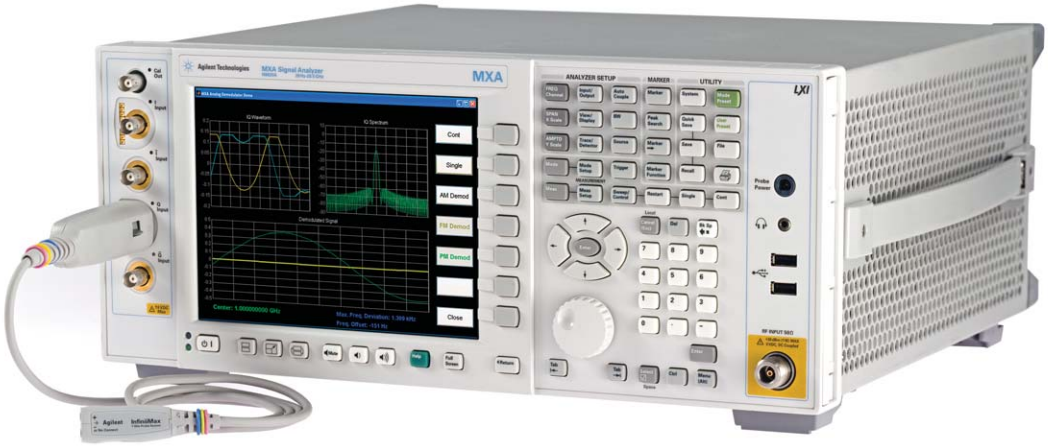
Using MATLAB® to Create Agilent Signal and Spectrum Analyzer Applications

Application Note



For use with
Agilent X-Series and PSA Series

Agilent Technologies

1.0 Introduction

Agilent X-Series signal and spectrum analyzers are built based on the Microsoft® Windows® operating system. Integrating this operating system into Agilent instruments enables users to interface these instruments to application software using industry-standard connectivity (LAN, USB, and GPIB), and develop and execute their own applications for the Agilent X-Series signal and spectrum analyzers using MATLAB software.

In the following application note, we discuss and demonstrate basic instrument communication using MATLAB code. We progress to more application-specific MATLAB code examples and then describe how you can start developing your own MATLAB-based applications integrated into the X-series signal and spectrum analyzers by referencing existing examples that you can download and modify.

This application note explains how to use MATLAB software to configure, control, and acquire data from X-Series signal and spectrum analyzers. The document then describes how you can use scripts developed in MATLAB to create, modifiy, and execute your own applications for X-Series signal and spectrum analyzers. You can also use many of the MATLAB examples in this application note with PSA high performance analyzers.

Finally, this application note provides references to on-line resources where you can download previously-developed MATLAB applications, which can be executed on your instrument directly or modified to suite your specific testing needs.

This application note aims to enhance your knowledge of using MATLAB with Agilent signal and spectrum analyzers. To learn more about getting started with MATLAB, users may also wish to refer to MATLAB's on-line documentation.

1.1 Overview of Agilent signal and spectrum analyzers

From DC to 325 GHz, Agilent Technologies offers X-Series and PSA signal and spectrum analyzers that enable you to analyze distortion, spurious, phase noise, and make 2G to 4G wireless communication measurements.

- The low-cost CXA is a versatile tool for essential signal characterization.
- The economy-class EXA is the fastest way to maximize throughput on the production line.
- The mid-performance MXA is the ultimate accelerator as your products move from design to manufacturing to the marketplace.
- The high-performance PXA is the evolutionary replacement for your current performance signal analyzer.

1.2 Agilent offers MATLAB software for Agilent instruments

MATLAB is used to create, develop, and execute X-Series applications. MATLAB is also used with Agilent instruments, including signal and spectrum analyzers to make measurements, analyze and visualize data, generate arbitrary waveforms, control instruments, execute modulation schemes, and build test systems. MATLAB provides interactive tools and command-line functions for data analysis tasks such as signal processing, signal modulation, digital filtering, and curve fitting.

MATLAB excels at math and matrix processing, can be used for communications DSP, and offers outstanding plotting and graphics functions. MATLAB makes an excellent companion program to the X-Series or PSA, whether running remotely on an external PC, or running directly inside the instrument. Under the control of MATLAB, the signal analyzer can acquire RF or microwave signals. These acquisitions can be scalar (magnitude-only versus frequency or time) or complex (magnitude and phase.) After transferring the data into MATLAB, a user-defined program can be used for fuctions such as further analysis, testing, and automatic test equipment (ATE) control.

In order to provide complete test and measurement solutions, Agilent now sells three different MATLAB software packages, which are available with the purchase of X-Series and PSA signal and spectrum analyzers. Each package contains MATLAB and various libraries called “toolboxes” to add functionality for specific fields. The MATLAB - Basic Signal Analysis Package (N6171A-M01) provides MATLAB and the Instrument Control Toolbox, which are required for instrument interaction in MATLAB. The MATLAB - Standard Signal Analysis Package (N6171A-M02) includes the products in the MATLAB - Basic Signal Analysis Package plus the Communications Toolbox and Signal Processing Toolbox. The MATLAB - Advanced Signal Analysis Package (N6171A-M03) includes the products in the MATLAB - Standard Signal Analysis Package plus the Filter Design Toolbox and RF Toolbox. To learn more about the typical uses of each package visit **www.agilent.com/find/n6171a**.

Please note that the examples provided in this application note require the products in the MATLAB - Standard Signal Analysis Package to run. (Inform your Agilent account manager if you need a trial of MATLAB software.) However, other toolboxes can be very useful for data analysis and signal processing, such as the Communications Toolbox and RF Toolbox.

2.0 Controlling Instruments in MATLAB using Interface Objects, Device Objects, and Instrument Drivers

MATLAB's Instrument Control Toolbox provides two different types of functions for interfacing with Agilent's signal and spectrum analyzers. I/O communication is made simple with the use of MATLAB's interface objects and device objects. An interface object allows the user to communicate with an instrument directly using a standard interface such as GPIB, TCP/IP, or VISA. A device object allows the user to communicate with an instrument at a higher level through an industry-standard instrument driver, such as an IVI-COM, IVI-C, or a self-developed instrument driver. The driver itself then uses the GPIB, TCP/IP, or VISA interface. The Instrument Control Toolbox also provides tools for developing MATLAB instrument drivers.

2.1 Controlling instruments in MATLAB using interface objects

The Instrument Control Toolbox provides several interface objects, but this section will focus on the GPIB, TCP/IP, and VISA interfaces. There are three basic steps to follow when using an interface object:

1) Open an instrument control session.
2) Connect to the instrument.
3) When finished, disconnect the instrument interface object.

The following is an example of how to communicate with an instrument using a General Purpose Interface Bus (GPIB) interface object.

1) Open the instrument control session.

```
GPIB_object = gpib('vendor', boardindex, primaryaddress);
```

If you were to use an Agilent 82357A USB to GPIB connector, then the statement might look like:

```
MXA = gpib('agilent', 7, 18);
```

2) Connect to the instrument.

```
fopen(MXA);
```

3) Communicate with the instrument.

```
identity = query(MXA, '*IDN?')
```

This will return the vendor, model number, serial number, and firmware revision of the instrument.

4) When finished, disconnect the instrument object.

```
fclose(MXA);
```

It is also a good idea to clear the object from the memory by using the **delete()** command.

```
delete(MXA);
```

The same basic steps are followed when using a Transmission Control Protocol/ Internet Protocol (TCP/IP) interface object.

1) Open the instrument control session.

```
TCPIP_object = tcpip('rhost', rport);
```

If you were to use an IP address of '10.10.10.10' and the port 5025, the command would look like:

```
mxa_ip = '10.10.10.10';
mxa_port = 5025;
mxa = tcpip(mxa_ip,mxa_port);
```

2) Connect to the instrument.

```
fopen(MXA);
```

3) Communicate with the instrument.

```
identity = query(MXA, '*IDN?')
```

This will return the vendor, model number, serial number, and firmware revision of the instrument.

4) When finished, disconnect the instrument object and delete it from memory.

```
fclose(MXA);
delete(MXA);
```

Again, the same steps are followed to create a virtual instrument standard architecture (VISA) interface object.
1) Open the instrument control session.

```
VISA_object = visa('vendor','rsrcname');
```

If we are using the Agilent N9020A X-Series signal and spectrum analyzer, then we could use the following command:

```
vendor = 'agilent';
rsrcname = 'TCPIP0::10.10.10.10::inst0::INSTR';
MXA = visa(vendor,rsrcname);
```

There are two ways to find out the resource name, if it is unknown. The first way is to use the Agilent Connection Expert (ACE). This program is included in version 15.0 or later of Agilent's IO Library Suite (available at **www.agilent.com/find/iolib**). To find the name in the ACE look at the address in the ( ) next to the model number. This is the resource name, which is also called the VISA address. The other way to find out the VISA address is to use the **instrhwinfo()** command provided by the Instrument Control Toolbox. In this case, you would type **info = instrhwinfo('visa','agilent')** in the MATLAB command window. This command will return a structure with a field named **ObjectConstructorName**.
info =

AdaptorDllName: [1x92 char]
AdaptorDllVersion: 'Version 2.6.0'
AdaptorName: 'AGILENT'
AvailableChassis: []
AvailableSerialPorts: ''
InstalledBoardIds: 0
ObjectConstructorName: {3x1 cell}
SerialPorts: ''
VendorDllName: 'agvisa32.dll'
VendorDriverDescription: 'Agilent Technologies VISA Driver'
VendorDriverVersion: 1

To look at the contents of this field, you would type **info.ObjectConstructorName** in the MATLAB command window. This will return a list of VISA addresses that are recognized by MATLAB. This list is directly connected to the instruments that are found in the ACE.

info.ObjectConstructorName

ans =

'visa('agilent', 'TCPIP0::141.121.94.85::inst0::INSTR');'
'visa('agilent', 'GPIB0::12::INSTR');'
'visa('agilent', 'GPIB0::18::INSTR');'

2) Connect to the instrument.

fopen(MXA);

3) Communicate with the instrument.

identity = query(MXA, '*IDN?')

This will return the vendor, model number, serial number, and firmware revision of the instrument.

