наблюдения и мощность сигнала

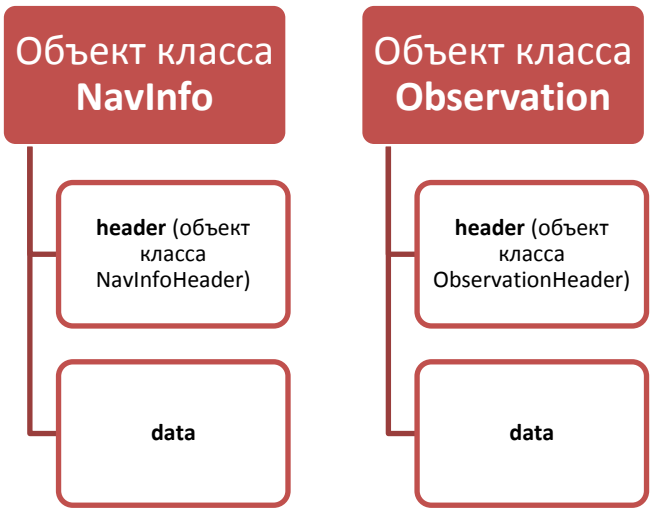

Рисунок 8. Структура объектов классов NavInfo и Observation

4.2. РЕАЛИЗАЦИЯ АЛГОРИТМА ДЛЯ ВЫЧИСЛЕНИЯ КООРДИНАТ СПУТНИКА ПО БОРТОВЫМ ЭФЕМЕРИДАМ

Второй модуль (sat.py) был создан на основе алгоритма, описанного во второй главе. Используя первый модуль производится чтение бортовых эфемерид, по которым производится вычисление.

Расчет производится функцией calculate (Рисунок 9) этого модуля, которой передаются значения момента времени, псевдодальность и спутник. Результатом функции являются координаты спутника X, Y, Z.

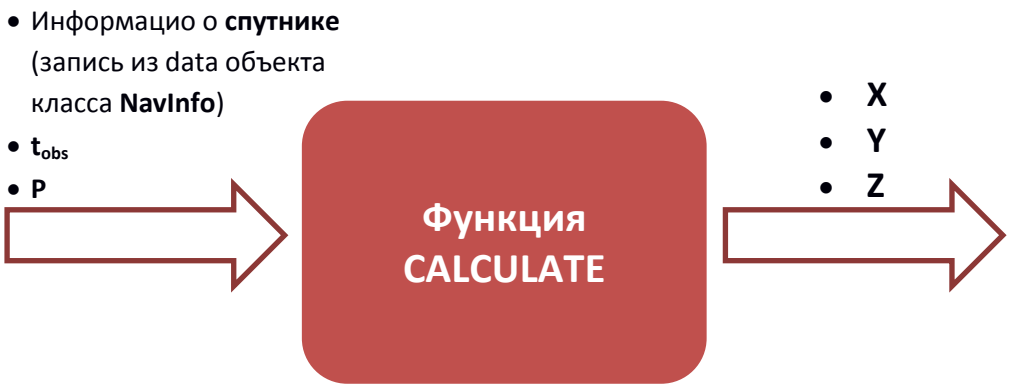

Рисунок 9. Схема работы функции calculate модуля sat.py

4.3. РЕАЛИЗАЦИЯ АЛГОРИТМА ДЛЯ ВЫЧИСЛЕНИЯ КООРДИНАТ МЕСТНОСТИ

Третий модуль (phase.py) является главной частью работы, именно он рассчитывапет координаты относительным фазовым методом.

Функция compute (Рисунок 10) производит вычисления поправок в координаты определяемого пункта (по двойным и тройным разностям), а также определяет число неоднозначностей. Затем перебирая целые числа неоднозначночтей находит «фиксированное» решение (то, у которого минимальны поправки).

Функция generateNeighbourhood генерирует 3 целых значения, ближайших ко входному числу. Эта функция необходима для перебора ближайших целых чисел неоднозначности.

Функция getCommonSatellites возвращает общие спутники для двух эпох из двух RINEX-файлов наблюдений.

