ФГБОУ ВПО «Национальный исследовательский университет «МЭИ»

Институт радиотехники и электроники

Кафедра радиотехнических систем

КУРСОВОЙ ПРОЕКТ

«АВТОМАТИЧЕСКИЕ ИСПЫТАНИЯ НАП»

Выполнил:
Днепров В. В.
гр. ЭР-18-08

Руководитель проекта:
к.т.н. Корогодин И. В.

Москва
2013г.

Содержание

Задание на курсовой проект ... 3

1. Введение ... 3

Цель работы ... 3

Решаемые задачи ... 3

Используемые методы и подходы ... 4

2. Разработка программного обеспечения проведения автоматических испытаний НАП ... 4

Классы ... 6

Класс CFSV ... 6

Класс CSMBV ... 7

Класс CRSC ... 8

Класс CReceiver ... 8

3. Проведение автоматических испытаний с помощью разработанного инструментария ... 9

Апробация методики ... 11

Заключение ... 14

Использованные источники ... 14

Приложение ... 15

Задание на курсовой проект

Организовать автоматические испытания НАП. Для этого создать библиотеку функций, позволяющую автоматизировать процесс испытаний НАП. Провести автоматические испытания НАП для проверки созданной библиотеки.

1. Введение

В настоящее время кафедра радиотехнических систем имеет хорошую материально - техническую базу для проведения экспериментов различного рода: лабораторных работ, научно - исследовательских работ, создания новой и тестирования уже имеющейся аппаратуры. В процессе проведения таких экспериментов возникает потребность фиксировать показания приборов, записывать результаты измерений, отсчитывать время, менять параметры эксперимента, повторять один и тот же эксперимент много раз. Описанные действия могут негативно влиять на конечный результат эксперимента - вносить ошибки. Проведение же одного и того же эксперимента много раз практически невозможно осуществить без ошибок экспериментатора. В такой ситуации может помочь автоматизация процесса проведения эксперимента.

Цель работы

Создать простой и удобный инструмент для проведения автоматических испытаний НАП.

Решаемые задачи

- Используя программный пакет MATLAB, создать библиотеки функций для управления различными приборами и навигационными модулями
- Реализовать сценарий эксперимента для демонстрации возможностей созданных библиотек

Используемые методы и подходы

Для проведения автоматических испытаний НАП необходимо организовать удаленное (с ПК пользователя) управление рядом используемых при испытаниях приборов. Как правило, в испытаниях НАП используются: имитатор сигналов СРНС, анализатор спектра, аттенюатор. Также нужно реализовать обработку информации, поступающей от НАП, контролировать его состояние. На кафедре радиотехнических систем имеется ряд приборов фирмы Rohde & Schwarz: генератор сигналов SMBV100A, анализатор спектра FSV, аттенюатор RSC. Все приборы имеют LAN интерфейс, что позволяет организовать удаленное управление ими через локальную сеть по протоколу TCP/IP. Программный пакет MATLAB также имеет несколько встроенных функций для работы с этим протоколом. Также программный пакет MATLAB является удобной средой для обработки полученных результатов испытаний. Для управления приборами были созданы классы MATLAB, методы которых позволяют проводить автоматические испытания.

2. Разработка программного обеспечения проведения автоматических испытаний НАП

Классы для управления приборами объединены в проект **ArcticSEA**. **ArcticSEA** (*System for Experiment Automatization*) - библиотека функций в виде совокупности matlab-скриптов, предназначенных для проведения автоматизированных экспериментов и испытаний навигационной аппаратуры с помощью лабораторного оборудования.

Цель проекта

Приборы Rohde & Schwarz позволяют достаточно просто управлять собой через локальную сеть. При этом некоторую сложность представляет огромное число специфичных команд управления, отвечающих стандарту SCPI (Standard Commands for Programmable Instruments). Цель создания данной библиотеки - написать классы - фасады, методы которых позволят выполнять необходимые действия с приборами без явного использования команд SCPI.

Состав библиотеки

На данный момент в состав библиотеки входят 3 класса для управления приборами и класс для общения с навигационным модулем.

Общие методы для всех классов управления приборами

Общие методы классов управления приборами		
Метод	**Входные аргументы**	**Возвращает**
SetConnection(IP, port) Установка соединения с прибором	IP, string - адрес устройства в сети port - порт, для приборов Rohde & Schwarz port = 5025	1, если соединение установлено успешно, иначе 0
CloseConnection Закрытие соединения с прибором	---	1, если соединение закрыто успешно, иначе 0
SendCommand(strCommand) Отправка команды в формате SCPI (т.е. как в документации на прибор)	strCommand, string - команда в формате SCPI	1, если команда отправлена успешно, иначе 0. Если отправленная команда не может быть выполнена прибором (например, прибор не поддерживает заданную частоту), сообщение от прибора будет выведено в командную строку.
SendQuery(strCommand) Отправка запроса в формате SCPI	strCommand, string - запрос в формате SCPI, заканчивается вопросительным знаком	[Status, Result] Status = 1, если запрос послан успешно, иначе 0. Result, string - ответ прибора на посланный запрос.
GetIDN Запрос информации о модели прибора, серийном номере	---	[Status, IDN] Status = 1, если операция успешна, иначе 0. IDN, string - информация о модели прибора, серийном номере.
Preset Установка настроек прибора "по умолчанию", очистка лога ошибок	---	1, если операция успешна, иначе 0
QueryError Запрос ошибок	---	1, если есть ошибки, сообщение прибора будет выведено в командную строку. 0 - ошибок нет.

Классы

Класс CFSV

Описание

Класс, созданный для управления анализатором спектра FSV. На данный момент реализованы следующие возможности:

1. Установка центральной частоты в режиме анализа спектра
2. Установка полосы в режиме анализа спектра
3. Измерение мощности сигнала в заданной полосе

Методы

Методы		
Метод	**Входные аргументы**	**Возвращает**
SetCenterFreq(Freq) Установка центральной частоты в режиме анализа спектра	Freq - центральная частота, может быть как string 10GHz, так и число 10E9	1, если операция успешна, иначе 0
SetSpan(Span) Установка полосы в режиме анализа спектра	Span - полоса, может быть как string 10MHz, так и число 10E6	1, если операция успешна, иначе 0
PowerMeasure(Bandwidth) Измерение мощности сигнала в заданной полосе	Bandwidth - полоса измерения, может быть как string 10MHz, так и число 10E6	[Status, measure] Status = 1, если операция успешна, иначе 0. measure - измеренная мощность, дБм.