4) When finished, disconnect the instrument object and delete it from memory.

fclose(MXA);
delete(MXA);

2.1.1 An example MATLAB program using interface objects

The following example shows the basic MXA communication procedure using functions of the Instrument Control Toolbox. This program **Basic_SCPI_Control.m** uses TCP/IP (LAN) connectivity to the instrument as it is one of the most widely used interfaces, and TCP/IP included in the MXA as a standard interface.

```
% TCPIP parameters of the Spectrum Analyzer
mxa_ip = '10.10.10.10';
mxa_port = 5025;

% MXA Interface creation and connection opening
fprintf('\nConnecting to Instrument ...\n');
mxa = tcpip(mxa_ip,mxa_port);
fopen(mxa);

% Intrument identification
idn = query(mxa,'*IDN?');
fprintf('Hello from %s', idn);

% Set the center frequency to 1 GHz using a SCPI command
fprintf(mxa,':FREQ:CENT 1 GHz');

% Set the span to 20 MHz
fprintf(mxa,':FREQ:SPAN 20 MHz');

% Set the reference level to +10 dBm
fprintf(mxa,':DISP:WIND:TRAC:Y:RLEV 10');

% Query the resolution bandwidth using fprinf()/fgets()
fprintf(mxa,':BAND:RES?');
rbw = str2double(fgets(mxa));
fprintf('Resolution bandwidth: %d kHz\n', rbw/1e3);

% Query the sweep time using query()
swp = str2double(query(mxa,':SWE:TIME?'));
fprintf('Sweep time: %d ms\n', round(swp*1000));

% Close the XA connection and clean up
fprintf('Disconnecting from Instrument ...\n');
fclose(mxa);

delete(mxa); % delete the interface object
clear mxa; % clear from the workspace
```

This program begins by defining the IP address and I/O port of the instrument, creating a TCP/IP interface and opening the interface for instrument communication. To find the IP address of a particular MXA, press the following keys [System] {Show} {System}. In this example, the TCP/IP interface is the variable **mxa**.

As shown in the previous section, the **fopen(mxa)** statement is used to open communication to the instrument. The **fprintf()** function is used to send SCPI commands to the instrument via the TCP/IP interface. This example sends the SCPI commands to set the center frequency, span, and reference level of the instrument.

Instrument parameters and data can be queried from the instrument using many different MATLAB functions. This example uses two different methods. The method used to query the resolution bandwidth of the instrument is a combination of the **fprintf()** and **fgets()** commands. The **fprintf(mxa,':BAND:RES?')**; command uses an SCPI command to place the value of resolution bandwidth in the output buffer and the **fgets(mxa)** command to read the data from the output buffer. The second method this example uses is the **query()** function. The **query()** function is essentially a combination of the **fprintf()** and **fgets()** functions. This example uses the **query(mxa,'*IDN?')**; function to retrieve information about the instrument's serial number, model number, and firmware revision. This program concludes by closing the TCP/IP interface using the **fclose(mxa)**; command and deleting the mxa variable from the workspace memory using the **delete(mxa)**; and **clear mxa**; commands.

The **instrhelp** command provides help on all the Instrument Control Toolbox functions and object properties. For example, typing **instrhelp tcpip** in the MATLAB command window displays the help text of the **tcpip()** function and lists all the properties of TCP/IP objects. Please refer to Product Help in MATLAB for a more detailed description of the Instrument Control Toolbox and its features.

2.1.2 A second example MATLAB program using interface objects

Acquiring trace data in spectrum analysis mode is one of the most common use cases when interacting with the instrument. The MATLAB script **Acquire Trace Single.m** demonstrates how a single trace can be acquired and plotted on a figure.

```
% Getting and plotting trace data
% Single trace acquisition
% Initial setup

mxa_ip = '10.10.10.10';
mxa_port = 5025;
mxa=tcpip(mxa_ip, 5025);

% input buffer size to receive trace data
% should be at least 4 times the number of trace
% points for 32-bit real format
set(mxa,'InputBufferSize',100000);
fopen(mxa);

% Set the data trace format to REAL, 32 bits
fprintf(mxa,':FORM:DATA REAL,32');

% Get the nr of trace points
nr_points = str2double(query(mxa,':SWE:POIN?'));

% Get the reference level
ref_lev = str2num(query(mxa,'DISP:WIND:TRAC:Y:RLEV?'));

% Get the trace data
fprintf(mxa,':TRAC? TRACE1');
data = binblockread(mxa,'float32');
fscanf(mxa); %removes the terminator character

% create and bring to front figure number 1
figure(1)

% Plot trace data vs sweep point index
plot(1:nr_points,data)

% Adjust the x limits to the nr of points
% and the y limits for 100 dB of dynamic range
xlim([1 nr_points])
ylim([ref_lev-100 ref_lev])

% activate the grid lines
grid on
title('Swept SA trace')
xlabel('Point index')
ylabel('Amplitude (dBm)')

% Disconnect an clean up
fclose(mxa);
delete(mxa);
clear mxa;
```

The input buffer size property of the TCP/IP interface object is changed from its default value (512) in order to accommodate longer trace records. The number of points should be at least the number of sweep points times the number of bytes required per data point (four in the case of real 32 data format) plus one for the terminator character. An input buffer size of 10^5 points is enough for this trace acquisition in Spectrum Analysis mode (SA) and for most processing in IQ Analyzer mode (Basic.)

Before the trace transfer, two instrument settings are read: the number of trace points and the reference level. The queried values are returned in a string format. The MATLAB functions **str2double()** and **str2num()** convert the string values into a numeric format. These values will be used to adjust the axis of the MATLAB plot. The trace data is obtained by sending the **':TRAC? TRACE1'** SCPI command. The trace data is then placed in the output buffer of the instrument and the data is retrieved in MATLAB via the **binblockread(object, precision)** function. This function interprets the binary data sent from the analyzer as an array of data (in this case an array of trace amplitudes) with the precision that is specified.

In this example the SCPI command **':FORM:DATA REAL,32'** is sent to set the format for the output data. The number format specified in **binblockread(mxa,'float32')** must be compatible with the data format that the instrument uses. The resulting plot is shown in Figure1.

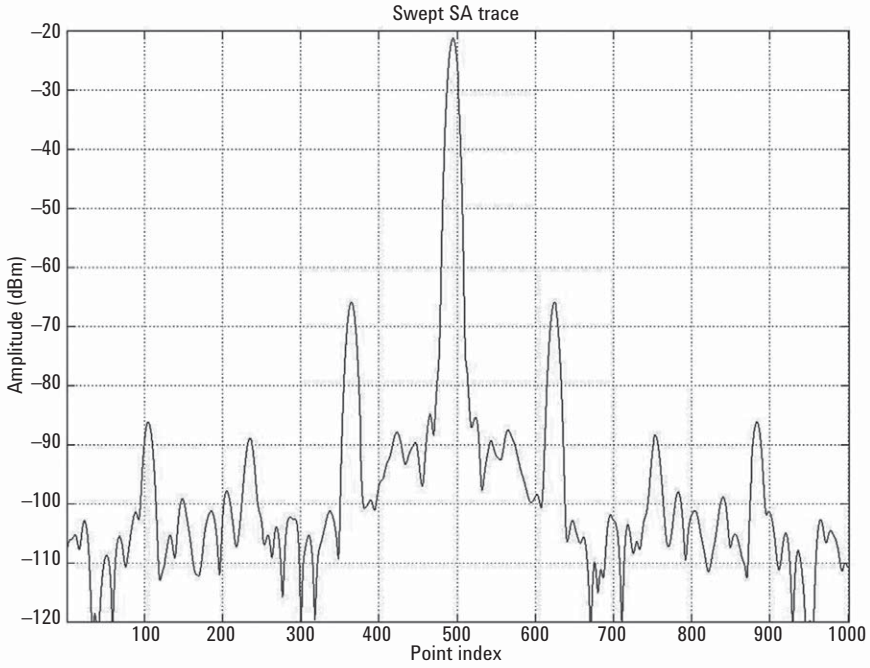


Figure 1. Plot of trace data obtained using MATLAB

If the frequency span is greater than zero hertz (also called zero span), the trace represents power versus frequency and the index point number can be converted into frequency using the following relationship:

freq = freq_center + freq_span * [point_index/(N-1) – 0.5]

For zero span, the trace is a representation of power versus time and the conversion from point index to time can be done using:

time = sweep_time * point_index/(N-1)

The **point_index** variable ranges from 0 to N-1, where N is the number of trace points. For consecutive trace acquisitions there are three methods available:

1) asynchronous
2) synchronous
3) timed trace

Asynchronous trace acquisition

This method simply transfers the trace contained in the instrument's memory, disregarding the sweep state. It provides the fastest possible trace update and requires only one SCPI command. The method is used in **Acquire Trace Cont Async.m** as shown in the Appendix.

Synchronous trace acquisition

In this method, a trace is transferred only after the instrument trace sweep is completed. It uses **':INIT:IMM;*OPC?'** and the program flow stops until the trace sweep operation completes. This method must be used if synchronization between the instrument trace sweep and the MATLAB trace processing is a requirement. **Acquire Trace Cont Timer.m** demonstrates this trace acquisition method.

Timed trace acquisition

The most versatile method, especially for graphical user interface (GUI) applications, is the timer method of trace acquisition. It relies on the **timer()** function, which defines the function to be called each time the timer expires and the timer repetition period. That function, **update_plot()**, is where the trace data is acquired and the plot updated. This method is covered in **Acquire Trace Timer.m** in the Appendix.

2.2 Creating a MATLAB instrument driver

The Instrument Control Toolbox provides a graphical instrument driver editor that allows you to develop drivers for Agilent's X-Series signal and spectrum analyzers. By creating a MATLAB instrument driver, you can create programs that use both the built-in functionality of the instrument and MATLAB. One of the major benefits to developing and using a driver in MATLAB is that it eliminates the need to look up the SCPI syntax for each of the intended instrument commands because these commands are now embedded inside the driver. Also, executing multiple commands or complex commands, such as obtaining trace data or raw IQ data, can be easily incorporated in the driver as the examples show in this section.

Note: If you are using an Agilent X-Series or PSA signal or spectrum analyzer, you typically will not have to create your own MATLAB instrument driver. A MATLAB instrument driver has already been created by Agilent and is available for download through **www.agilent.com/find/n6171a** or **www.mathworks.com/agilent**.

The tool used to develop drivers in MATLAB is called the MATLAB Instrument Driver Editor and it can be started through the MATLAB Start menu or by simply typing **midedit** in the MATLAB command window. This command will start the tool (see Figure 2) where you can begin to create your unique instrument driver.