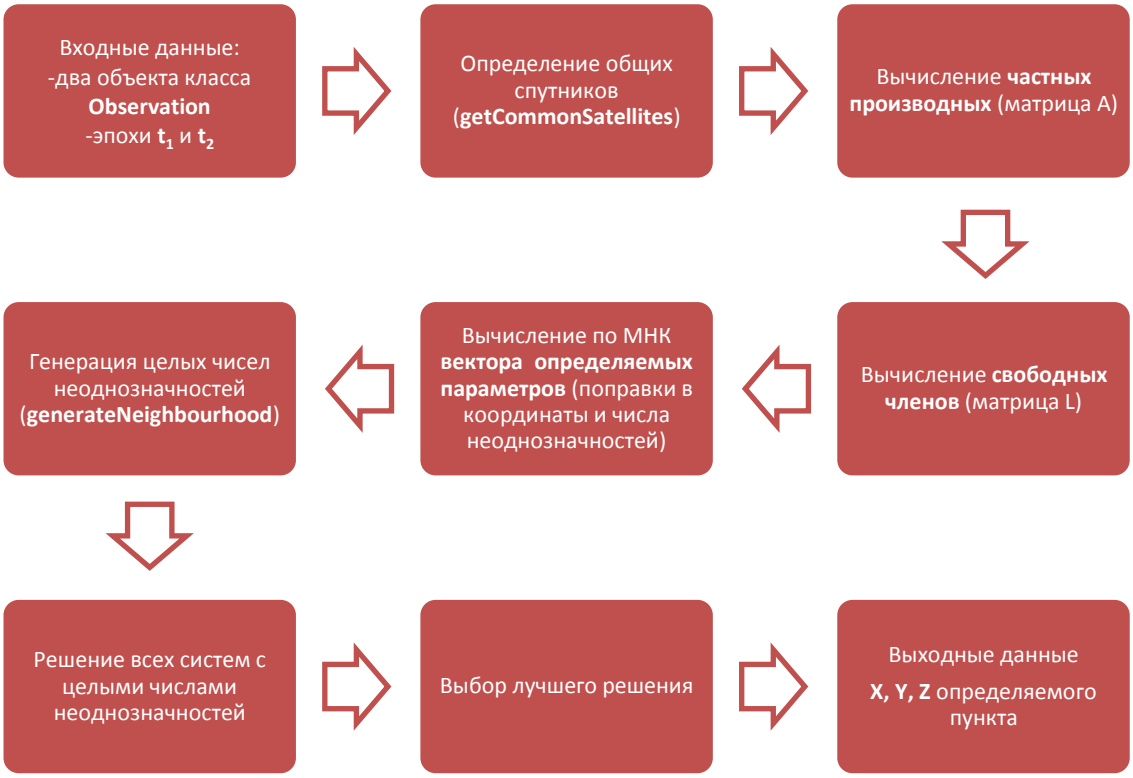

Рисунок 10. Схема работы функции compute модуля phase.py

Глава 5. Тестирование созданного программного обеспечения.

В процессе разработки продукта, проводилось модульное тестирование программных модулей. Данная процедура подразумевает проверку корректности выполнения каждой функции по отдельности на предмет соответствия заданным требованиям (функции модулей rinex.py и phase.py). Также проверялась правильность работы модуля sat.py на модельных данных с заранее известным результатом вычислений (Витязев, Гусева, 2011). Результаты, полученные модулем, отличаются от рассчитанных авторами не более чем 0.05 м.

Однако основное внимание уделялось тестированию всего приложения с целью удовлетворения выполнению задачи в целом. В качестве модельных данных здесь были использованы спутниковые наблюдения на референцных станциях на Васильевском острове, наб. реки Фонтанка и в Петергофе (Рисунок 11).

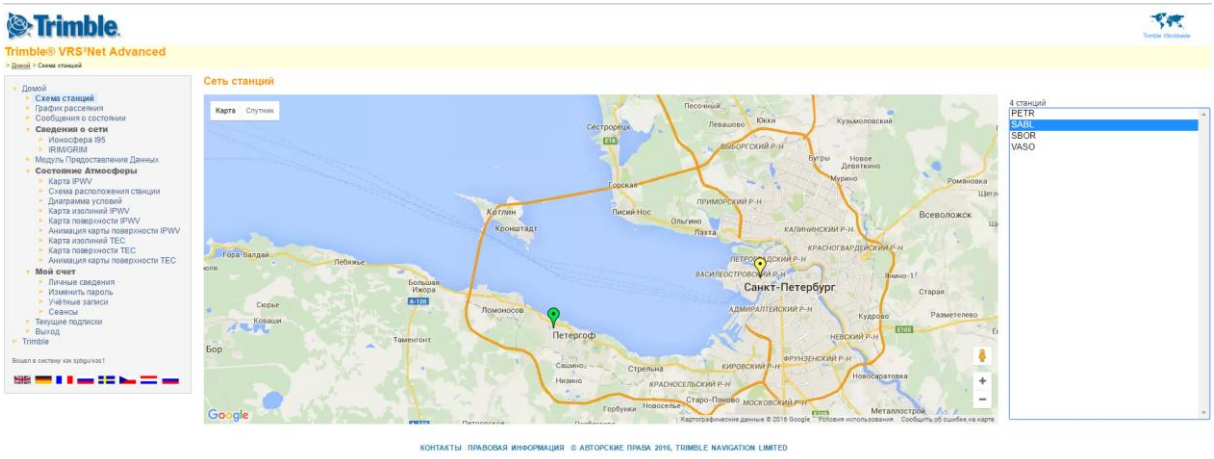

Рисунок 11. Сеть станций

Была проведена обработка по двадцать пар эпох по всем спутникам для двух векторов: между «Василеостровской» станцией и «Фонтанкой» и между «Василеостровской» и «Петергофом».

По первому вектору ошибки по оси X колеблются от -0.33 до +0.06 метров. Среднее значение $\bar{\Delta} = -0.15$. Ошибки по оси Y колеблются от -0.23 до +0.08 метров. Среднее значение $\bar{\Delta} = -0.09$ (Рисунок 12).

По второму вектору получены менее точные результаты. Ошибки по оси X колеблются от -0.30 до +0.67 метров. Среднее значение $\bar{\Delta} = -0.39$. Ошибки по оси Y колеблются от

-0.47 до +0.16 метров. Среднее значение $\bar{\Delta} = -0.21$ (Рисунок 13).

Как видно, точность вычисления координат порядка десятков сантиметров, что не достаточно для высокоточных видов геодезических работ. При том во втором случае

результаты хуже, чем в первом. Одной из причин данных расхождений является тот факт, что расстояние между пунктами измеряется в десятках километров, в связи с чем атмосферные условия над пунктами будут различными. Но при проведении расчетов данные атмосферные условия не учитываются, так как при вычислении координат относительным фазовым методом полагают, что пункты находятся в равных условиях.

Кроме того, на точность определений влияет геометрическое положение спутников. При малых углах возвышения спутника над горизонтом задержки в распространении сигнала в атмосфере плохо прогнозируются (Антонович, 2006).

Таким образом, результаты показали, что алгоритм работает в целом верно, но для повышения точности необходимо предварительно анализировать используемые наблюдения от спутников.