Класс CSMBV

Описание

Класс, созданный для управления генератором сигналов SMBV. На данный момент реализованы следующие возможности:

1. Установка частоты сигнала
2. Установка мощности сигнала
3. Имитация сигналов заданного числа спутников(GPS, диапазон L1, расположение - Москва, статика, мощность всех сигналов одинакова)

Методы

Методы		
Метод	**Входные аргументы**	**Возвращает**
SetLevel(Level) Установка мощности сигнала	Level - мощность сигнала, дБм	1, если операция успешна, иначе 0
SetFreq(Freq) Установка частоты сигнала	Freq - частота сигнала, может быть как string 10MHz, так и число 10E6	1, если операция успешна, иначе 0
SetRFOutput(State) Включение/выключение RF выхода	State, string - ON/OFF	1, если операция успешна, иначе 0
SetGPS(SatNumber) Имитация сигналов заданного числа спутников(GPS, диапазон L1, расположение - Москва, статика, мощность всех сигналов одинакова)	SatNumber - количество спутников	1, если операция успешна, иначе 0

Класс CRSC

Описание

Класс, созданный для управления аттенюатором RSC. На данный момент реализованы следующие возможности:

1. Установка заданного ослабления

Методы

Методы		
Метод	**Входные аргументы**	**Возвращает**
SetAttenuation(ATT) Установка заданного ослабления	ATT - ослабление, дБ	1, если операция успешна, иначе 0

Класс CReceiver

Описание

Класс, созданный для работы с навигационными модулями. На данный момент реализованы следующие возможности:

1. Рестарт приемника (для GEOS - 3)
2. Получение статуса решения и сохранение его в переменную класса FixType (для GEOS - 3)
3. Чтение данных, посылаемых приемником по последовательному порту

Методы

Методы		
Метод	**Входные аргументы**	**Возвращает**
SerialConfig(COM, Baud) Настройка соединения	COM,string - имя порта Baud - скорость передачи данных	---
SerialConnect Соединение с приемником	---	1, если операция успешна, иначе 0

RecieveString Однократное чтение данных	---	[Answer] строка данных
SerialClose Закрытие соединения	---	1, если операция успешна, иначе 0
Reset Перезагрузка приемника	---	---
GetSolutionStatus Получение статуса решения и сохранение его в переменную класса FixType (для GEOS - 3)	---	---

3. Проведение автоматических испытаний с помощью разработанного инструментария

Общие положения

Объектом исследования является навигационный модуль, предметом - чувствительность данного модуля в режиме слежения. Задача - провести большое число экспериментов в автоматическом режиме по близкой к уже имеющейся методике [1].

Алгоритм эксперимента

После включения генератора, приемника, каждые 30 секунд мощность сигнала уменьшается на шаг. Если в течение 5 секунд приемник перестает выдавать 3D решение - считается, что произошел срыв выдачи решения. При этом текущая мощность сигнала записывается и помечается как мощность срыва. Приемник перезагружается, уровень сигнала сбрасывается на начальное значение. Эксперимент повторяется вновь.

Оборудование

1. Набор классов и скрипт эксперимента
2. Генератор сигналов SMBV100A
3. Анализатор спектра FSV
4. Аттенюатор 15-20 дБ
5. МШУ
6. Навигационный модуль
7. Компьютер с Matlab и интерфейсным ПО навигационного модуля
8. Соединительные кабели

Шаг 1

Так как скрипт считывает мощность сигнала с генератора (параметр Level), нужно измерить при помощи анализатора спектра мощность сигнала и определить, на сколько дБ она отличается от показания Level генератора. В дальнейшем эксперименте анализатор спектра не участвует.

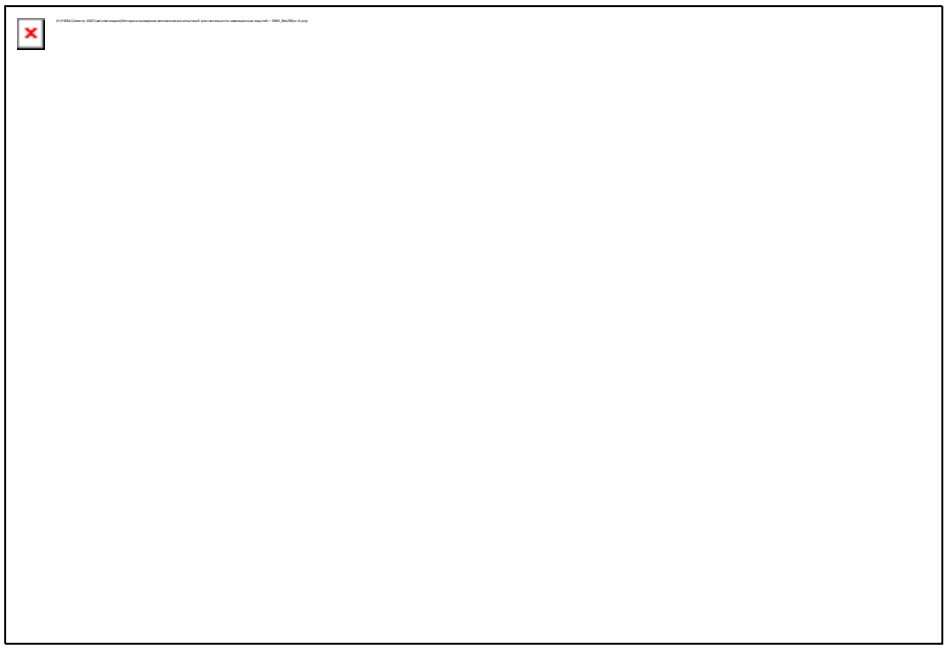

Рисунок 1. Схема установки

Шаг 2

Собрать схему установки (см рис. 1). Включить модуль, запустить интерфейсное ПО. Один из портов оставить для визуального контроля через ПО, другой - закрыть.

Шаг 3

Запустить Matlab, открыть скрипт эксперимента (Sensivity.m). В методе SerialConfig указать закрытый ранее порт, задать IP адрес генератора в методе SetConnection, запустить скрипт. Визуально проконтролировать запуск генератора, наличие спутников в решении, изменение уровня мощности сигнала. Если все нормально - можно оставлять установку на неопределенное время.

Шаг 4

Обработка результата. Время одного эксперимента при текущем шаге изменения мощности примерно 10 минут. Результаты эксперимента записываются в виде "массива массивов" [P 1/0], где P - мощность сигнала дБм, 1 записывается, если на данной мощности модуль нормально выдавал решение, 0 записывается в случае срыва решения при мощности P. Из полученного массива нужно вычесть поправку, измеренную в шаге 1, ослабление аттенюатора, а также примерно 6 или 8 дБ, т.к. в массиве записана мощность 4х или 6ти спутников.

Апробация методики

В соответствии с данной методикой проведена серия из 347 испытаний чувствительности навигационного модуля Геос - 3, результаты которой занесены в протокол.

Протокол испытаний №2013.03.03-1

Чувствительность НМ Геос-3 выполнен в рамках курсового проекта Днепрова В.В. и апробации системы автоматического тестирования *ArcticSEA*.

Объект испытаний

Навигационный модуль Геос - 3.

Цель испытаний

Целью испытаний является определение чувствительности навигационного модуля Геос - 3 в режиме слежения.