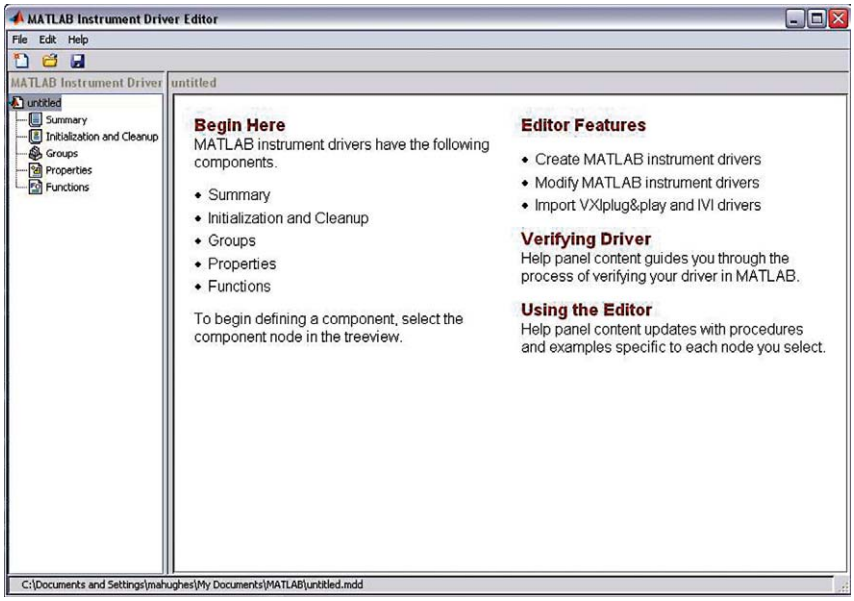


Figure 2. MATLAB Instrument Driver Editor

As you can see in Figure 2, the MATLAB Instrument Driver Editor can be comprised of five main components. We will talk about four of these components: summary, initialization and cleanup, functions, and properties.

The first step to creating a driver is to enter in the basic information about the driver in the summary section such as the instrument models the driver will support and the driver version.

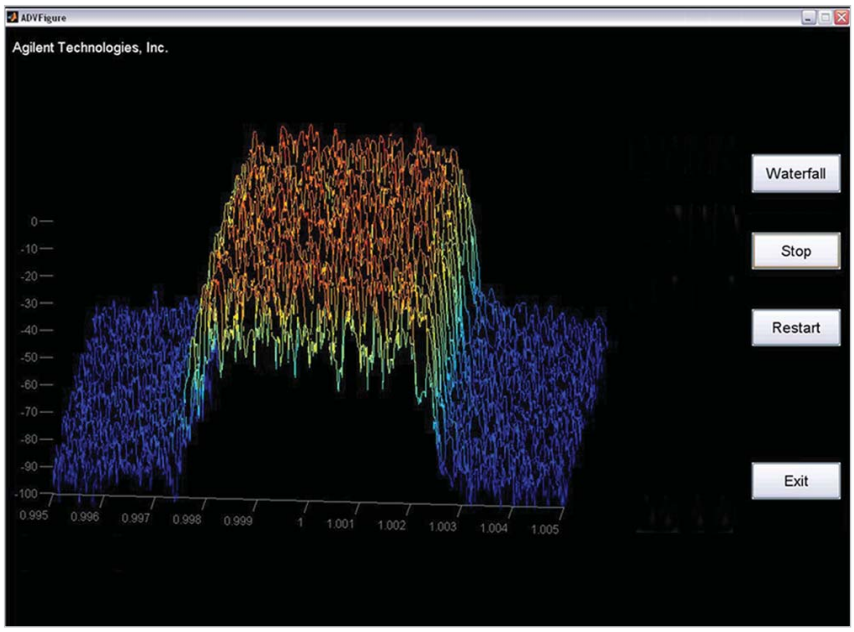


Figure 3. Enter driver summary information

The next step in creating the driver is to enter any initialization and cleanup code that is needed. For example, here is a portion of the **Create** code used in the driver that is used in the following examples:

```
function init(obj)
% This function is called after the object is created.
% OBJ is the device object.
interface=get(obj,'interface');
set(interface,'InputBufferSize',2e6);
set(interface,'Timeout',10);
fclose(interface);
% OBJ is the device object.
% End of function definition - DO NOT EDIT
```

As you can see, this code sets up some interface properties (buffer size and timeout.) In this portion of the code, you can also add functions or send SCPI commands (such as ***RST** or ***IDN?**) for initial instrument setup.

The “endianness” of your computer and instrument is an important consideration for your initialization code. Endianness is used to distinguish the byte order used to represent data. The little-endian requires that the data is sent with the least significant bit (LSB) first, whereas big-endian requires that the data is sent with the most significant bit (MSB) first.

The SCPI standard dictates that instruments send the data in the big-endian format. The MATLAB command **[str,maxsize,endian] = computer** will determine the computer type, maximum array size, and endianness of your computer. If your PC is little-endian, then you can include the **‘FORM:BORD SWAP’** SCPI command to change the byte order transmitted by the instrument. To set your analyzer back to big-endian, send the **‘FORM:BORD NORM’** SCPI command. Please note that these commands are only necessary for querying binary data.

The next step to creating a MATLAB instrument driver is to create any desired properties or functions. The properties portion of the driver allows you to use the **set** and **get** functions of the Instrument Control Toolbox. The **set** function allows you to send SCPI commands to the instrument. The **get** function allows you to query the instrument also via SCPI commands.

To add a property, right click on the **Properties** icon and select **Add Property** or highlight the **Properties** icon and enter the property name in the **Add Property** field. Once the property name has been entered, you can enter in the desired SCPI commands in the **get** or **set** fields under the **Code** tab. You will notice in this window a drop down menu for **Property style**. This drop down menu provides the option of using m-code or instrument commands (in this case SCPI commands) to create instrument properties. Under the **Properties Values** tab, you can set the data type for the property such as **Double** or **String**. Under the **General** tab, you can place any help documentation about the property.

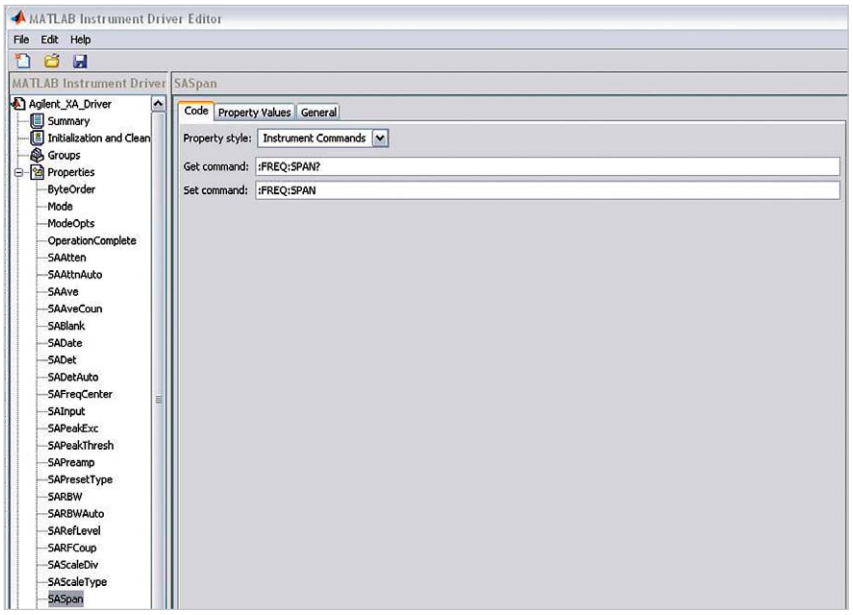


Figure 4. Creating desired properties or functions for MATLAB instrument driver

The **Functions** tab can be used to combine SCPI commands and MATLAB tools to perform complicated functions. You add a function in the same way as you add a property. Figure 5 shows a driver function called **"WavReadIQData"** that will read IQ data from the instrument and use MATLAB's matrix processing tools to create the I +jQ complex vector data.

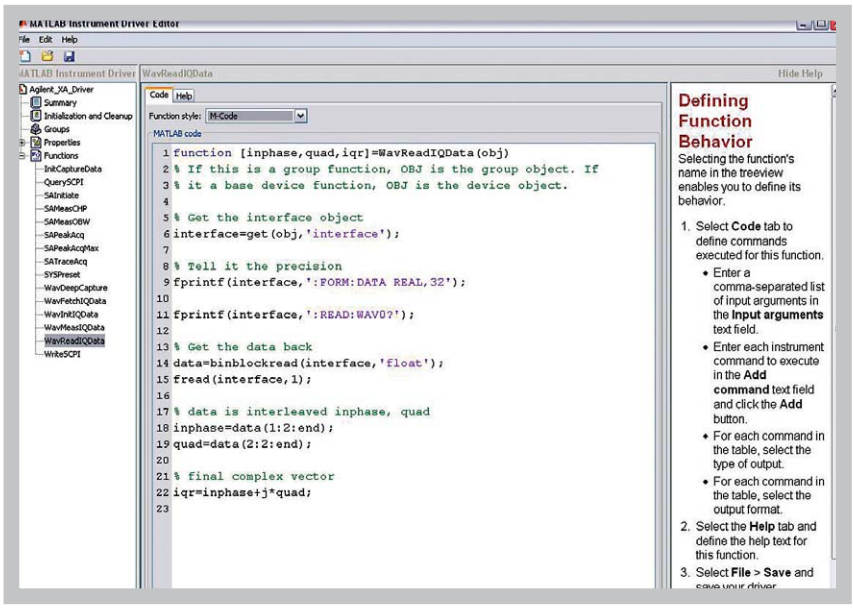


*Figure 5. **"WavReadIQData"** driver function*

The **':READ:WAV0?'** is sent to the analyzer to obtain the unprocessed I/Q trace data as a series of trace point values in volts. The data is read back from the analyzer in binary form using the **binblockread()** function (line 14).

It is very important to note that the **binblockread()** function in MATLAB does not read the terminating character from the output buffer of the instrument. Hence, any subsequent data read from the instrument will appear incorrectly as the terminating character will try to be interpreted as data. The **fread()** function (line 15) solves this issue by reading the terminating character from the buffer. Please see the MATLAB help documentation for more details about the **binblockread()** or **fread()** function.

As in the **Properties** section, you can also create any help documentation under the **Help** tab. For more help with creating MATLAB drivers, please see the **Product Help** under the **Help** menu in MATLAB.