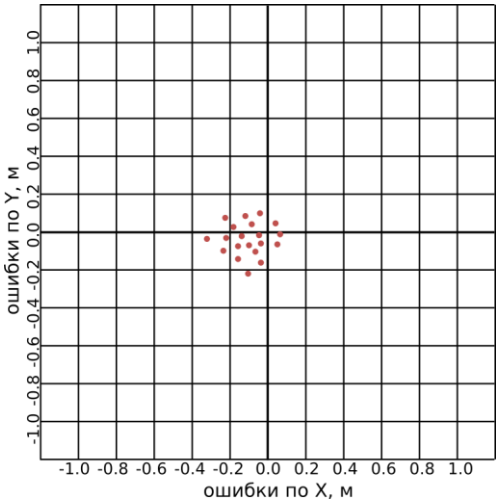

Рисунок 12. Результаты по вектору между «Василеостровской» станцией и «Фонтанкой»

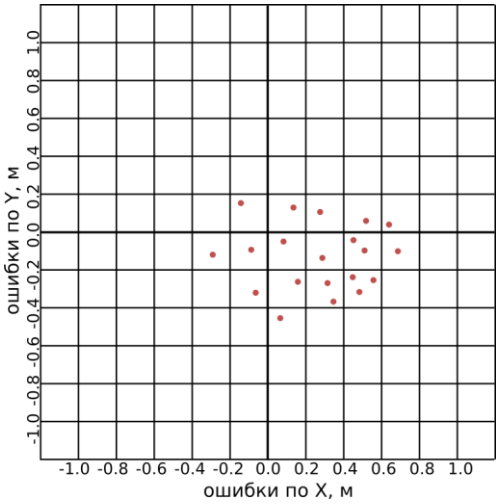

Рисунок 13. Результаты по вектору между «Василеостровской» станцией и «Петергофом»

Заключение.

В ходе выполнения данной работы изучена имеющаяся литература по обработке спутниковых геодезических измерений. В частности, рассмотрены относительные фазовые методы определений координат. Особое внимание уделено одинарным, двойным и тройным разностям, определению целого числа неоднозначности и вычислению «фиксированного» решения (координаты определяемого пункта). В работе приведены математические выводы уравнений, которые используются в дальнейшем в разработанных программных модулях.

Реализация рассмотренных подходов осуществлена на языке программирования Python, выбор которого обусловлен гибкостью и простотой разработки. Созданные модули предназначены для выполнения следующих функций:

1) чтение RINEX-файлов версии 3.0;
2) вычисление координат спутника по бортовым эфемеридам;
3) определение координат местности относительным фазовым методом.

В работе приведено полное описание разработанных модулей, а также блок-схемы реализованных алгоритмов.

Дальнейшее направление развитие работы может вестись в двух направлениях. С теоретической точки зрения следует рассмотреть рекуррентный алгоритм вычисления координат по всем эпохам, а не только по двум, а также методы для повышения точности обработки. С практической точки зрения необходимо разработать алгоритмы анализа наблюдений, отбрасывания измерений от спутников, углы возвышения которых недостаточно большие.

Реализованные алгоритмы и программные модули могут быть использованы при разработке программы обработки спутниковых измерений для использования в научных и образовательных целях. Здесь нужно будет уделить внимание быстродействию рассмотренных подходов.

Список литературы.

1. Антонович К.М. Использование спутниковых радионавигационных систем в геодезии: в 2 т. – М.: ФГУП «Картгеоцентр», 2006. – 360 с.
2. ГОСТ Р 53608 – 2009. Глобальная навигационная спутниковая система. Методы и технологии выполнения геодезических и землеустроительных работ. Разрешение неоднозначности фазовых измерений псевдодальности. Основные положения. – М.: Стандартинформ, 2010. – 10 с.
3. Губанов В.С. Обобщенный метод наименьших квадратов. Теория и применение в астрометрии. – СПб.: Наука, 1997. – 318 с.
4. Коугия В.А., Павлов В.И. Современные проблемы уравнивания инженерно-геодезических сетей. – Спб.: Национальный минерально-сырьевой университет «Горный», 2012. – 105 с.
5. Маркузе Ю.И. Теория математической обработки геодезических измерений: в 2 ч. - Ч. 2. Основы метода наименьших квадратов и уравнительных вычислений. – М.: МИИГАиК, 2005. – 280 с.
6. МДС 13-23.2009 Рекомендации по проведению динамического мониторинга высотных зданий и сооружений с использованием навигационного поля глобальных навигационных спутниковых систем (ГНСС). – М.: ОАО «ЦПП», 2010. – 28 с.
7. Небесные и земные координаты: Учеб. пособие. Под редакцией Витязев В.В., Гусева И.С., Князев В.И., Мищенко М.П., Петров С.Д., Титов О.А., Цветков А.С. – Спб.: Изд-во С.-Петерб. ун-та, 2011. – 314 с.
8. Серапинас Б.Б. Введение в ГЛОНАСС и GPS измерения: Учеб. пособие – Ижевск: Удм. гос. ун-т, 1999. – 93 с.
9. Спутниковые навигационные системы: Учеб. пособие. – М.: МАИ каф. 604, 2004. – 336 с.
10. Тяпкин В.Н., Гарин Е.Н. Методы определения навигационных параметров подвижных средств с использованием спутниковой радионавигационной системы ГЛОНАСС. – Красноярск: Сиб. федер. ун-т, 2012. – 260 с.
11. Hofmann-Wellenhof, B. Global Positioning System. Theory and practice. – Fifth, revised edition / B. Hofmann-Wellenhof, H. Lichtenegger and J. Collins – Wienn, New-York: Springer. – 2001. – P. 384.
12. GOCA - Основанная на GNSS/LPS/LS онлайн-система контроля и оповещения. http://goca.info/index_ru.html