Оцениваемые показатели и расчетные соотношения

В ходе испытаний оценивается мощность сигнала 1 НС, при которой навигационный модуль перестанет выдавать 3D решение в течение 5 секунд.

Материально-техническое обеспечение

- Набор классов и скрипт эксперимента
- Генератор сигналов SMBV100A
- Анализатор спектра FSV
- Аттенюатор 15-20 дБ
- МШУ
- Навигационный модуль
- Компьютер с Matlab и интерфейсным ПО навигационного модуля
- Соединительные кабели

Условия проведения испытаний

В ходе испытаний использовались следующие параметры имитации сигналов GPS: количество НС - 6, мощность сигнала каждого спутника одинакова, расположение - Москва, статика. По описанной ранее методике была проведена серия из 347 экспериментов. Условия проведения каждого эксперимента одинаковы.

Результаты испытаний

По результатам испытаний получены зависимости вероятности срыва решения / выдачи 3D решения от мощности сигнала 1 НС (см. рис 2, 3).

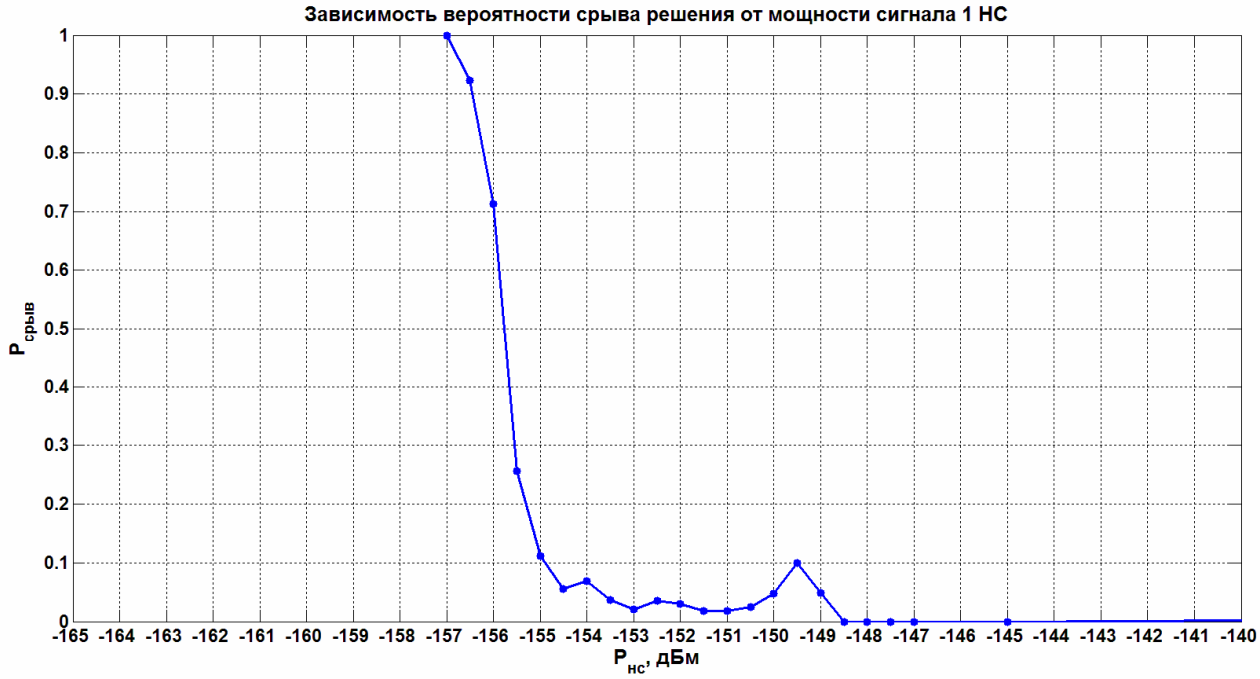

Рисунок 2. Вероятность срыва решения

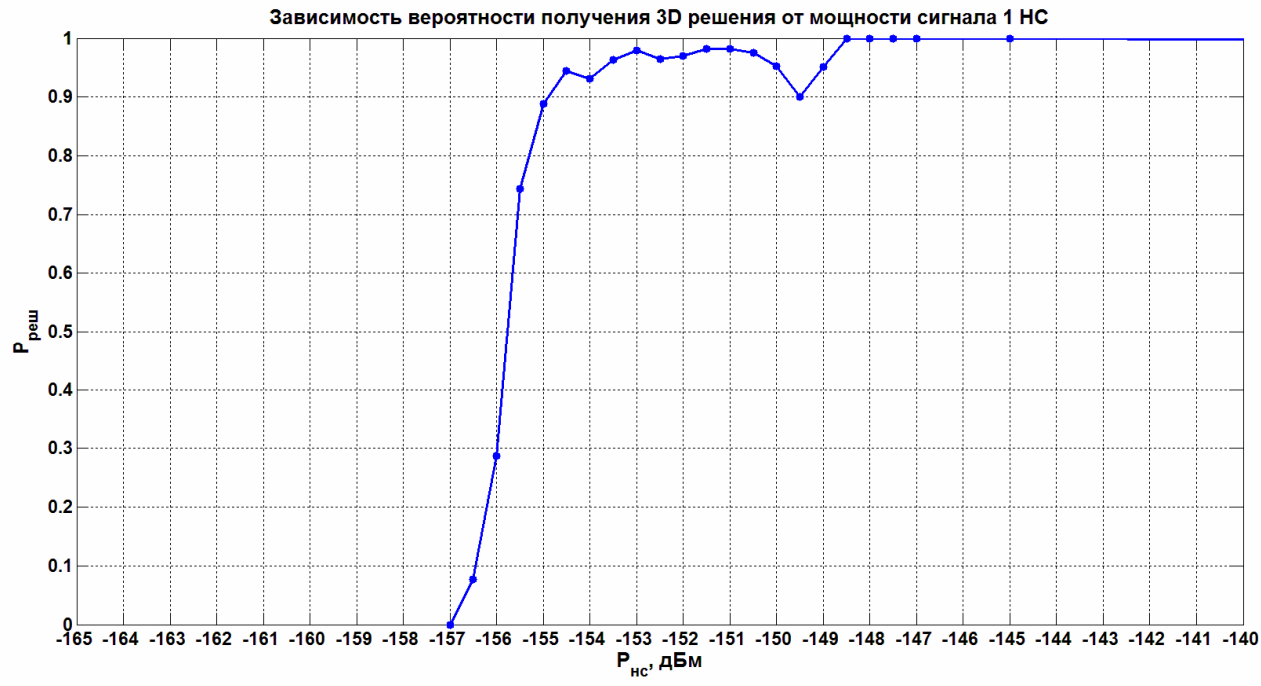

Рисунок 3. Вероятность выдачи решения

Выводы

Полученные результаты позволяют оценить вероятность работы (выдачи 3D решения) приемника при данной мощности сигнала 1 НС. Можно сказать, что с вероятностью 0.9 приемник будет выдавать навигационное решение при мощности сигнала 1 НС -155 дБм (-185 дБВт). Чувствительность приемника Геос - 3, указанная производителем, составляет -160 дБм. Куда делись 5 дБ мне трудно сказать, возможно, причина в прошивке приемника.