2.3 Controlling instruments in MATLAB using device objects

As mentioned earlier, a device object allows MATLAB to communicate with an instrument via an industry-standard instrument driver, such as an IVI-COM, IVI-C, or a self-developed instrument driver. This section explores the use of the MATLAB instrument driver developed for the Agilent X-Series signal and spectrum analyzers. The instrument driver provides much easier access to I/O communication with the instrument.

The MATLAB instrument driver file (called **Agilent_SA_Driver.mdd** in this example) can be downloaded from one of the following Web sites: **www.agilent.com/find/n6171a_sa** or **www.mathworks.com/agilent**.

Note: This driver is a native MATLAB instrument driver and does not require an IVI-C or IVI-COM instrument driver to operate. You may choose to download or automatically create a MATLAB instrument driver wrapper that uses an IVI-COM or IVI-C driver, if the driver used in this example does not provide the functionality you need. Using MATLAB with an IVI-COM or IVI-C instrument driver is not described in this application note. More information on using MATLAB with IVI instrument drivers can be found at **www.mathworks.com/ivi**.

The driver file must be copied to the current working directory or to any other directory in the MATLAB path. To add a file or folder to the MATLAB search path, click on the **File** menu in MATLAB and select **Set Path**. A directory already exists where example drivers (shipped with the Instrument Control Toolbox) are already located. This directory is:

<matlabroot>\toolbox\instrument\instrument\drivers

There are four basic steps to follow when using a device object:

1) Create an interface object.
2) Use the **obj = icdevice('driver', hwobj)** constructor function to create a device object, where **hwobj** is the interface object you created in step one.
3) Use the **connect(obj)** command to connect the device object to the instrument (similar to the **fopen(mxa**) command for the interface objects.)
4) When finished, use the **disconnect(obj)** command to disconnect the device object from the instrument (similar to **fclose(mxa).)**

The following illustrates how to create a device object using a TCP/IP interface object.

1) Create the interface object.

```
mxa_ip = '10.10.10.10';
mxa_port = 5025;
mxa_if = tcpip(mxa_ip,mxa_port);
```

2) Create the device object.

```
mxa = icdevice('Agilent_SA_Driver.mdd', mxa_if);
```

3) Connect the device object to the instrument.

```
connect(mxa);
```

4) When finished disconnect the device object.

```
disconnect(mxa);
```

It is also a good idea to delete the interface object and clear the device object from the memory by using the **delete()** and **clear()** command.

```
delete(mxa_if);
clear(mxa);
```

2.3.1 An example MATLAB program using device objects

The following code has the same functionality as the first MATLAB example program, but this program now uses the MATLAB instrument driver.

```
% Using the SA Instrument Driver
% Initial setup
    mxa_ip = '10.10.10.10';
    mxa_port = 5025;
    fprintf('\nConnecting to Instrument ...\n');
% MXA Interface creation and connection opening
    mxa_if = tcpip(mxa_ip,mxa_port);
    mxa = icdevice(Agilent_SA_Driver.mdd', mxa_if);
    connect(mxa,'object')
% Instrument identification
    idn = get(mxa,'Identify');
    fprintf('Hello from %s\n', idn);
% Set instrument mode to SA
    set(mxa, 'Mode', 'Spectrum Analysis');
% Set the center frequency to 1 GHz
    mxa.SAFreqCenter = 1e9;
% optionally, the center frequency parameter can be
% changed using set(): set(mxa, 'SAFreqCenter', 1e9);
% Set the span to 20 MHz
    mxa.SASpan = 20e6;
% Set the reference level to +10 dBm using set()
    set(mxa, 'SARefLevel', 10);
% Query the resolution bandwidth
    rbw = mxa.SARBW;
    fprintf('Resolution bandwidth: %d kHz\n', rbw/1e3);
% Query the sweep time using get()
    swp = get(mxa, 'SASweepTime');
    fprintf('Sweep time: %d ms\n', round(swp*1000));
% Close the XA connection and clean up
    fprintf('Disconnecting from Instrument ...\n');
    disconnect(mxa);
    clear mxa;
```

This program begins by defining the TCP/IP interface object and creating the device object **(mxa)** using the **icdevice()** constructor function. The **connect(mxa)** command is needed for I/O communication between the instrument driver and instrument. The **get()** function is used to query the instrument's identity. The driver sets or queries the instrument properties in two different ways: 1) using the **mxa.PropertyName** notation and 2) using the **set()** or **get()** command. The span of the instrument is set using the first method, and the reference level is set using the second method. With the **set()** command it is possible to change one or more parameters in a single code line as shown in the next command:

```
set(mxa, 'SARefLevel', -10, 'SAFreqCenter', 2e9);
```

In the end, the object should be disconnected from the instrument using **disconnect()**, before the final clean up procedure.

The basic **fprintf()** and **query()** functions can still be used to send the instrument commands if some required functionality in not included in the driver. For example, the reference level property of the instrument could have been set using the following SCPI command:

```
fprintf(mxa_if,':DISP:WIND:TRAC:Y:RLEV -10);
```

The available properties and corresponding values of the object driver can be known using the **get(mxa)** command. For the current status of the driver this command returns:

```
>> get(mxa)
Hello from Agilent Technologies,N9020A,US46220185,A.01.50
    ConfirmationFcn =
    DriverName = Agilent_SA_Driver.mdd
    DriverType = MATLAB interface object
    InstrumentModel = Agilent Technologies,N9020A,
    US46220185,A.01.50
    Interface = [1x1 tcpip]
    LogicalName = TCPIP-141.121.90.55
    Name = Signal and Spectrum Analyzer-Agilent_SA_Driver
    ObjectVisibility = on
    RsrcName =
    Status = open
    Tag =
    Timeout = 10
    Type = Signal and Spectrum Analyzer
    UserData = []
```

SIGNAL AND SPECTRUM ANALYZER specific properties:

ByteOrder = Swapped

Identify = Agilent Technologies,N9020A,US46220185,A.01.50

Mode = Spectrum Analysis

ModeOpts = "NFIG 219, PNOISE 14, SA 1, EDGEGSM 13, WCDMA 9, VSA89601 101, CDMA2K 10, VSA 100, BASIC 8, ADEMOD 234, WIMAXOFDMA 75, TDSCDMA 211"

OperationComplete = 1

SAAtten = 10

SAAttnAuto = on

SAAve = off

SAAveCoun = 100

SABlank = 1

SADate = May 12, 2008

SADet = Normal

SADetAuto = on

SAFreqCenter = 1.3255e+010

SAInput = RFport

SAPeakExc = 6

SAPeakThresh = -90

SAPreamp = off

SAPresetType = mode

SARBW = 3e+006

SARBWAuto = on

SARefLevel = 0

SARFCoup = ac

SAScaleDiv = 10

SAScaleType = log

SASpan = 2.649e+010

SAStartFreq = 1e+007

SAStopFreq = 2.65e+010

SASweepPoints = 1001

SASweepSingle = 1

SASweepTime = 0.0662667

SATime = 17h 18m 54s

SATitle = "Swept SA"

SATrigger = freerun

SAVBW = 3e+006

SAVBWAuto = on

SAYunits = dbm

WavAcquisitionTime =

WAVAver =

WavCurrentCapture =

WavFirstCapture =

WavHardAvg =

WavIFWidth =

WavLastCapture =

WavNextCapture =

WavQueryData =

WavRBW =

WavSampleRate =

WavTimeCapture =

WavTraceDisplay =

WavTriggerSource =

Changeable properties are initialized with default values upon the driver object creation with **icdevice()**. These default values can be displayed using **set(mxa)**:

>> set(mxa)

ConfirmationFcn: string -or- function handle -or- cell array

Name:

ObjectVisibility: [{on} | off]

Tag:

Timeout:

UserDatca:

SIGNAL AND SPECTRUM ANALYZER specific properties:

ByteOrder: [{Normal} | Swapped]

Identify:

Mode: [{Spectrum Analysis} | cdmaOne | NADC | PDC | Basic | W-CDMA | cdma2000 | GSM_EDGE | Phase Noise | CDMA_1xEVDO | WLAN | Noise Figure | VSA_Link | Measuring Reciever | Digital Demod]

ModeOpts:

OperationComplete:

SAAtten: [0.0 to 70.0]

SAAttnAuto: [{on} | off]

SAAve: [on | off]

SAAveCoun: [1.0 to 8192.0]

SABlank: [{Off} | On]

SADate:

SADet: [Average | Negative | Positive | {Normal} | Sample | RMS | QP]

SADetAuto: [{on} | off]

SAFreqCenter:

SAInput: [{AmpRef} | RFport]

SAPeakExc:

SAPeakThresh:

SAPreamp: [on | {off}]

SAPresetType: [factory | user | {mode}]

SARBW: [1.0 to 8000000.0]

SARBWAuto: [{on} | off]

SARefLevel:

SARFCoup: [{ac} | dc]

SAScaleDiv: [0.1 to 20.0]

SAScaleType: [lin | log]

SASpan:

SAStartFreq: [-1.0E8 to 2.7E10]

SAStopFreq: [-1.0E8 to 2.7E10]

SASweepPoints:

SASweepSingle: [{On} | Off]

SASweepTime:

SATime:

SATitle:

SATrigger: [{freerun} | video | line | external1 | external2

| RFburst]

SAVBW: [1.0 to 5.0E7]

SAVBWAuto: [{on} | off]

SAYunits: [{dbm} | dbmV | dbuV | V | W | dbma | dbua | dbuvm | dbuam | dbpt | dbg]

WavAcquisitionTime:

WAVAver: [{On} | Off]

WavCurrentCapture:

WavFirstCapture:

WavHardAvg:

WavIFWidth: [{Wide} | Narrow]

WavLastCapture:

WavNextCapture:

WavQueryData:

WavRBW:

WavSampleRate:

WavTimeCapture:

WavTraceDisplay: [{On} | Off]

WavTriggerSource: [{External_Front} | External_Rear | Frame | Video | Free_Run | Line | RF_Burst]

The {} indicate the default parameter setting and the valid range is also included for the numeric ones.

The command **propinfo(mxa, 'PropertyName')** provides more detailed information about each property:

>> propinfo(mxa,'SARBW')

ans =

Type: 'double'

Constraint: 'bounded'

ConstraintValue: [1 8000000]

DefaultValue: 3000000

ReadOnly: 'never'

InterfaceSpecific: 1

The **instrhelp()** function can also be used with instrument driver objects in order to get specific help on any property or function of the driver.