13. Leica GeoMoS Monitoring Solution. http://leica-geosystems.com/products/total-stations/software/leica-geomos
14. Trimble 4D Control. http://www.trimble.com/Infrastructure/Trimble-4D-Control.aspx
15. Trimble Integrity Manager App. http://www.trimble.com/Infrastructure/Trimble-Integrity-Manager-App.aspx?tab=Processing_Engines

Приложение А. Модуль rinex.py

```
from datetime import datetime
from warnings import warn

SAT_SYSTEM = {"G": "GPS", "R": "GLONASS", "M": "Mixed"}


def safeFloat(s):
    try:
        s = s.replace('D', 'E').replace('d', 'e')
        ret = float(s)
    except ValueError:
        ret = 0.0

    return ret


def safeInt(s):
    try:
        ret = int(s)
    except ValueError:
        ret = 0

    return ret


class NavInfoHeader(object):
    def __init__(self, fname = None):
        self.version = (3, 0)
        self.satSystem = "Mixed"
        self.mixed = True
        self.program = ""
        self.company = ""
        self.date = datetime.now()
        self.gpsIonAlpha = None
        self.gpsIonBeta = None
        self.galIonAlpha = None
        self.timeCorr = None
        self.leapSec = 0
        self.lineCount = 0

        if fname != None:
            self.read(fname)
        return


    def read(self, fname):
        input = open(fname, 'r')

        lineCount = 0
        for line in input:
            lineCount += 1
            lineType = line[60:80]
            if lineType == "RINEX VERSION / TYPE":
                self.version = (int(line[:6]), int(line[7:9]))

                if line[20] != "N":
                    raise RuntimeError("Wrong file type: " + line[20])

                if self.version[0] == 3:
                    if line[40] in SAT_SYSTEM:
                        self.satSystem = SAT_SYSTEM[line[40]]
```

```
                else:
                    raise RuntimeError("Wrong satellite system: " + line[40])

                self.mixed = (line[40] == "M")

            else:
                self.satSystem = SAT_SYSTEM["G"] # HACK
                self.mixed = False

        if lineType == "PGM / RUN BY / DATE ":
            self.program = line[:20]
            self.company = line[20:40]

            if line[56:59] != "UTC":
                raise RuntimeError("Wrong time zone: " + line[56:60])

            self.date = datetime(
                int(line[40:44]),
                int(line[44:46]),
                int(line[46:48]),
                int(line[49:51]),
                int(line[51:53]),
                int(line[53:55]))

        if lineType == "COMMENT             ":
            continue

        if lineType == "IONOSPHERIC CORR    ":
            coeffs = [float(line[5 + i*12 : 17 + i*12]) for i in xrange(4)]
            if line[:4] == "GPSA":
                self.gpsIonAlpha = coeffs
            elif line[:4] == "GPSB":
                self.gpsIonBeta = coeffs
            elif line[:4] == "GAL ":
                self.galIonAlpha = coeffs[:3]
            else:
                raise RuntimeError("Wrong correction type: " + line[:4])

        if lineType == "TIME SYSTEM CORR    ":
            self.timeCorr = {
                "a0": float(line[5:22]),
                "a1": float(line[22:38]),
                "T": int(line[38:45]),
                "W": int(line[45:50]),
                "S": line[51:56],
                "U": safeInt(line[57:59])}

        if lineType == "LEAP SECONDS        ":
            self.leapSec = int(line[:6])

        if lineType == "END OF HEADER       ":
            self.lineCount = lineCount
            return

    raise RuntimeError("Incomplete header")
    return


class NavInfo(object):
    def __init__(self, fname = None):
        self.header = NavInfoHeader()
```

```
    self.data = {}

    if fname != None:
        self.read(fname)
    return


def read(self, fname):
    self.header.read(fname)
    self.data = {}

    input = open(fname, 'r')

    for i in xrange(self.header.lineCount):
        input.readline() # skip line

    if self.header.version[0] == 3:
        while True:
            line = input.readline()
            if not line:
                break

            if line[0] not in SAT_SYSTEM or line[0] == "M":
                raise RuntimeError("Wrong satellite system: " + line[0])

            satSystem = SAT_SYSTEM[line[0]]
            if (not self.header.mixed) and satSystem != self.header.satSystem:
                warn("Wrong satellite system: " + line[0], RuntimeWarning)

            satName = line[:3]
            msgDatetime = datetime(
                int(line[4:8]),
                int(line[9:11]),
                int(line[12:14]),
                int(line[15:17]),
                int(line[18:20]),
                int(line[21:23]))

            data = [float(line[23 + 19*i:42 + 19*i]) for i in xrange(3)]

            for i in xrange(7):
                line = input.readline()
                if not line:
                    raise RuntimeError("Unexpected end of file")

                data.extend((float(line[4 + 19*i:23 + 19*i]) for i in xrange(4)))

            if satName not in self.data:
                self.data[satName] = {}

            self.data[satName][msgDatetime] = data

    else:
        while True:
            line = input.readline()
            if not line:
                break

            satSystem = SAT_SYSTEM["G"] # HACK

            satName = "G" + line[:2]
            msgDatetime = datetime(
```

```
                2000 + int(line[3:5]),
                int(line[6:8]),
                int(line[9:11]),
                int(line[12:14]),
                int(line[15:17]),
                int(float(line[17:22])))

            data = [safeFloat(line[22 + 19*i:41 + 19*i]) for i in xrange(3)]

            for i in xrange(7):
                line = input.readline()
                if not line:
                    raise RuntimeError("Unexpected end of file")

                data.extend((safeFloat(line[3 + 19*i:22 + 19*i]) for i in xrange(4)))

            if satName not in self.data:
                self.data[satName] = {}

            self.data[satName][msgDatetime] = data

        return


    def __getitem__(self, index):
        if type(index) is tuple:
            satName, t = index
        else:
            satName = index
            t = None

        if satName not in self.data:
            return None

        bestData = None
        bestT = None

        for ti in sorted(self.data[satName]):
            if t is None:
                if bestT is None:
                    bestData = self.data[satName][ti]
                    bestT = ti
            else:
                # TODO: implement method to determine bestT
                raise NotImplementedError("Unimplemented logic")

        return bestData


class ObservationHeader(object):
    def __init__(self, fname = None):
        self.version = (3, 0)
        self.satSystem = "Mixed"
        self.mixed = True
        self.program = ""
        self.company = ""
        self.date = datetime.now()
        self.markerName = ""
        self.markerNumber = ""
        self.markerType = "GEODETIC".ljust(20)
        self.observer = ""
        self.agency = ""
```

```
    self.reciever = {}
    self.antenna = {}
    self.approxPos = [0, 0, 0]
    self.centerOfMass = [0, 0, 0]
    self.satSystemParams = {}
    self.signalStrenght = ""
    self.interval = ""
    self.timeFirst = datetime.now()
    self.timeLast = datetime.now()
    self.clockOffs = 0

    self.leapSec = 0
    self.lineCount = 0

    if fname != None:
       self.read(fname)
    return


def read(self, fname):
    input = open(fname, 'r')

    lastSatSystem = None

    lineCount = 0
    for line in input:
       line = line.strip('\n').ljust(80)
       lineCount += 1
       lineType = line[60:80]
       if lineType == "RINEX VERSION / TYPE":
          self.version = (int(line[:6]), int(line[7:9]))

          if line[20] != "O":
             raise RuntimeError("Wrong file type: " + line[20])

          if line[40] in SAT_SYSTEM:
             self.type = SAT_SYSTEM[line[40]]
          else:
             raise RuntimeError("Wrong satellite system: " + line[40])

       if lineType == "PGM / RUN BY / DATE ":
          self.program = line[:20]
          self.company = line[20:40]

          if self.version[0] == 3:
             if line[56:59] != "UTC":
                raise RuntimeError("Wrong time zone: " + line[56:60])

             self.date = datetime(
                int(line[40:44]),
                int(line[44:46]),
                int(line[46:48]),
                int(line[49:51]),
                int(line[51:53]),
                int(line[53:55]))
          else:
             self.date = line[40:60]

       if lineType == "COMMENT             ":
          continue

       if lineType == "MARKER NAME         ":
```

```
    self.markerName = line[:60]

if lineType == "MARKER NUMBER       ":
    self.markerNumber = line[:20]

if lineType == "OBSERVER / AGENCY   ":
    self.observer = line[:20]
    self.agency = line[20:60]