Заключение

Задание на курсовой проект выполнено. Для достижения поставленной цели решены следующие задачи:

1. Создана библиотека функций - набор классов MATLAB, позволяющая проводить автоматические испытания НАП
2. Разработана методика проведения автоматических испытаний чувствительности навигационных модулей с использованием созданной библиотеки функций
3. Проведена серия из 347 испытаний, в результате которой установлено: чувствительность навигационного модуля Геос - 3 составляет -155 дБм с вероятностью срыва решения 0.1
4. Проведенные испытания показали, что созданные классы успешно решают задачу автоматизации испытаний НАП

Использованные источники

1. http://srns.ru/wiki/Чувствительность_навигационных_модулей_(лабораторная_работа) – описание лабораторной работы по определению чувствительности навигационных модулей в курсе АП СРНС.
2. http://www.rohde-schwarz.com – сайт фирмы – производителя используемых в работе приборов.
3. http://code.google.com/p/arcticsea/ - репозиторий проекта, созданные в ходе работы классы доступны для скачивания, изменения; имеется справочная информация по каждому классу.

Приложение

Листинг классов

```
%> @file CSMBV.m
%> @author  Vladimir Dneprov <vvdneprov@gmail.com>
%> Moscow Power Engineering Institute
%>
%> @section LICENSE
%>
%> This program is free software; you can redistribute it and/or
%> modify it under the terms of the GNU General Public License as
%> published by the Free Software Foundation; either version 2 of
%> the License, or (at your option) any later version.
%>
%> This program is distributed in the hope that it will be useful, but
%> WITHOUT ANY WARRANTY; without even the implied warranty of
%> MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
%> General Public License for more details at
%> http://www.gnu.org/copyleft/gpl.html
%>
%> @section DESCRIPTION
%>
%> Class for Rohde&Schwarz SMBV100A vector signal generator\n
%> Features:\n
%> 1) Set up output power\n
%> 2) Set up carrier frequency\n
%> 3) Simulate GPS signals of the given number of satellites\n

%> @brief Class for Rohde&Schwarz SMBV100A vector signal generator
classdef CSMBV < handle

    properties
        %> Frequency of the carrier Hz
```

```
    Freq
    %> Output power dBm
    Level
    %> Pointer of TCP/IP connection
    Instr
end

methods

    %> @brief Consturctor of this class
    %>
    %> This is constructor of SMBV control class\n
    %> Example: SMBV1 = CSMBV;
    %>
    %> @return RS Object of this class
    function RS = CSMBV

    end

    %> @brief Connect to unit by local network
    %>
    %> Example: SMBV1.SetConnection('192.168.1.55', 5025);
    %> @param IP String IP-address of unit
    %> @param port Port for TCP/IP connection, usually 5025
    %> @return Status Is 0 - fail; 1 - ok.
    function Status = SetConnection(RS, IP, port)
        Status = 0;
        RS.Instr = tcpip(IP, port);

        %Uncomment to change tcpip object's R/W buffer size.
        %Default = 512 bytes
        %set(RS.Instr, InputBufferSize, 2000);
        %set(RS.Instr, OutputBufferSize, 2000);
```

```
    fopen(RS.Instr);
    if  strcmp(get(RS.Instr,'Status'),'open')
        fprintf('SMBV:Connection OK\n');
        Status = 1;
    else
        fprintf('SMBV:Connection Problem');
    end
end

%> @brief Close connection and return to manual control
%>
%> Example: SMBV1.CloseConnection;
%> @return Status Returns a status of 0 when the close operation is successful. Otherwise, it returns -1
function Status = CloseConnection(RS)
    fprintf(RS.Instr,'&GTL');
    Status = fclose(RS.Instr);
end
%> @brief Send SCPI command to SMBV
%>
%> Example: SMBV1.SendCommand('*RST');
%> @param strCommand String of SCPI command
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
function [Status] = SendCommand(RS, strCommand)
    Status = 0;
    % Check number of input args
    if( nargin ~= 2 )
        disp( '*** Wrong number of input arguments' )
        return;
    end

    % Check first parameter
    if( isobject(RS) ~= 1 )
        disp( '*** The first parameter is not an object.' );
        return;
```

```
    end

    % Check second parameter
    if( isempty(strCommand) || (ischar(strCommand)~= 1) )
        disp ('*** Command string is empty or not a string.');
        return;
    end

    % Command sending
    fprintf (RS.Instr, strCommand);
    Error = QueryError(RS);
    if (Error==1)
        return;
    end
    Status = 1;
end

%> @brief Send request for answer
%>
%> Example: SMBV1.SendQuery('*OPC?');
%> @param strCommand String of SCPI command
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
%> @return Result Returns a answer of SMBV
function [Status, Result] = SendQuery(RS, strCommand)
    Status = 0;
    Result = '';

    % Check number of input args
    if( nargin ~= 2 )
        disp( '*** Wrong number of input arguments' )
        return;
    end

    % Check first parameter
```

```
    if( isobject(RS) ~= 1 )
        disp( '*** The first parameter is not an object.' );
        return;
    end

    % Check second parameter
    if( isempty(strCommand) || (ischar(strCommand)~= 1) )
        disp ('*** Command string is empty or not a string.');
        return;
    end

    % Check: is strCommand the question?
    if( isempty( strfind(strCommand, '?' ) ) )
        disp( '*** Queries must end with a question mark.' );
        return;
    end

    % Send request and receive answer
    fprintf (RS.Instr, strCommand);
    Result = fscanf(RS.Instr, '%c');
    Status = 1;
end

%> @brief Get information about instrument
%>
%> Example: SMBV1.GetIDN;
%> @return IDN String information about instrument
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
function [Status, IDN] = GetIDN(RS)
    Status = 0;
    IDN = '';
    [Stat, IDN] = RS.SendQuery('*IDN?');
    if (Stat == 0)
        return;
```

```
    end
    Status = 1;
end

%> @brief Preset instrument and clear errors log
%>
%> Example: SMBV1.Preset;
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
function [Status] = Preset(RS)
    Status = 0;
    [Stat] = RS.SendCommand('*RST; *CLS');
    if (Stat == 0)
        return;
    end
    [status, result] = RS.SendQuery('*OPC?');
    if (status == 0 || result(1)~='1')
        return;
    end
    Status = 1;
end

%> @brief Check errors in SMBV
%>
%> Example: SMBV1.QueryError;
%> @return Err Err = 1 instrument error occured, 0 no error
function [Err] = QueryError(RS)
    Result = '1';
    Counter = 0;
    Err = 0;
    % query for errors in a loop until "0, No error" is returned
    % and limit the number of iterations to 100
    while (Result(1) ~= '0' && Counter < 100)
        [status, Result] = RS.SendQuery('SYST:ERR?');
        if (status == 0)
```

```
            disp( '*** Error occurred' );
            Err = 1;
            break;
        end
        if (Result(1)~='0')
            disp (['*** Instrument Error: ' Result]);
            Err = 1;
        end
        Counter = Counter + 1;
    end
end

%> @brief Set the output power level
%>
%> Example: SMBV1.SetLevel(-130);
%> @param Level Output power level, dBm
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
function [Status] = SetLevel(RS, Level)