>> instrhelp(mxa,'SADet')

SADET [Average | Negative | Positive | {Normal} | Sample | RMS | QP]

Sets or queries the detector type

Besides controlling the instrument settings, the driver also includes more complex functions, called methods. The available methods for the driver can be recalled using:

>> methods(mxa)

Methods for class icdevice:

Contents	**display**	**inspect**	**isa**
openvar			
class	**end**	**instrcallback**	**isequal**
propinfo			
close	**eq**	**instrfind**	**isetfield selftest**
connect	**fieldnames**	**instrfindall**	**isvalid**
set			
ctranspose	**get**	**instrhelp**	**length**
size			
delete	**geterror**	**instrhwinfo**	**methods subsasgn**
devicereset	**horzcat**	**instrnotify**	**ne**
subsref			
disconnect	**icdevice**	**instrument**	**obj2mfile**
vertcat			
disp	**igetfield**	**invoke**	**open**

Driver-specific methods for class icdevice:

InitCaptureData	**SAMeasCHP**	**SAPeakAcqMax**	**WavDeepCapture**
WavMeasIQData			
QuerySCPI	**SAMeasOBW**	**SATraceAcq**	**WavFetchIQData**
WavReadIQData			
SAInitiate	**SAPeakAcq**	**SYSPreset**	**WavInitIQData**
WriteSCPI			

Help on any of these methods can be obtained using again **instrhelp()**, for example:

>> instrhelp(mxa, 'SATraceAcq')

> **SATraceAcq method gets the swept SA trace data**
>
> **Iutput parameters: none**
>
> **Output parameters: one column vector with trace data**

The MATLAB instrument driver can be modified to include more properties and functions using the MATLAB Instrument Driver Editor. For more information on using device objects, please refer to the MATLAB product help.

2.3.2 Getting IQ data

Getting IQ data is one of the most frequently performed operations on this type of signal analyzer. Most of the demodulation takes place at the baseband IQ level. The procedure is similar to that of getting trace data, but with the instrument in IQ Analyzer (or Basic) mode. **Acquire IQ Vector.m** demonstrates how IQ can be acquired and displayed in a complex vector plot.

```
% Getting IQ data using the SA driver and plot display
% SOURCE SETUP...QPSK signal, @ 1 GHz carrier, 5 Msps,
& Gausian % filter

% TCPIP parameters
mxa_ip = '10.10.10.10';
mxa_port = 5025;

% MXA Interface creation and connection opening
mxa_if = tcpip(mxa_ip,mxa_port);
mxa = icdevice('Agilent_SA_Driver.mdd', mxa_if);
connect(mxa)

%% Measurement Setup
set(mxa,'Mode','Basic')
invoke(mxa, 'WriteSCPI', '*RST')

% if using VISA object switch Byte Order
set(mxa, 'ByteOrder', 'Swapped')

set(mxa,'SAFreqCenter',1000000000)

set(mxa,'SASweepSingle', 'Off')

set(mxa,'WavAcquisitionTime',.00007)

set(mxa,'WavRBW',8000000)

invoke(mxa, 'SAInitiate');

%% Get IQ data
iq = invoke(mxa,'WavReadIQData');
```

```
% Create a figure 1 and bring it to the front
figure(1)
% Vector plot (imag vs real)
plot(real(iq),imag(iq))

% Axis adjustment
axis square

% Labels
xlabel('I')
ylabel('Q')
title('IQ vector plot')
% Close the XA connection and clean up
disconnect(mxa);
clear mxa;
```

The resulting plot for QPSK modulation using baseband raised cosine filtering is displayed in Figure 6. The plot parameters may need to be edited for each specific setup.

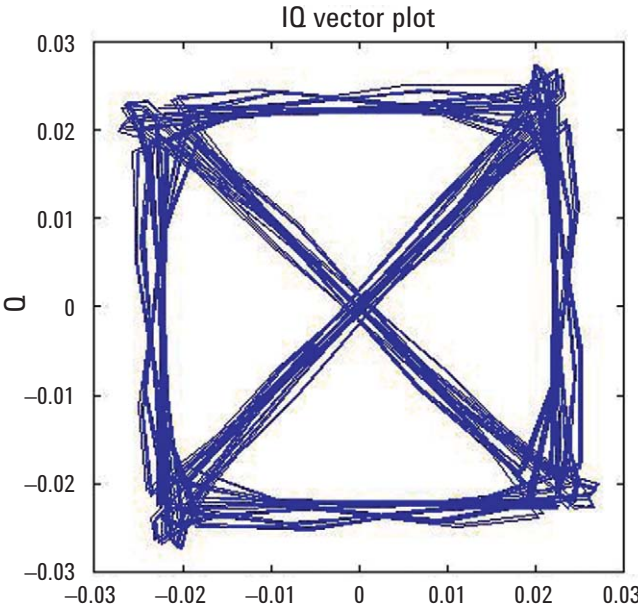


Figure 6. QPSK modulation plot using baseband raised cosine filtering

Using the instrument driver, the IQ data is transferred into MATLAB using the WavReadIQData function. The details of the WavReadIQData function can be found by looking at the driver code using the midedit command. The following code is the sequence of commands that retrieves the IQ data.

```
function [inphase,quad,iqr]=WavReadIQData(obj)
% If this is a group function, OBJ is the group object. If
% it a base device function, OBJ is the device object.
% Get the interface object
interface=get(obj,'interface');

% Tell it the precision
fprintf(interface,':FORM:DATA REAL,32');

fprintf(interface,':READ:WAV0?');

% Get the data back
data=binblockread(interface,'float');
fread(interface,1);

% data is interleaved inphase, quad
inphase=data(1:2:end);
quad=data(2:2:end);

% final complex vector
iqr=inphase+j*quad;
```

IQ arrays may be very long. The 'InputBufferSize' property may need to be increased as required. The default buffer size is $2e^6$ and this may be suitable in most cases. If the buffer size needs to be increased, this can be accomplished by editing the "Create" code in the driver or by editing the interface object property using either of the following commands:

```
set(mxa_if, 'InputBufferSize', new_buffer_value); or
mxa_if.InputBufferSize = new_buffer_value;
```

It is important to remember that interface parameters such as buffer size or time out need to be set prior to connecting the device object to the instrument using the connect() command.

2.3.3 TOI measurement example

Two-tone third order intermodulation (TOI) is an important measurement used to characterize amplifiers and mixers. The program TOI Automation.m in the appendix shows how to implement a complete TOI measurement capable of producing the surface plot shown in Figure 7 (a 3-D color intensity plot.) This program demonstrates how to control a signal generator in conjunction with the X-Series signal and spectrum analyzers and the powerful plotting features of MATLAB.

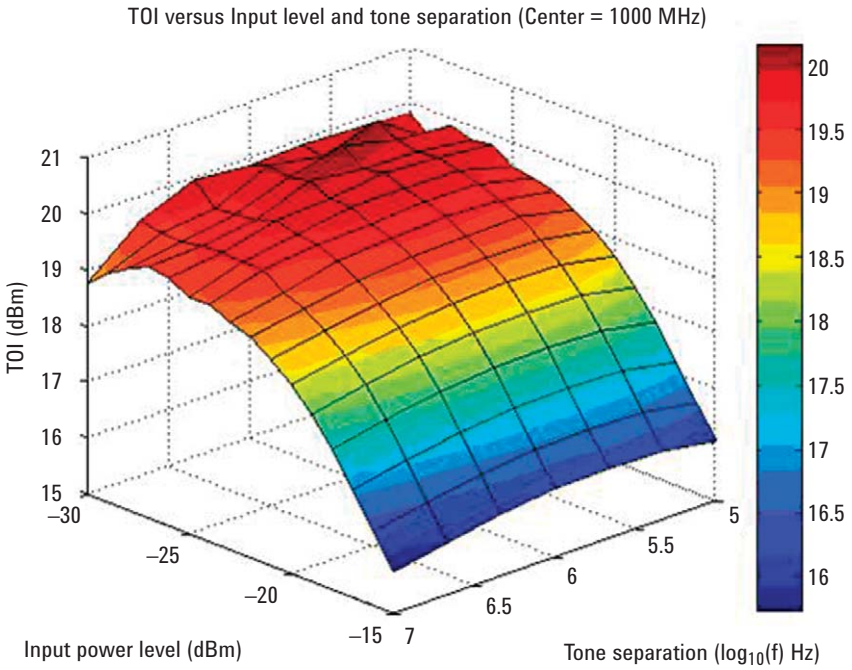


Figure 7. Surface plot

A signal generator with multitone capability and a LAN interface, such as Agilent's ESG or MXG signal generator, is used to produce the two-tone signal required for this measurement. The measurement parameters are specified in the initial portion of the code. These parameters include:

- Center frequency of the measurement
- Power level range
- Tone separation range
- Peak criteria
- Number of trace averages per measurement

Next, the program enters the measurement cycle, sweeping the input power level for each tone separation. This is a faster combination than sweeping tone separation for each power level since changing tone separation in the generator takes a relatively longer time.

The actual measurement routine is based on the peak table functionality of the X-Series signal and spectrum analyzers. Only one SPCI command is required to retrieve the detected peaks. The peaks are then processed and the relevant ones for the TOI measurement are identified. The TOI computation follows the following rules:

TOI_left = [min(y1,y2)-y3]/2 + y1

TOI_right = [min(y1,y2)-y4]/2 + y2

where y1 and y2 are the main tone levels, y3 is the lower frequency third order product and y4 is the higher frequency third order product. In the event a measurement is invalid (no peak is detected), a value of –999.0 is returned.

3.0 Creating, Modifying, and Executing X-Series and PSA Applications using MATLAB Software

While explaining the specific steps to develop entire X-Series and PSA applications with a GUI is outside the scope of this paper, there are already several example programs with a GUI developed by Agilent that will run from inside, or remote to, the X-Series signal and spectrum analyzers. The following example programs (analog demodulation and advanced data visualization) are complete applications as they use the front panel keys of the instrument to control the program.

Those desiring to create their own GUI-based applications for the signal or spectrum analyzers are encouraged to download an existing example from **www.agilent.com/find/n6171a** or **www.mathworks.com/agilent**, and modify those applications as needed to suit your specific testing needs. Developing your own applications is one of the key benefits of having MATLAB installed or interfaced to your X-Series or PSA signal or spectrum analyzer.