if lineType == "REC # / TYPE / VERS ":
    rec = {}
    rec["number"], rec["type"], rec["version"] = \
        [(line[20*i:20 + 20*i]) for i in xrange(3)]
    self.reciever = rec

if lineType == "ANT # / TYPE        ":
    ant = self.antenna
    ant["number"], ant["type"] = \
        [(line[20*i:20 + 20*i]) for i in xrange(2)]

if lineType == "ANTENNA: DELTA H/E/N":
    ant = self.antenna
    ant["height"], ant["eccEast"], ant["eccNorth"] = \
        [float(line[14*i:14 + 14*i]) for i in xrange(3)]

if lineType == "APPROX POSITION XYZ ":
    self.approxPos = [float(line[14*i:14 + 14*i]) for i in xrange(3)]

if lineType == "CENTER OF MASS: XYZ ":
    self.centerOfMass = [float(line[14*i:14 + 14*i]) for i in xrange(3)]

if lineType == "SIGNAL STRENGTH UNIT":
    self.signalStrenght = float(line[:20])

if lineType == "SYS / # / OBS TYPES ":
    if self.version[0] != 3:
        raise RuntimeError("Wrong file version: " + str(self.version))

    if line[0] == ' ':
        if line[:6] != ' ' * 6:
            raise RuntimeError("Wrong file format")
    else:
        ss = line[0]
        if not self.mixed:
            if not(ss in SAT_SYSTEM) or SAT_SYSTEM[ss] != self.satSystem:
                raise RuntimeError("Wrong satellite system: " + ss)

        if not(ss in self.satSystemParams):
            self.satSystemParams[ss] = {}

        self.satSystemParams[ss]["numObsrv"] = int(line[3:6])
        self.satSystemParams[ss]["obsrvTypes"] = []
        lastSatSystem = ss

    ssParams = self.satSystemParams[lastSatSystem]
    count = min(13, ssParams["numObsrv"] - len(ssParams["obsrvTypes"]))
    for i in xrange(count):
        ssParams["obsrvTypes"].append(line[7 + i * 4 : 10 + i * 4])

if lineType == "# / TYPES OF OBSERV ":
    if self.version[0] != 2:
        raise RuntimeError("Wrong file version: " + str(self.version))
```

```
            if line[:6] != ' ' * 6:
               for ss in "GR":
                  if not(ss in self.satSystemParams):
                     self.satSystemParams[ss] = {}

                  self.satSystemParams[ss]["numObsrv"] = int(line[:6])
                  self.satSystemParams[ss]["obsrvTypes"] = []

               for ss in "GR":
                  ssParams = self.satSystemParams[ss]
                  count = min(9, ssParams["numObsrv"] - len(ssParams["obsrvTypes"]))
                  for i in xrange(count):
                     ssParams["obsrvTypes"].append(line[10 + i * 6 : 12 + i * 6] + "C") # HACK: for same L1C C1C
codes as in RINEX 3.00

            if lineType == "INTERVAL            ":
               self.interval = float(line[:10])

            if lineType == "TIME OF FIRST OBS   ":
               if line[48:51] != "GPS":
                  raise RuntimeError("Wrong time zone: " + line[48:51])

               seconds = safeFloat(line[30:43])
               microSeconds = int((seconds - int(seconds)) * 1e6)
               seconds = int(seconds)

               self.timeFirst = datetime(
                     int(line[:6]),
                     int(line[6:12]),
                     int(line[12:18]),
                     int(line[18:24]),
                     int(line[24:30]),
                     seconds,
                     microSeconds)

            if lineType == "TIME OF LAST OBS    ":
               if line[48:51] != "GPS":
                  raise RuntimeError("Wrong time zone: " + line[48:51])

               seconds = safeFloat(line[30:43])
               microSeconds = int((seconds - int(seconds)) * 1e6)
               seconds = int(seconds)

               self.timeLast = datetime(
                     int(line[:6]),
                     int(line[6:12]),
                     int(line[12:18]),
                     int(line[18:24]),
                     int(line[24:30]),
                     seconds,
                     microSeconds)

            if lineType == "RCV CLOCK OFFS APPL ":
               self.clockOffs = int(line[:6])

            if lineType == "LEAP SECONDS        ":
               self.leapSec = int(line[:6])

            if lineType == "END OF HEADER       ":
               self.lineCount = lineCount
```

```
            if not self.satSystemParams:
                raise RuntimeError("No observation types were specified")

            for satSystem in self.satSystemParams:
                params = self.satSystemParams[satSystem]
                if len(params["obsrvTypes"]) != params["numObsrv"]:
                    raise RuntimeError("Not enough observation types were specified")

            return

        raise RuntimeError("Incomplete header")
        return


class Observation(object):
    def __init__(self, fname = None):
        self.header = ObservationHeader()

        self.data = {}

        if fname != None:
            self.read(fname)
        return


    def read(self, fname):
        self.header.read(fname)
        self.data = {}

        input = open(fname, 'r')

        for i in xrange(self.header.lineCount):
            input.readline() # skip line

        if self.header.version[0] == 3:
            while True:
                line = input.readline()
                if not line:
                    break

                if line[0] != ">":
                    raise RuntimeError("Wrong format of line: " + line[:10])

                msg = {}

                seconds = safeFloat(line[19:29])
                seconds = int(seconds)

                epoch = datetime(
                    int(line[2:6]),
                    int(line[7:9]),
                    int(line[10:12]),
                    int(line[13:15]),
                    int(line[16:18]),
                    seconds)

                msg["epochFlag"] = int(line[31])
                msg["recClockOffset"] = float(line[41:56])

                count = int(line[32:35])
                data = {}
                for i in xrange(count):
```

```
            line = input.readline()
            if not line:
                raise RuntimeError("Not enough lines in epoch")

            satSystem = line[0]
            if satSystem not in self.header.satSystemParams:
                RuntimeError("Wrong satellite system: " + line[0])

            params = self.header.satSystemParams[satSystem]
            numObsrv = params["numObsrv"]
            obsrvTypes = params["obsrvTypes"]

            sat = {}
            for j in xrange(numObsrv):
                sat[obsrvTypes[j]] = safeFloat(line[3 + 16*j:17 + 16*j])

            data[line[:3]] = sat

        msg["data"] = data
        self.data[epoch] = msg

else:
    while True:
        line = input.readline()
        if not line:
            break

        line = line.strip('\n').ljust(80)

        msg = {}

        seconds = safeFloat(line[15:26])
        seconds = int(seconds)

        epoch = datetime(
            2000 + int(line[1:3]),
            int(line[4:6]),
            int(line[7:9]),
            int(line[11:12]),
            int(line[13:15]),
            seconds)

        msg["epochFlag"] = int(line[28])
        msg["recClockOffset"] = safeFloat(line[68:80])

        count = int(line[29:32])
        sats = [line[32 + i*3 : 35 + i*3] for i in xrange(12)]
        sats = sats[:count]

        while len(sats) < count:
            line = input.readline()
            if not line:
                raise RuntimeError("Not enough lines in epoch")

            line = line.strip('\n').ljust(80)
            sats.extend([line[32 + i*3 : 35 + i*3] for i in xrange(12)])
            sats = sats[:count]

        data = {}
        for i in xrange(count):
            line = input.readline()
            if not line:
```

```
                raise RuntimeError("Not enough lines in epoch")

            line = line.strip('\n').ljust(80)

            satSystem = sats[i][0]
            if satSystem not in self.header.satSystemParams:
                RuntimeError("Wrong satellite system: " + sats[i][0])

            params = self.header.satSystemParams[satSystem]
            numObsrv = params["numObsrv"]
            obsrvTypes = params["obsrvTypes"]

            sat = {}
            for k in xrange((numObsrv + 4) / 5):
                if k > 0:
                    line = input.readline()
                    if not line:
                        raise RuntimeError("Not enough lines in epoch")

                    line = line.strip('\n').ljust(80)

                for j in xrange(5):
                    jk = k * 5 + j
                    if jk >= numObsrv:
                        break
                    sat[obsrvTypes[jk]] = \
                        safeFloat(line[16 * j : 14 + 16 * j])

            data[sats[i]] = sat

        msg["data"] = data
        self.data[epoch] = msg

    return


  def __getitem__(self, index):
    if type(index) is tuple:
        epoch, sat = index
    else:
        epoch = index
        sat = None

    if sat is None:
        return self.data[epoch]

    return self.data[epoch]["data"][sat]


if __name__ == "__main__":
  import main
  main.main()
```