    Status = 0;
    RS.Level = Level;
    [Stat] = RS.SendCommand(sprintf('SOUR:POW %.1f', Level));
    if (Stat == 0)
        return;
    end
    Status = 1;
end

%> @brief Set the frequency of the carrier
%>
%> Example:\n SMBV1.SetFreq('1GHz');\n
%> SMBV1.SetFreq('1E9');\n
%> @param Freq Frequency of the carrier. Freq can be String 10GHz or float(Hz) 10.12345E9
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
function [Status] = SetFreq(RS, Freq)
```

```
    Status = 0;
    if (ischar(Freq))
        [Stat] = RS.SendCommand(['SOUR:FREQ ' Freq]);
        RS.Freq = str2num(Freq);
        if ( Stat == 0)
            return;
        end
    else
        [Stat] = RS.SendCommand(sprintf('SOUR:FREQ %.5f', Freq));
        RS.Freq = Freq;
        if (Stat == 0)
            return;
        end
    end
    RS.CenterFreq = Freq;
    Status = 1;
end

%> @brief Set the RF output ON/OFF
%>
%> Example: SMBV1.SetRFOutput('ON');
%> @param State String ON/OFF
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
function [Status] = SetRFOutput(RS, State)
    Status = 0;
    if (ischar(State))
        [Stat] = RS.SendCommand(['OUTP ' State]);
        if (Stat == 0)
            return;
        end
    else
        disp('Output state is string ON/OFF');
        return;
    end
```

```
    Status = 1;
end


%> @brief Simulate signals of several GPS satellites, satellite signal power is the same, the frequency range is L1, location - Moscow
%>
%> Example: SMBV1.SetGPS(6);
%> @param SatNumber Integer number of satellites
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
function [Status] = SetGPS(RS, SatNumber)
    Status = 0;
    %           RS.RefPoW = RefPow;
    [Stat] = RS.SendCommand('SOUR:BB:GPS:PRES');
    if (Stat == 0)
        return;
    end
    [Stat, result] = RS.SendQuery('*OPC?');
    if (Stat == 0 || result(1)~='1')
        return;
    end
    [Stat] = RS.SendCommand('SOUR:BB:GPS:STAT OFF');
    if (Stat == 0)
        return;
    end
    [Stat] = RS.SendCommand('SOUR:BB:GPS:SMODE USER');
    if (Stat == 0)
        return;
    end
    [Stat] = RS.SendCommand('SOUR:BB:GPS:LOC:SEL "Moscow"');
    if (Stat == 0)
        return;
    end
    %           [status] = RS.sendCommand(sprintf('SOUR:BB:GPS:POWER:REF %.1f', RefPow));
    %           if (status<0)
```

```
%           return;
%          end

[Stat] = RS.SendCommand(sprintf('BB:GPS:SAT:COUN %i', SatNumber));
if (Stat == 0)
   return;
end
[Stat] = RS.SendCommand('SOUR:BB:GPS:GOC');
if (Stat == 0)
   return;
end
[Stat, result] = RS.SendQuery('*OPC?');
if (Stat == 0 || result(1)~='1')
   return;
end
[Error] = RS.QueryError;
if (Error == 1)
   return;
end

[Stat] = RS.SendCommand('SOUR:BB:GPS:STAT ON');
if (Stat == 0)
   return;
end

progress = '0';
[Stat, progress] = RS.SendQuery('SOUR:BB:PROG:MCOD?');
if (Stat == 0)
   return;
end
while (str2num(progress) ~= 100)
   [Stat, progress] = RS.SendQuery('SOUR:BB:PROG:MCOD?');
   if (Stat == 0)
      return;
```

```
            end
        end

        [Stat, result] = RS.SendQuery('*OPC?');
        if (Stat == 0 || result(1)~='1')
            return;
        end

        [Stat] = RS.SendCommand('BB:GPS:TRIGger:EXECute');
        if (Stat == 0)
            return;
        end

        [Error] = RS.QueryError;
        if (Error == 1)
            return;
        end
        Status = 1;
    end

  end

end


%> @file CRSC.m
%> @author  Vladimir Dneprov <vvdneprov@gmail.com>
%> Moscow Power Engineering Institute
%>
%> @section LICENSE
%>
%> This program is free software; you can redistribute it and/or
%> modify it under the terms of the GNU General Public License as
%> published by the Free Software Foundation; either version 2 of
```

```
%> the License, or (at your option) any later version.
%>
%> This program is distributed in the hope that it will be useful, but
%> WITHOUT ANY WARRANTY; without even the implied warranty of
%> MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
%> General Public License for more details at
%> http://www.gnu.org/copyleft/gpl.html
%>
%> @section DESCRIPTION
%>
%> Class for Rohde&Schwarz RSC attenuator\n
%> Features:\n
%> 1) Set up attenuation

%> @brief Class for Rohde&Schwarz RSC attenuator
classdef CRSC < handle

   properties
      %> Pointer of TCP/IP connection
      Instr
      %> Attenuation dB
      Attenuation
   end

   methods



      %> @brief Consturctor of this class
      %>
      %> This is constructor of RSC control class\n
      %> Example: RSC3 = CRSC;
      %>
      %> @return RS Object of this class
      function RS = CRSC
```

```
end

%> @brief Connect to unit by local network
%>
%> Example: RSC3.SetConnection('192.168.1.58', 5025);
%> @param IP String IP-address of measurement unit
%> @param port Port for TCP/IP connection, usually 5025
%> @return Status Is 0 - fail; 1 - ok.
function Status = SetConnection(RS, IP, port)
   Status = 0;
   RS.Instr = tcpip(IP, port);

   %Uncomment to change tcpip object's R/W buffer size.
   %Default = 512 bytes
   %set(RS.Instr, InputBufferSize, 2000);
   %set(RS.Instr, OutputBufferSize, 2000);

   fopen(RS.Instr);
   if  strcmp(get(RS.Instr,'Status'),'open')
      fprintf('RSC:Connection OK\n');
      Status = 1;
   else
      fprintf('RSC:Connection Problem');
   end
end

%> @brief Close connection and return to manual control
%>
%> Example: RSC3.CloseConnection;
%> @return Status Returns a status of 0 when the close operation is successful. Otherwise, it returns -1
function Status = CloseConnection(RS)
   fprintf(RS.Instr,'&GTL');
   Status = fclose(RS.Instr);
```

```
end

%> @brief Send SCPI command to RSC
%>
%> Example: RSC3.SendCommand('*RST');
%> @param strCommand String of SCPI command
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
function [Status] = SendCommand(RS, strCommand)
   Status = 0;
   % Check number of input args
   if( nargin ~= 2 )
      disp( '*** Wrong number of input arguments' )
      return;
   end

   % Check first parameter
   if( isobject(RS) ~= 1 )
      disp( '*** The first parameter is not an object.' );
      return;
   end

   % Check second parameter
   if( isempty(strCommand) || (ischar(strCommand)~= 1) )
      disp ('*** Command string is empty or not a string.');
      return;
   end

   % Command sending
   fprintf (RS.Instr, strCommand);
   Error = QueryError(RS);
   if (Error==1)
      return;
   end
   Status = 1;
```

```
end

%> @brief Send request for answer
%>
%> Example: RSC3.SendQuery('*IDN?');
%> @param strCommand String of SCPI command
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
%> @return Result Returns an answer of FSV
function [Status, Result] = SendQuery(RS, strCommand)
   Status = 0;
   Result = '';

   % Check number of input args
   if( nargin ~= 2 )
      disp( '*** Wrong number of input arguments' )
      return;
   end

   % Check first parameter
   if( isobject(RS) ~= 1 )
      disp( '*** The first parameter is not an object.' );
      return;
   end

   % Check second parameter
   if( isempty(strCommand) || (ischar(strCommand)~= 1) )
      disp ('*** Command string is empty or not a string.');
      return;
   end

   % Check: is strCommand the question?