3.1 Analog signal demodulation example

Analog demodulation is a baseband IQ processing operation frequently used to test analog communications. This example shows how to process IQ data captured from a modulated signal (in this case using FM.) The program continuously acquires IQ data and displays it on the top plot of shown in Figure 8. For this program, the trace update is controlled by a timer. The AM and FM demodulation is accomplished using simple math operations and the resulting signals displayed on the bottom plots. The operations for each modulation type are:

- AM:abs(iq) / mean(abs(iq)) -1
- FM:diff(unwrap(angle(iq)))

The demodulation takes place inside the **update_plot()** function.

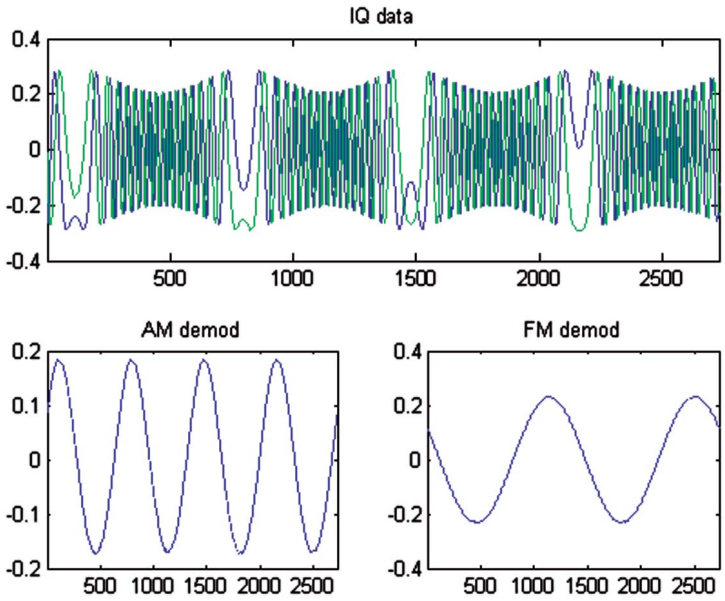


Figure 8. Surface plot

Figure 8 shows a more complete use of the MATLAB GUI capabilities. This application is driven from the front panel keyboard of the instrument, using the group of soft keys. Simultaneously, it can also be driven by a mouse device. GUI objects in MATLAB are identified by handles. This particular code (**MXA GUI Example.m**) uses the following objects and respective handles (in parentheses) and is created in the GUI layout section:

- Figure **(fh)**
- Axes **(ah)**
- Plot **(ph)**
- Text objects for the frequency, span, and reference level (**freq_text_h, span_text_h, ref_text_h**)
- Four-button objects for the soft keys (**softkeyh1, softkeyh2, softkeyh3, softkeyh7**)

The properties of these objects are configured by the **set()** command, in a similar fashion as that used for the instrument object. Two relevant properties for the figure object are the **KeyPressFcn** and **DefaultUicontrolKeyPressFcn**. These properties define the function to be called upon by any front panel key press when the figure has the mouse focus (the first property) and when any of the figure objects have the mouse focus (the pushbuttons in this case.) Key press events are processed by the **keypress()** function. It traps the name of the pressed key and any modifier like SHIFT or CONTROL. For instance, the first soft key produces the sequence Shift+Control+F2; in this event, the function associated to this button is called **cont_button_callback()**. Also, the [FREQ], [SPAN], and [AMPTD] hard keys are processed and when any of these are pressed, the corresponding text object is highlighted by the green color. A screenshot of this application example is shown in the Figure 9.

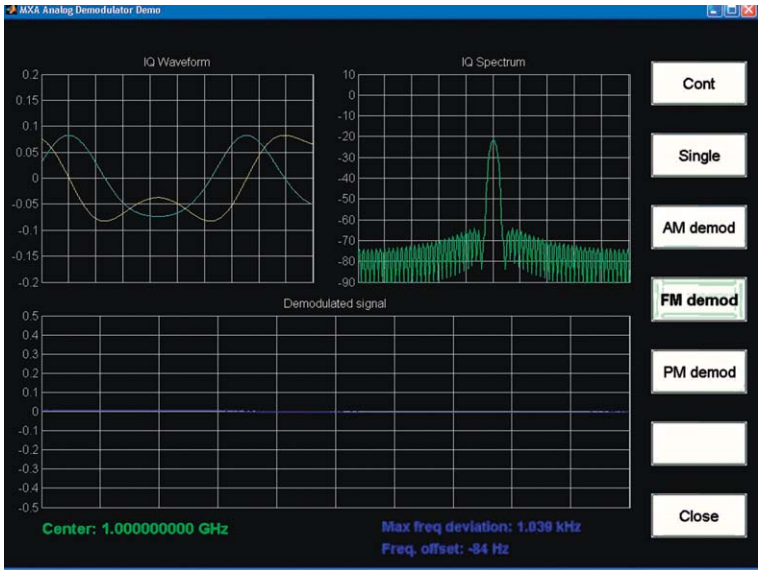


Figure 9. Example of program using MATLAB's GUI capabilities

3.2 Advanced data visualization example

As seen in the TOI example, MATLAB provides powerful plotting tools for graphing data. Using MATLAB and an X-Series signal and spectrum analyzer can provide a way to transfer measurement data and plot it in a dynamically. The advanced data visualization uses MATLAB to constantly acquire trace data and display it in four unique ways. This program provides a GUI to allow the user to select which mode to use.

The Analog Advanced mode provides a color-graded persistence display as shown in Figure 10. A signal will change from blue to red the longer a signal remains at a given frequency. This mode stores 200 traces in a first-in, first-out (FIFO) buffer and then processes the traces to create the color-graded display.

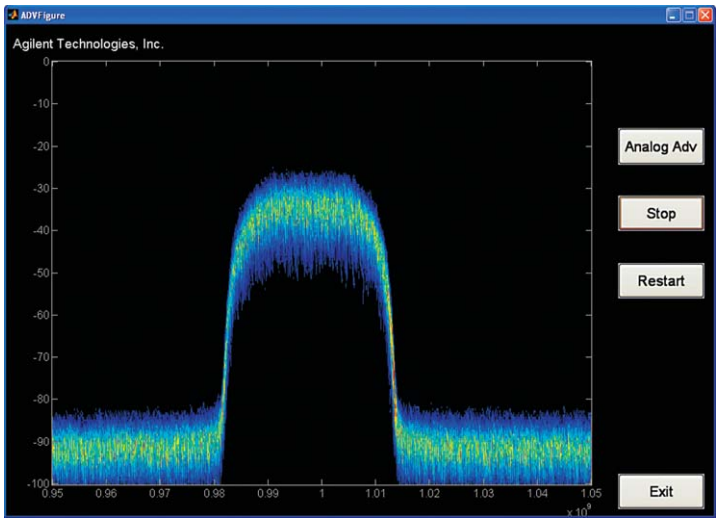


Figure 10. A color-graded persistence display provided by Analog Advanced mode

The Analog Plus mode is identical to the Analog Advanced mode, except that it is not color graded and the measurement points are not connected with lines. (See Figure 11.)

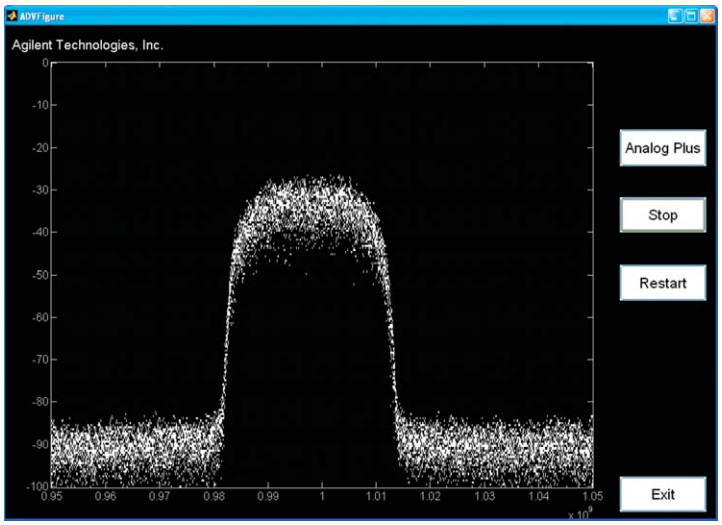


Figure 11. Plot generated by Analog Plus mode

The Waterfall mode displays individual spectra in a three dimensional format, with the most recent trace displayed in the front. A total of 50 traces are displayed at a time.

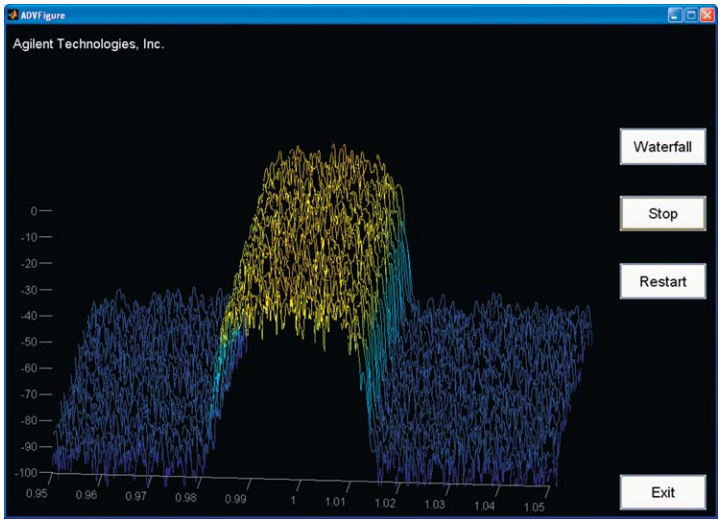


Figure 12. Waterfall mode's individual spectra plot in a three dimension

The Spectrogram mode displays a standard color-graded spectrogram (refer to Figure 13.)