Приложение В. Модуль sat.py.

```
import rinex
from datetime import datetime, timedelta
from math import *
import os
from warnings import warn
```

```
def sampleCalculate():
    tobs = datetime(2006,07,10,11,53,45)
    P = 21267472.262

    sat = [0] * 31
    sat[4] = -135.6875
    sat[5] = 0.00000000383337396113
    sat[6] = -1.78927512368
    sat[7] = -7.13393092155e-6
    sat[8] = 0.00963819015305
    sat[9] = 9.02265310287e-6
    sat[10] = 5153.62374496
    sat[11] = 129600
    sat[12] = -1.86264514923e-7
    sat[13] = 2.73843067938
    sat[14] = -5.21540641785e-8
    sat[15] = 0.974034050279
    sat[16] = 214.21875
    sat[17] = 2.66223304346
    sat[18] = -7.68603443990e-9
    sat[19] = 1.67864135072e-11

    return calculate(tobs, P, sat)


def calculate(tobs, P, sat):
    if sat is None:
        raise RuntimeError("No nav info is provided")

    mu = 398600441800000
    O_e = 7.2921151467e-5
    c = 299792458

    n0 = sqrt(mu) / pow(sat[10], 3)

    w = tobs.weekday()
    tobs_ = ((w + 1) % 7 )* 86400 + 3600 * tobs.hour + 60 * tobs.minute + tobs.second
    tem = tobs_ - P / c

    t = tem-sat[11]
    if t > 302400:
        t = t - 604800
    if t < -302400:
        t = t + 604800

    n = n0 + sat[5]

    M = sat[6] + n * t

    E = M # E0 = M; E(i + 1) = M + e * sin(Ei)
    while True:
        delta = M + sat[8] * sin(E) - E
        if abs(delta) < 1e-10:
            break

        E += delta

    v = (atan(sqrt((1 + sat[8]) / (1 - sat[8]))*tan(E / 2))) * 2

    phi = v + sat[17]

    deltaU = sat[9] * sin(2*phi) + sat[7] * cos(2*phi)
```

```
    deltaR = sat[4] * sin(2*phi) + sat[16] * cos(2*phi)
    deltaI = sat[14] * sin(2*phi) + sat[12] * cos(2*phi)

    u = phi + deltaU
    r = pow(sat[10],2) * (1 - sat[8] * cos(E)) + deltaR
    i = sat[15] + deltaI + sat[19] * t

    Xorb = r * cos(u)
    Yorb = r * sin(u)

    O = sat[13] + (sat[18] - O_e) * t - O_e * sat[11]
    X = Xorb * cos(O) - Yorb * cos(i) * sin(O)
    Y = Xorb * sin(O) + Yorb * cos(i) * cos(O)
    Z = Yorb * sin(i)

    return X, Y, Z


def printFirstXYZ(navInfo, observation, satName):
    if satName[0] != "G":
        warn("Unsupported satellite type: " + satName[0], RuntimeWarning)

    for epochTime in observation.data:
        epoch = observation.data[epochTime]
        if satName not in epoch["data"]:
            continue

        print calculate(
            epochTime,
            epoch["data"][satName]["C1C"],
            navInfo[satName])

        return

    raise RuntimeError("No epoch have " + satName)

def main():
    os.chdir(os.path.dirname(os.path.abspath(__file__)))
    fname = "Rinex/VASO338R"
    try:
        navInfo = rinex.NavInfo(fname + ".14n")
        observ = rinex.Observation(fname + ".14o")
    except RuntimeError as e:
        print "Something went wrong:", e

    for epochTime in observ.data:
        print epochTime
        epoch = observ.data[epochTime]
        for sat in sorted(epoch["data"]):
            if sat[0] != "G":
                warn("Unsupported satellite type: " + sat[0], RuntimeWarning)
                continue

            print sat, calculate(
                epochTime,
                epoch["data"][sat]["C1C"],
                navInfo[sat])
        print
        print

    print "G02"
    printFirstXYZ(navInfo, observ, "G02")
```

```
    print sampleCalculate()


if __name__ == "__main__":
    main()
```