   if( isempty( strfind(strCommand, '?' ) ) )
      disp( '*** Queries must end with a question mark.' );
      return;
```

```
    end

    % Send request and receive answer
    fprintf (RS.Instr, strCommand);
    Result = fscanf(RS.Instr, '%c');
    Status = 1;
end

%> @brief Get information about instrument
%>
%> Example: RSC3.GetIDN;
%> @return IDN String information about instrument
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
function [Status, IDN] = GetIDN(RS)
    Status = 0;
    IDN = '';
    [Stat, IDN] = RS.SendQuery('*IDN?');
    if (Stat == 0)
        return;
    end
    Status = 1;
end

%> @brief Preset instrument and clear errors log
%>
%> Example: RSC3.Preset;
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
function [Status] = Preset(RS)
    Status = 0;
    [Stat] = RS.SendCommand('*RST; *CLS');
    if (Stat == 0)
        return;
    end
    [status, result] = RS.SendQuery('*OPC?');
```

```
    if (status == 0 || result(1)~='1')
        return;
    end
    Status = 1;
end

%> @brief Check errors in RSC
%>
%> Example: RSC3.QueryError;
%> @return Err Err = 1 instrument error occured, 0 no error
function [Err] = QueryError(RS)
    Result = '1';
    Counter = 0;
    Err = 0;
    % query for errors in a loop until "0, No error" is returned
    % and limit the number of iterations to 100
    while (Result(1) ~= '0' && Counter < 100)
        [status, Result] = RS.SendQuery('SYST:ERR?');
        if (status == 0)
            disp( '*** Error occurred' );
            Err = 1;
        end
        if (Result(1)~='0')
            disp (['*** Instrument Error: ' Result]);
            Err = 1;
        end
        Counter = Counter + 1;
    end
end

%> @brief Set the attenuation
%>
%> Example: RSC3.SetAttenuation(30);
%> @param ATT attenuation dB
```

```
        %> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
        function [Status] = SetAttenuation(RS, ATT)
            Status = 0;
            [Stat] = RS.SendCommand(sprintf('ATT1:ATT %.2f', ATT));
            if ( Stat == 0)
                return;
            end
            RS.Attenuation = ATT;
            Status = 1;
        end
    end

end



%> @file CFSV.m
%> @author  Vladimir Dneprov <vvdneprov@gmail.com>
%> Moscow Power Engineering Institute
%>
%> @section LICENSE
%>
%> This program is free software; you can redistribute it and/or
%> modify it under the terms of the GNU General Public License as
%> published by the Free Software Foundation; either version 2 of
%> the License, or (at your option) any later version.
%>
%> This program is distributed in the hope that it will be useful, but
%> WITHOUT ANY WARRANTY; without even the implied warranty of
%> MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
%> General Public License for more details at
%> http://www.gnu.org/copyleft/gpl.html
%>
%> @section DESCRIPTION
%>
```

```
%> Class for Rohde&Schwarz FSV signal & spectrum analyzers\n
%> Features:\n
%> 1) Power measurement in the given bandwidth\n
%> 2) Setup center frequency of the spectrum analysis\n
%> 3) Setup bandwidth of the spectrum analysis\n

%> @brief Class for Rohde&Schwarz FSV signal & spectrum analyzers
classdef CFSV < handle


    properties
        %> Pointer of TCP/IP connection
        Instr
        %> Band for spectrum analysis
        Span
        %> Central frequency for spectrum analysis
        CenterFreq
    end

    methods


        %> @brief Consturctor of this class
        %>
        %> This is constructor of FSV control class\n
        %> Example: FSV3 = CFSV;
        %>
        %> @return RS Object of this class
        function RS = CFSV

        end

        %> @brief Connect to unit by local network
        %>
```

```
%> Example: FSV3.SetConnection('192.168.1.100', 5025);
%> @param IP String IP-address of measurement unit
%> @param port Port for TCP/IP connection, usually 5025
%> @return Status Is 0 - fail; 1 - ok.
function Status = SetConnection(RS, IP, port)
   Status = 0;
   RS.Instr = tcpip(IP, port);

   %Uncomment to change tcpip object's R/W buffer size.
   %Default = 512 bytes
   %set(RS.Instr, InputBufferSize, 2000);
   %set(RS.Instr, OutputBufferSize, 2000);

   fopen(RS.Instr);
   if  strcmp(get(RS.Instr,'Status'),'open')
      fprintf('FSV:Connection OK\n');
      Status = 1;
   else
      fprintf('FSV:Connection Problem');
   end
end

%> @brief Close connection and return to manual control
%>
%> Example: FSV3.CloseConnection;
%> @return Status Returns a status of 0 when the close operation is successful. Otherwise, it returns -1
function Status = CloseConnection(RS)
   fprintf(RS.Instr,'&GTL');
   Status = fclose(RS.Instr);
end

%> @brief Send SCPI command to FSV
%>
%> Example: FSV3.SendCommand('*RST');
```

```
%> @param strCommand String of SCPI command
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
function [Status] = SendCommand(RS, strCommand)
   Status = 0;
   % Check number of input args
   if( nargin ~= 2 )
      disp( '*** Wrong number of input arguments' )
      return;
   end

   % Check first parameter
   if( isobject(RS) ~= 1 )
      disp( '*** The first parameter is not an object.' );
      return;
   end

   % Check second parameter
   if( isempty(strCommand) || (ischar(strCommand)~= 1) )
      disp ('*** Command string is empty or not a string.');
      return;
   end

   % Command sending
   fprintf (RS.Instr, strCommand);
   Error = QueryError(RS);
   if (Error==1)
      return;
   end
   Status = 1;
end

%> @brief Send request for answer
%>
%> Example: FSV3.SendQuery('*IDN?');
```

```
%> @param strCommand String of SCPI command
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
%> @return Result Returns an answer of FSV
function [Status, Result] = SendQuery(RS, strCommand)
   Status = 0;
   Result = '';

   % Check number of input args
   if( nargin ~= 2 )
      disp( '*** Wrong number of input arguments' )
      return;
   end

   % Check first parameter
   if( isobject(RS) ~= 1 )
      disp( '*** The first parameter is not an object.' );
      return;
   end

   % Check second parameter
   if( isempty(strCommand) || (ischar(strCommand)~= 1) )
      disp ('*** Command string is empty or not a string.');
      return;
   end

   % Check: is strCommand the question?
   if( isempty( strfind(strCommand, '?' ) ) )
      disp( '*** Queries must end with a question mark.' );
      return;
   end

   % Send request and receive answer
   fprintf (RS.Instr, strCommand);
   Result = fscanf(RS.Instr, '%c');
```

```
    Status = 1;
end

%> @brief Get information about instrument
%>
%> Example: FSV3.GetIDN;
%> @return IDN String information about instrument
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
function [Status, IDN] = GetIDN(RS)
    Status = 0;
    IDN = '';
    [Stat, IDN] = RS.SendQuery('*IDN?');
    if (Stat == 0)
        return;
    end
    Status = 1;
end

%> @brief Preset instrument and clear errors log
%>
%> Example: FSV3.Preset;
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
function [Status] = Preset(RS)
    Status = 0;
    [Stat] = RS.SendCommand('*RST; *CLS');
    if (Stat == 0)
        return;
    end
    [status, result] = RS.SendQuery('*OPC?');