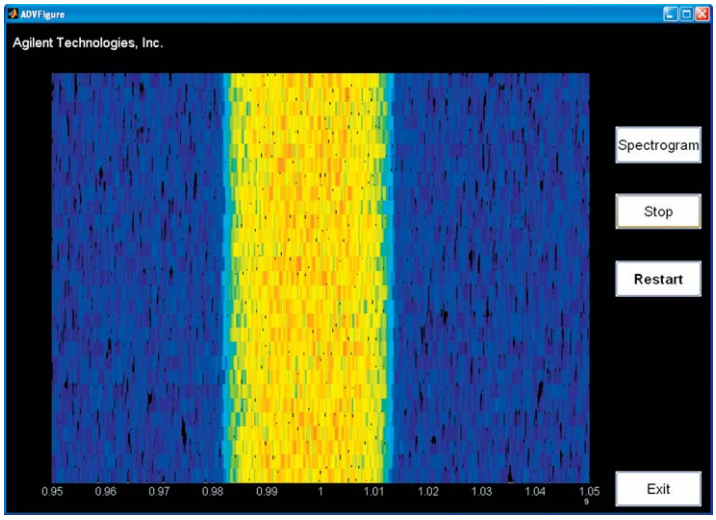


Figure 13. Color-graded spectrogram generated in Spectorgram mode

3.3 Tune and listen example

Similar to the advanced data visualization program, the tune and listen program uses a GUI to control the program. Tune and listen uses several built-in MATLAB functions, such as **angle()**, **unwrap()**, and **wavplay()**, allowing the user to capture and demodulate an AM or FM signal. Once the data has been transferred to MATLAB, it is demodulated and then re-sampled to play the demodulated signal through a computer's sound card. The GUI allows the user to set the frequency to acquire the demodulation bandwidth, the demodulation time, and power range of the signal (see Figure 14.)

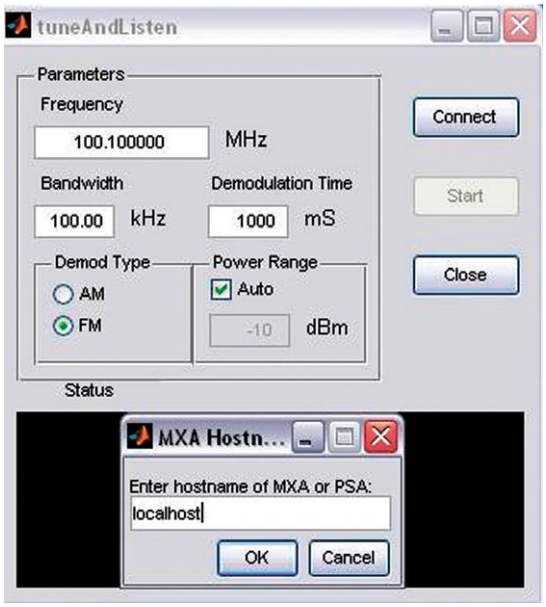


Figure 14. MATLABS's GUI used for tune and listen program

4.0 Summary

The Agilent X-Series signal and spectrum analyzers are built using the Microsoft Windows operating system. Integrating this operating system into Agilent instruments enables users to interface their instruments to application software using industry-standard connectivity (LAN, USB, GPIB), and provides the ability to develop and modify custom applications to execute on Agilent signal and spectrum analyzers. MATLAB is a well respected software environment used to make measurements, analyze and visualize data, generate arbitrary waveforms, control instruments, execute modulation schemes, and build test systems. Agilent is now able to sell MATLAB software to enable users of Agilent signal and spectrum analyzers to create, modify, and execute their own applications for specific testing needs.

Appendix – Additional Examples

The MATLAB code used for many of the examples described in this application note is provided below. Most of this code can also be downloaded from **www.agilent.com/find/n6171a**.

1. Basic instrument control using SCPI

```
% A example program using basic SCPI commands

% TCPIP parameters of the analyzer
mxa_ip = '10.10.10.10';
mxa_port = 5025;

% MXA Interface creation and connection opening
fprintf('\nConnecting to XA ...\n');
mxa = tcpip(mxa_ip,mxa_port);
fopen(mxa);

% Intrument identification
idn = query(mxa,'*IDN?');
fprintf('Hello from %s', idn);

% Set the center frequency to 1 GHz
fprintf(mxa,':FREQ:CENT 1 GHz');

% Set the span to 20 MHz
fprintf(mxa,':FREQ:SPAN 20 MHz');

% Set the reference level to +10 dBm
fprintf(mxa,':DISP:WIND:TRAC:Y:RLEV 10');

% Query the resolution bandwidth using fprinf()/fgets()
fprintf(mxa,':BAND:RES?');
rbw = str2double(fgets(mxa));
fprintf('Resolution bandwidth: %d kHz\n', rbw/1e3);

% Query the resolution bandwidth using fprinf()/fgets()
fprintf(mxa,':BAND:RES?');
rbw = str2double(fgets(mxa));

fprintf('Resolution bandwidth: %d kHz\n', rbw/1e3);
% Query the sweep time using query()
swp = str2double(query(mxa,':SWE:TIME?'));
fprintf('Sweep time: %d ms\n', round(swp*1000));

% Close the XA  connection and clean up
fprintf('Disconnecting from XA ...\n');
fclose(mxa);
clear; % clear from the workspace
```

2. Basic instrument control using the MATLAB instrument driver

```
% Using the MXA Instrument Driver

% Initial setup
mxa_ip = '10.10.10.10';
mxa_port = 5025;
fprintf('\nConnecting to MXA ...\n');

% MXA Interface creation and connection opening
mxa_if = tcpip(mxa_ip,mxa_port);
mxa = icdevice(Agilent_SA_Driver.mdd', mxa_if);
connect(mxa,'object')

% Intrument identification
idn = get(mxa,'Identify');
fprintf('Hello from %s\n', idn);

% Set the center frequency to 1 GHz
mxa.SAFreqCenter = 1e9;

% optionally, the center frequency parameter can be
% changed using set(): set(mxa, 'SAFreqCenter', 1e9);

% Set the span to 20 MHz
mxa.SASpan = 20e6;

% Set the reference level to +10 dBm using set()
set(mxa, 'SARefLevel', 10);

% Query the resolution bandwidth
rbw = mxa.SARBW;
fprintf('Resolution bandwidth: %d kHz\n', rbw/1e3);

% Query the sweep time using get()
swp = get(mxa, 'SASweepTime');
fprintf('Sweep time: %d ms\n', round(swp*1000));

% Close the MXA connection and clean up
fprintf('Disconnecting from MXA ...\n');
disconnect(mxa);
clear mxa;
```

3. Acquire single trace

```
% Single trace aquisition
oldobjs=instrfind;
if ~isempty(oldobjs)
disp('Cleaning up ...')
delete(oldobjs);
clear oldobjs;
end

% Initial setup
mxa_ip = '10.10.10.10';
mxa_port = 5025;
mxa=tcpip(mxa_ip, 5025);

% input buffer size to receive trace data
% should be at least 4 times the number of trace
% points for 32-bit real format
set(mxa,'InputBufferSize',4005);

% instrument response timeout
set(mxa,'Timeout',5);
fopen(mxa);

% Set the data trace format to REAL, 32 bits
fprintf(mxa,':FORM:DATA REAL,32');

% Get the nr of trace points
nr_points = str2double(query(mxa,':SWE:POIN?'));

% Get the reference level
ref_lev = str2num(query(mxa,':DISP:WIND:TRAC:Y:RLEV?'));
% Get the trace data
fprintf(mxa,':INIT:IMM;*WAI'); % start a sweep and
%wait until it completes
fprintf(mxa,':TRAC? TRACE1');
data = binblockread(mxa,'float32'); % get the trace data
fscanf(mxa); %removes the terminator character

% create and bring to front figure number 1
figure(1)
% Plot trace data vs sweep point index
plot(1:nr_points,data)

% Adjust the x limits to the nr of points
% and the y limits for 100 dB of dynamic range
xlim([1 nr_points])
ylim([ref_lev-100 ref_lev])

% activate the grid lines
grid on
title('Swept SA trace')
xlabel('Point index')
ylabel('Amplitude (dBm)')

% Disconnect an clean up
fclose(mxa);
delete(mxa);
clear;
```

4. Acquire trace continuous async

```
% Continuous trace aquisition (async)

oldobjs=instrfind;
if ~isempty(oldobjs)
disp('Cleaning up ...')
delete(oldobjs);
clear oldobjs;
end

% Initial setup
mxa_ip = '10.10.10.10';
mxa_port = 5025;
mxa=tcpip(mxa_ip, 5025);
set(mxa,'InputBufferSize',100000);
set(mxa,'Timeout',5);
fopen(mxa);

% Set the data trace format to REAL, 32 bits
fprintf(mxa,':FORM:DATA REAL,32');

% Get the nr of trace points
nr_points = str2double(query(mxa,':SWE:POIN?'));

% Get the reference level
ref_lev = str2num(query(mxa,':DISP:WIND:TRAC:Y:RLEV?'));

% Put the instrument in continuos mode
fprintf(mxa,':INIT:CONT ON');

% Create and bring to front figure number 1
figure(1)

% Create a plot handle, ph, and draw a line at the reflevel
ph = plot(1:nr_points,ref_lev*ones(1,nr_points));

% Adjust the x limits to the nr of points
% and the y limits for 100 dB of dynamic range
xlim([1 nr_points])
ylim([ref_lev-100 ref_lev])

% Activate the grid
grid on

% Plot cycle
for i=1:100
fprintf(mxa,':TRAC? TRACE1');
data = binblockread(mxa,'float32');
fscanf(mxa); %removes the terminator character

% Change the plot line data (fast update method)
set(ph,'Ydata',data);

% flushes the plot event queue
drawnow
end

% Disconnect an clean up
fclose(mxa);
delete(mxa);
clear;
```

5. Acquire trace timer

```
% Continuous aquisition with timer
oldobjs=instrfind;
if ~isempty(oldobjs)
disp('Cleaning up ...')
delete(oldobjs);
clear oldobjs;
end

% Initial setup
mxa_ip = '141.121.92.157';
mxa=tcpip(mxa_ip, 5025);
set(mxa,'InputBufferSize',30000);
set(mxa,'Timeout',5);
fopen(mxa);

% Set the data trace format to REAL, 32 bits
fprintf(mxa,':FORM:DATA REAL,32');

% Get the nr of trace points
nr_points = str2double(query(mxa,':SWE:POIN?'));

% Get the reference level
ref_lev = str2num(query(mxa,'DISP:WIND:TRAC:Y:RLEV?'));

% Put the instrument in continuos mode
fprintf(mxa,':INIT:CONT ON');

% create and bring to front figure number 1
figure(1)
ph = plot(1:nr_points,ref_lev*ones(1,nr_points));

% Adjust the x limits to the nr of points
% and the y limits for 100 dB of dynamic range
xlim([1 nr_points])
ylim([ref_lev-100 ref_lev])
grid on
th =timer('timerfcn',@update_plot,...
'ExecutionMode','FixedRate',...
'Period',0.1);
start(th)
pause(10)
stop(th)

% Disconnect an clean up
fclose(mxa);
delete(mxa);
clear mxa;
function update_plot(varargin)

% Get the trace data
fprintf(mxa,'TRAC? TRACE1');
data = binblockread(mxa,'float32');
fscanf(mxa); %removes the terminator character

% Plot trace data vs sweep point index
set(ph,'Ydata',data);
drawnow
end
end
```