Приложение С. Модуль phase.py.

```
import rinex
from datetime import datetime, timedelta
from sat import calculate, printFirstXYZ
import numpy as np
from numpy.linalg import norm as length
import os
from warnings import warn

def generateNeighbourhood(v, prevIterNeighbourhood):
    if not prevIterNeighbourhood:
        return prevIterNeighbourhood

    n = len(prevIterNeighbourhood[0])
    values = [round(v[n])]
    values = [values[0] - 1] + values + [values[-1] + 1]
    outputNeighbourhood = []
    for pt in prevIterNeighbourhood:
        pt = list(pt)
        for x in values:
            outputNeighbourhood.append(pt + [x])

    if n == len(v) - 1:
        return outputNeighbourhood

    return generateNeighbourhood(v, outputNeighbourhood)


def getCommonSatellites(obsA_t1, obsA_t2, obsB_t1, obsB_t2, satSystem = "G"):
    satsA_t1 = obsA_t1["data"].keys()
    satsA_t2 = obsA_t2["data"].keys()
    satsB_t1 = obsB_t1["data"].keys()
    satsB_t2 = obsB_t2["data"].keys()

    sats = []
    for sat in sorted(satsA_t1):
        if satSystem in "GR":
            if sat[0] != satSystem:
                continue

        if sat not in satsA_t2:
            continue
        if sat not in satsB_t1:
            continue
        if sat not in satsB_t2:
            continue

        sats.append(sat)

    return sats

def compute(obsA, navA, posA, obsB, navB, posB, t1, t2):
    sats = getCommonSatellites(obsA[t1], obsA[t2], obsB[t1], obsB[t2])
```

```
if len(sats) < 4:
   print "Not enough common satellites for A and B at", t1, "and", t2
   exit(1)

M = len(sats)#min(len(sats), 4)

deltas = np.zeros((2, 2, M, 3))
phases = np.zeros((2, 2, M))
for i, (obs, nav, pos) in enumerate(((obsA, navA, posA), (obsB, navB, posB))):
   for j, tj in enumerate((t1, t2)):
      for k, sk in enumerate(sats[:M]):
         calcPos = calculate(tj, obs[tj, sk]["C1C"], nav[sk])
         deltas[i, j, k] = pos - calcPos
         phases[i, j, k] = obs[tj, sk]["L1C"]

dists = length(deltas, axis = 3)

waveLength = 0.190293672798364880 47631742646405
#0.190293672798364880 47631742646405
A = deltas[1] / (waveLength * dists[1].reshape(2, M, 1))
A_ = (A[1, 1:] - A[1, :1]) - (A[0, 1:] - A[0, :1])

Phi = phases[1] - phases[0]
Phi = Phi[1] - Phi[0]
Phi = Phi[1:] - Phi[:1]

Phi_ = (dists[1] - dists[0]) / waveLength
Phi_ = Phi_[1] - Phi_[0]
Phi_ = Phi_[1:] - Phi_[:1]

L = (Phi - Phi_).reshape(-1, 1)
b_ = L
K = np.ones((M - 1, M - 1)) * 4
np.fill_diagonal(K, 8)
K *= 4e-6
P = np.linalg.inv(K)
R = A_.T. dot(P).dot(A_)
X = -np.linalg.inv(R).dot(A_.T).dot(P).dot(b_)

A_ = (A[:, 1:] - A[:, :1]).reshape(-1, 3)
E = np.concatenate((np.eye(M - 1), np.eye(M - 1)), axis = 0)
A_ = np.concatenate((A_, E), axis = 1)

Phi = phases[1] - phases[0]
Phi = (Phi[:, 1:] - Phi[:, :1]).reshape(-1)

Phi_ = (dists[1] - dists[0]) / waveLength
Phi_ = (Phi_[:, 1:] - Phi_[:, :1]).reshape(-1)

L = (Phi - Phi_).reshape(-1, 1)
b_ = L
K = np.ones(((M - 1) * 2, (M - 1) * 2)) * 2
K[: M - 1, M - 1 :] = 0
K[M - 1 :, : M - 1] = 0
np.fill_diagonal(K, 4)
K *= 4e-6
P = np.linalg.inv(K)
R = A_.T.dot(P).dot(A_)
X_ = -np.linalg.inv(R).dot(A_.T).dot(P).dot(b_)

intNeighbourhood = np.array(generateNeighbourhood(X_[3:], [[]]))
b_ = L + E.dot(intNeighbourhood.T)
```

```
    A_ = A_[:,:3]
    R = A_.T.dot(P).dot(A_)
    X_fixed = -np.linalg.inv(R).dot(A_.T).dot(P).dot(b_)

    V = A_.dot(X_fixed) + b_
    err = np.sum(V * P.dot(V), axis = 0)
    best = np.argmin(err)
    X_fixed = X_fixed[:, best].reshape(-1, 1)
    N_fixed = intNeighbourhood[best]

    print X
    print X_[:3]
    print X_fixed
    print N_fixed
    print X_[3:,0]

    return posB - X_fixed.reshape(posA.shape)


if __name__ == "__main__":
    import main
    main.main()
```