    if (status == 0 || result(1)~='1')
        return;
    end
    Status = 1;
end
```

```
%> @brief Check errors in FSV
%>
%> Example: FSV3.QueryError;
%> @return Err Err = 1 instrument error occured, 0 no error
function [Err] = QueryError(RS)
   Result = '1';
   Counter = 0;
   Err = 0;
   % query for errors in a loop until "0, No error" is returned
   % and limit the number of iterations to 100
   while (Result(1) ~= '0' && Counter < 100)
      [status, Result] = RS.SendQuery('SYST:ERR?');
      if (status == 0)
         disp( '*** Error occurred' );
         Err = 1;
      end
      if (Result(1)~='0')
         disp (['*** Instrument Error: ' Result]);
         Err = 1;
      end
      Counter = Counter + 1;
   end
end

%> @brief Set the center frequency of the spectrum analysis
%>
%> Example:\n FSV3.SetCenterFreq('1GHz');\n
%> FSV3.SetCenterFreq(1E9);
%> @param Freq Central frequency string 1GHz or float 1.11E9 (Hz)
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
function [Status] = SetCenterFreq(RS, Freq)
   Status = 0;
   if (ischar(Freq))
```

```
        [Stat] = RS.SendCommand(['FREQ:CENT ' Freq]);
        if ( Stat == 0)
            return;
        end
    else
        [Stat] = RS.SendCommand(sprintf('FREQ:CENT %.5f',Freq));
        if ( Stat == 0)
            return;
        end
    end
    RS.CenterFreq = Freq;
    Status = 1;
end

%> @brief Set bandwidth of the spectrum analysis
%>
%> Example:\n FSV3.SetSpan('1MHz');\n
%> FSV3.SetSpan(1E6);
%> @param Span String 1MHz or float 1.1E6
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
function [Status] = SetSpan(RS, Span)
    Status = 0;
    if (ischar(Span))
        [Stat] = RS.SendCommand(['FREQ:SPAN ' Span]);
        if ( Stat == 0)
            return;
        end
    else
        [Stat] = RS.SendCommand(sprintf('FREQ:SPAN %.5f',Span));
        if ( Stat == 0)
            return;
        end
    end
    RS.Span = Span;
```

```
    Status = 1;
end


%> @brief Measure power in given bandwith
%>
%> Example:\n FSV3.PowerMeasure('10MHz');\n
%> FSV3.PowerMeasure(1E7);
%> @param Bandwidth String 1MHz or float 1.123E6 (Hz)
%> @return measure Power in bandwith, dBm
%> @return Status Returns a status of 1 when the operation is successful. Otherwise, it returns 0
function [Status, measure] = PowerMeasure(RS, Bandwidth)
    Status = 0;
    if (ischar(Bandwidth))
        [Stat] = RS.SendCommand(['SENS:POW:ACH:BAND:CHAN1 ', Bandwidth]);
        if ( Stat == 0)
            return;
        end
    else
        [Stat] = RS.SendCommand(sprintf('SENS:POW:ACH:BAND:CHAN1 %.5f', Bandwidth));
        if ( Stat == 0)
            return;
        end
    end
    [status, result] = RS.SendQuery('*OPC?');
    if (status == 0 || result(1)~='1'),
        return;
    end
    [Stat] = RS.SendCommand('INIT:CONT OFF');
    if ( Stat == 0)
        return;
    end
    [Stat] = RS.SendCommand('INIT;*WAI');
    if ( Stat == 0)
        return;
```

```
            end
            [m, measure] = RS.SendQuery('CALC:MARK:FUNC:POW:RES? CPOW');
            if (m == 0 || result(1)~='1'),
                return;
            end
            Status = 1;
            disp(['Measure result:' measure]);
        end
    end
end
```

```
%> @file CReceiver.m
%> @author  Vladimir Dneprov <vvdneprov@gmail.com>
%> Moscow Power Engineering Institute
%>
%> @section LICENSE
%>
%> This program is free software; you can redistribute it and/or
%> modify it under the terms of the GNU General Public License as
%> published by the Free Software Foundation; either version 2 of
%> the License, or (at your option) any later version.
%>
%> This program is distributed in the hope that it will be useful, but
%> WITHOUT ANY WARRANTY; without even the implied warranty of
%> MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
%> General Public License for more details at
%> http://www.gnu.org/copyleft/gpl.html
%>
%> @section DESCRIPTION
%>
%> Class to work with navigation receiver via serial port\n
%> Features:\n
```

```
%> 1) Reset receiver ( only for GEOS - 3 )\n
%> 2) Get solution status ( only for GEOS - 3 )\n
%> 3) Read data from receiver

%> @brief Class to work with navigation receiver via serial port
classdef CReceiver < handle


  properties
    %> Pointer of serial connection
    obj
    %> Type of solution (1 - No solution, 2 - 2D, 3 - 3D)
    FixType
  end

  methods


    %> @brief Consturctor of this class
    %>
    %> This is constructor of Receiver control class\n
    %> Example: Rec1 = CReceiver;
    %>
    %> @return RCV Object of this class
    function RCV = CReceiver

    end

    %> @brief Set parameters of serial port
    %>
    %> Example: Rec1.SerialConfig('COM7', 115200);
    %> @param COM String port name
    %> @param Baud Baud rate
    function SerialConfig(RCV, COM, Baud)
```

```
    if ( nargin ~= 3)
        disp('***Wrong number of input arguments')
        return;
    end
    if ( isempty(COM) || (ischar(COM)~= 1) )
        disp('***Enter COM<x> as a string');
        return;
    end
    if (isempty(Baud))
        disp('***Enter Baud rate');
        return;
    end
    RCV.obj = serial(COM);
    set(RCV.obj,'BaudRate',Baud);
end

%> @brief Connect to receiver by serial port
%>
%> Example: Rec1.SerialConnect;
%> @return Status Is 0 - fail; 1 - ok.
function [Status] = SerialConnect(RCV)
    Status = 0;
    fopen(RCV.obj);
    if strcmp(get(RCV.obj,'Status'),'open')
        disp('***Serial: connection OK');
        Status = 1;
    else
        disp('***Serial: connection error');
        return;
    end
end

%> @brief Read data from receiver
%>
```

```
%> Example: Rec1.RecieveString;
%> @return Answer Data from receiver
function [Answer] = RecieveString(RCV)
   Answer = fscanf(RCV.obj);
end

%> @brief Close connection to receiver
%>
%> Example: Rec1.SerialClose;
%> @return Status Is 0 - fail; 1 - ok.
function [Status] = SerialClose(RCV)
   Status = 0;
   fclose(RCV.obj);
   if strcpm(get(RCV.obj,'Status'),'close')
      disp('***Serial: close OK');
      Status = 1;
   else
      disp('***Serial: close error');
      return;
   end
end

%> @brief Reset receiver ( NMEA string is true for GEOS-3 )
%>
%> Example: Rec1.Reset;
function Reset(RCV)
   fprintf(RCV.obj,'$GPSGG,CSTART*6B\n\r');
end

%> @brief Get solution status and store it in FixType
%>
%> Example: Rec1.GetSolutionStatus;
function GetSolutionStatus(RCV)
   while(1)
```

```
            answer = RecieveString(RCV);
            if numel(answer) < 10
                return;
            end
            if strcmp(answer(1:6),'$GNGSA')||strcmp(answer(1:6),'$GPGSA')
                sol = answer(10);
                switch sol
                    case '1'
                        RCV.FixType = 1;
                        disp('No solution')
                        return;
                    case '2'
                        RCV.FixType = 2;
                        disp('2D Fix')
                        return;
                    case '3'
                        RCV.FixType = 3;
                        disp('3D Fix')
                        return;
                end
            end
        end
    end
end
```