6. Acquire IQ vector

```
% Getting IQ data using the XA driver and plot display

% TCPIP parameters
mxa_ip = '10.10.10.10';
mxa_port = 5025;

% MXA Interface creation and connection opening
mxa_if = tcpip(mxa_ip,mxa_port);
mxa = icdevice('Agilent_SA_Driver.mdd', mxa_if);
connect(mxa)

%% Get IQ data
%% Measurement Setup
set(mxa,'Mode','Basic')
invoke(mxa, 'WriteSCPI', '*RST')

% if using VISA object switch Byte Order
set(mxa, 'ByteOrder', 'Swapped')

set(mxa,'SAFreqCenter',1000000000)

set(mxa,'SASweepSingle', 'Off')

set(mxa,'WavAcquisitionTime',.00007)

set(mxa,'WavRBW',8000000)

invoke(mxa, 'SAInitiate');

%% Get IQ data
iq = invoke(mxa,'WavReadIQData');

% Create a figure 1 and bring it to the front
figure(1)
% Vector plot (imag vs real)
plot(real(iq),imag(iq))

% Axis adjustment
axis square

% Labels
xlabel('I')
ylabel('Q')
title('IQ vector plot')

% Close the MXA connection and clean up
disconnect(mxa);
clear mxa;
```

7. TOI automation

```
function example8
% TOI automated measurement and surface plot
% Version: 1.0
% Date: Sep 11, 2006
% 2006 Agilent Technologies, Inc.
% Clean up any unclosed instrument object
oldobjs=instrfind;
if ~isempty(oldobjs)
disp('Cleaning up ...')
delete(oldobjs);
clear oldobjs;
end

% TOI measurement parameters
f_cent = 1e9; % center frequency
pow = -30:1:-15; % input power range
sep = logspace(5,7,7); % tone separation range
pk_threshold = -80; % peak_threshold criterium
pk_excursion = 6; % peak_excursion criterium
num_ave = 4; % number of trace averages for eachmeasurement

n_sep = length(sep);
TOI = zeros(n_sep,n_pow,2);

% Initial setup
nimitz = 'nimitzcpu142.soco.agilent.com';
baker = 'baker.soco.agilent.com';
socodhcpe28 = 'socodhcpe28.soco.agilent.com';
randyesg = '141.121.88.234';
eric_mxg = '141.121.92.32';
mxa = tcpip(nimitz, 5025);
esg = tcpip(randyesg, 5025);
disp(' ')
disp('Connecting to MXA/ESG ...');
set(mxa,'InputBufferSize',2000);
set(mxa,'Timeout',10);
fopen(mxa)

fopen(esg)

% MXA initial setup
fprintf(mxa,[':FREQ:CENT ' num2str(f_cent)]);
fprintf(mxa,[':FREQ:SPAN ' num2str(5*sep(1))]);
fprintf(mxa,'DISP:WIND:TRAC:Y:RLEV 10');
fprintf(mxa,'INIT:IMM;*OPC?');
fscanf(mxa);
fprintf(mxa,'FORM:DATA REAL,32');
fprintf(mxa,['CALC:MARK:PEAK:THR ' num2str(pk_threshold)]);
fprintf(mxa,['CALC:MARK:PEAK:EXC ' num2str(pk_excursion)]);
fprintf(mxa,'CALC:MARK:PEAK:TABLE:STAT ON');
fprintf(mxa,['AVER:COUNT ' num2str(num_ave)]);
fprintf(mxa,'TRAC:TYPE AVER');
fprintf(mxa,'INIT:IMM;*OPC?');
fscanf(mxa);
```

```
% ESG/MXG initial setup
fprintf(esg,[':FREQ ' num2str(f_cent)]);
fprintf(esg,':POW -20');
fprintf(esg,':RADIO:MTONE:ARB:SETUP:TABLE:NTONES 2');
fprintf(esg,':RADIO:MTONE:ARB:SETUP:TABLE:FSPACING 1e5');
fprintf(esg,':RADIO:MTONE:ARB on');
fprintf(esg,':OUTPUT on');
fprintf(esg,':OUTPUT:MOD on');
fprintf('\nStarting toi measurement:');
fprintf('\nPower sweep range: %d dBm to %d dBm in %d dB steps',
min(pow), max(pow), pow(2)-pow(1));
fprintf(mxa,':INIT:CONT OFF');
for x = 1:n_sep
fprintf(esg,[':RADIO:MTONE:ARB:SETUP:TABLE:FSPACING '
num2str(sep(x))]);
fprintf(esg,'*OPC?');
fscanf(esg);
fprintf(mxa,[':FREQ:SPAN ' num2str(5*sep(x))]);
fprintf(mxa,'*OPC?');
fscanf(mxa);
fprintf(['\nFrequency separation: ' num2str(round(sep(x)/1e3))
' kHz ']);
for y = 1:n_pow
fprintf(esg,[':POW ' num2str(pow(y))]');
fprintf(esg,'*OPC?');
fscanf(esg);
toi = toi_meas(mxa,pk_threshold,pk_excursion);
TOI(x,y,1) = toi(1);
TOI(x,y,2) = toi(2);
fprintf('.')
end

%pause(4)
end
fprintf(mxa,'CALC:MARK:PEAK:TABLE:STAT OFF');
fprintf(mxa,':INIT:CONT ON');

% Surface plot
fprintf('\nPlotting results ...\n')
surf(log10(sep),pow,((TOI(:,:,1)+TOI(:,:,2))/2)',...
'FaceColor','interp',...
'EdgeColor','k',...
'Linewidth',.2) % plot the average of toi lower and right
colormap(jet(256))
xlabel('Tone Separation (log_{10}(f) Hz)')
ylabel('Input Power Leval (dBm)')
zlabel('TOI (dBm)')
title(['TOI vs Input Level and Tone Separation (Center = '
num2str(f_cent/1e6) ' MHz)'])
colorbar

%Disconnect an clean up
disp(' ')
disp('Disconnecting from MXA/ESG ...')
fclose(mxa)
fclose(esg)
delete([mxa esg]);
clear mxa esg

function toi = toi_meas(mxa,pk_thr,pk_exc)
```

```
fprintf(mxa,':INIT:IMM;*OPC?');
fscanf(mxa);
fprintf(mxa,[':CALC:DATA1:PEAK? ' num2str(pk_thr) ','
num2str(pk_exc) ',AMPL,ALL']);
peak_data = binblockread(mxa,'float32');
peak_num = peak_data(1);
peak_amp = peak_data(2:2:end);
peak_freq = peak_data(3:2:end);
toi = [-999.0 -999.0];
if peak_num <= 2 % no 3rd order products found
return
end
if peak_freq(1) < peak_freq(2)
f1 = peak_freq(1); y1 = peak_amp(1);
f2 = peak_freq(2); y2 = peak_amp(2);
else
f1 = peak_freq(2); y1 = peak_amp(2);
f2 = peak_freq(1); y2 = peak_amp(1);
end
peak_sep = f2-f1;
f3_idx = find(abs(peak_freq - (f1 - peak_sep)) <
peak_sep/25);
if ~isempty(f3_idx)
f3 = peak_freq(f3_idx(1)); y3 = peak_amp(f3_idx(1));
toi(1) = (min(y1,y2)-y3)/2+y1;
end
f4_idx = find(abs(peak_freq - (f2 + peak_sep)) <
peak_sep/25);
if ~isempty(f4_idx)
f4 = peak_freq(f4_idx(1)); y4 = peak_amp(f4_idx(1));
toi(2) = (min(y1,y2)-y4)/2+y2;
end
end
end
```

For More Information

To learn more about using MATLAB software with your EXA, MXA, and PSA signal and spectrum analyzers, as well to download free reference applications to execute on these instruments, visit **www.agilent.com/find/n6171a**.

www.agilent.com
www.agilent.com/find/PSA
www.agilent.com/find/xseries

Agilent Email Updates

www.agilent.com/find/emailupdates
Get the latest information on the products and applications you select.

www.lxistandard.org
LXI is the LAN-based successor to GPIB, providing faster, more efficient connectivity. Agilent is a founding member of the LXI consortium.

Remove all doubt

Our repair and calibration services will get your equipment back to you, performing like new, when promised. You will get full value out of your Agilent equipment throughout its lifetime. Your equipment will be serviced by Agilent-trained technicians using the latest factory calibration procedures, automated repair diagnostics and genuine parts. You will always have the utmost confidence in your measurements. For information regarding self maintenance of this product, please contact your Agilent office.

Agilent offers a wide range of additional expert test and measurement services for your equipment, including initial start-up assistance, onsite education and training, as well as design, system integration, and project management.

For more information on repair and calibration services, go to:

www.agilent.com/find/removealldoubt

For more information on Agilent Technologies' products, applications or services, please contact your local Agilent office. The complete list is available at:

www.agilent.com/find/contactus

Americas

Canada	(877) 894-4414
Latin America	305 269 7500
United States	(800) 829-4444

Asia Pacific

Australia	1 800 629 485
China	800 810 0189
Hong Kong	800 938 693
India	1 800 112 929
Japan	0120 (421) 345
Korea	080 769 0800
Malaysia	1 800 888 848
Singapore	1 800 375 8100
Taiwan	0800 047 866
Thailand	1 800 226 008

Europe & Middle East

Austria	01 36027 71571
Belgium	32 (0) 2 404 93 40
Denmark	45 70 13 15 15
Finland	358 (0) 10 855 2100
France	0825 010 700*
	*0.125 €/minute
Germany	07031 464 6333
Ireland	1890 924 204
Israel	972-3-9288-504/544
Italy	39 02 92 60 8484
Netherlands	31 (0) 20 547 2111
Spain	34 (91) 631 3300
Sweden	0200-88 22 55
Switzerland	0800 80 53 53
United Kingdom	44 (0) 118 9276201

Other European Countries:
www.agilent.com/find/contactus

Revised: July 2, 2009

Product specifications and descriptions in this document subject to change without notice.

Printed in USA, September 3, 2009
5989-9377EN

Agilent Technologies