Приложение D. Модуль main.py.

```
import rinex
import os
import sys
import phase
import numpy as np

def getCommonEpochs(obsA, obsB):
    epochs = []
    for epoch in sorted(obsA.data):
        if epoch not in obsB.data:
            continue

        epochs.append(epoch)

    return epochs

def main():
    usage = "Usage: main.py <input_A.??o> <input_B.??o>"

    ########## DEBUG ######################
    if len(sys.argv) == 1:
        print "Debug"
        sys.argv.append("Rinex/lgu/0560176k.13O")
        sys.argv.append("Rinex/kol/0546176k.13O")
    ########################################

    if len(sys.argv) != 3:
        print usage
        exit(1)
    try:
        fnameA = sys.argv[1]
        print "Read A from", fnameA
        obsA = rinex.Observation(fnameA)
        fnameA = fnameA[:-1] + 'n'
        if not os.path.exists(fnameA):
            fnameA = fnameA[:-1] + 'N'
        navA = rinex.NavInfo(fnameA)
```

```
        fnameB = sys.argv[2]
        print "Read B from", fnameB
        obsB = rinex.Observation(fnameB)
        fnameB = fnameB[:-1] + 'n'
        if not os.path.exists(fnameB):
            fnameB = fnameB[:-1] + 'N'
        navB = rinex.NavInfo(fnameB)

        epochs = getCommonEpochs(obsA, obsB)

        if len(epochs) < 2:
            print "Not enough common epochs in A and B"
            exit(1)

        posA = np.array(obsA.header.approxPos)
        posB = np.array(obsB.header.approxPos)
        phase.compute(obsA, navA, posA, obsB, navB, posB, epochs[17], epochs[18])

    except RuntimeError as e:
        print "Something went wrong:", e
        print usage

    return


if __name__ == "__main__":
    main()
```