Скрипт эксперимента

```
SMBV = CSMBV;
Rec = CReceiver;

%соединение с SMBV
[Stat] = SMBV.SetConnection('192.168.1.22',5025);
if (Stat == 0)
    error('Connection problem')
```

```
end
%сброс настроек SMBV в дефолтные, очистка лога ошибок
[Stat] = SMBV.Preset;
if (Stat == 0)
   error('Error')
end
%запрос модели, серийного номера
[Stat, result] = SMBV.GetIDN;
if (Stat == 0)
   error('Error')
end
disp(result);
%проверка системных ошибок
[status, result] = SMBV.SendQuery('SYST:SERR?');
if (result(1) ~= '0' || status == 0 )
disp (['*** Instrument error : ' result]);
return;
end
%начальные настройки эксперимента
StartLevel = -95; % стартовая мощность сигнала
%запуск имитации сигнала GPS
[Stat] = SMBV.SetGPS(6);
if (Stat == 0)
   error('Error')
end
%установка стартовой мощности
[Stat] = SMBV.SetLevel(StartLevel);
if (Stat == 0)
   error('Error')
end
[Stat] = SMBV.SetRFOutput('ON');
if (Stat == 0)
   error('Error')
end
```

```
%Настройка соединения с приемником
Rec.SerialConfig('COM6',115200);
%Соединение с приемником
Stat = Rec.SerialConnect;
if (Stat == 0)
   error('Serial: connection problem')
end
%Перезагрузка приемника, запуск отсчета времени на данй мощности
Rec.Reset;
pause(70);
tin_thislevel = tic;

%Параметры эксперимента: шаг изменения мощности, ожидание на мощности
LevelStep = 1; PauseOnLevel = 30;
HaveFix = 0;
k = 1;
RecIsDead5sec = 0;
RecOkOnLastStep = 0;
Pow_arr = cell(1,1);
p = 1;
m = 0;

%цикл эксперимента
while (1)

   Rec.GetSolutionStatus;

   if (Rec.FixType == 3)
      RecOkOnLastStep = 1;
      HaveFix = 1;
      if (toc(tin_thislevel) > PauseOnLevel)
         LastOkLevel = SMBV.Level;
         Pow_arr{p,1} = [LastOkLevel 1];
```

```
        p = p + 1;
        if (LastOkLevel <= -95 && LastOkLevel >= -118)
            LevelStep = 8;
        elseif (LastOkLevel == -119)
            LevelStep = 2;
        elseif (LastOkLevel <= -120 && LastOkLevel >= -135)
            LevelStep = 0.5;
        end
        Stat = SMBV.SetLevel(LastOkLevel - LevelStep);
        if (Stat == 0)
            error('SMBV error')
        end
        tin_thislevel = tic;
    end
elseif ((Rec.FixType == 1 || Rec.FixType == 2) && RecOkOnLastStep == 1 )
    DeathTime = tic;
end

if ((Rec.FixType == 1 || Rec.FixType ==2) && HaveFix == 1 && RecOkOnLastStep == 0 )
if ( toc(DeathTime) > 5 )
    RecIsDead5sec = 1;
else
    RecIsDead5sec = 0;
end
end

if (Rec.FixType == 1 || Rec.FixType == 2)
    RecOkOnLastStep = 0;
end

if (RecIsDead5sec == 1)
    ResultLevel(k) = LastOkLevel;
    k = k + 1;
    Pow_arr{p,1} = [(LastOkLevel - LevelStep) 0];
```

```
        p = p + 1;
        file = [num2str(m) 'MSHUpower.mat'];
        save(file, 'Pow_arr');
        m = m + 1;
        Rec.Reset;
        SMBV.SetLevel(StartLevel);
        HaveFix = 0;
        RecOkOnLastStep = 0;
        RecIsDead5sec = 0;
        toc(DeathTime);
        pause(70);
        tin_thislevel  = tic;

    end

end
```