ANEXO A

Objetivos

En este anexo se presentan de forma completa los requisitos y objetivos establecidos para el autor en el proyecto FCAT

Propuesta de Prácticas

El documento que puede leerse en este apartado contiene de forma íntegra y sin modificaciones la propuesta realizada por el IGN para el puesto de prácticas desempeñado el autor durante la realización del presente proyecto final de carrera.

Institute of Geodesy and Navigation • University FAF Munich • D-85577 Neubiberg
institut@ifen.bauv.unibw-muenchen.de • +49-89-6004-3425 (phone) • +49-89-6004-3019 (fax)

Diploma Thesis: Development of Signal Compatibility Tool

The aim of the thesis will be to write a small high-quality Matlab-based software application with uses of best practices of sustainable software development. Within the framework of the work the student will concentrate on the UML-described design securing modularity, well-defined internal structure and interfaces that combine the specific scripts developed by team of specialists into the final application. A GUI should be designed and implemented according to users' needs. The test plan should be included and performed to secure quality of the final product.

Requirements:

- Studies of computer science or telecommunications.
- Basic experience with Matlab.
- Basic knowledge in physics of signals and electromagnetic waves.
- Very good knowledge of C++, Java, Matlab or other higher languages.
- Knowledge of software design and engineering.
- Fluent English knowledge in speech and writing.
- Team and communication skills.
- Individual initiative.

Detailed Contents:

The target of the thesis to develop a Matlab based software tool able to compute all the relevant parameters which are necessary to perform the radio-frequency compatibility analysis between global navigation satellite systems (GNSS) in all the current and potential future frequency bands. In particular the compatibility metrics (CM) that the tool should analyze are:

- Link budget for the maximum and minimum received power levels
- Effective carrier-to-noise density ratio
- Carrier-to-noise density degradation
- Inter-system spreading codes cross-correlation characteristics

The first part will be the specification and analysis of the tool as well as the preliminary design of the graphical user interface. The rules to be followed during the software development by all participants of the project should be defined in details.

The second part will be implementation of the core of the tool and integration of the specific scripts developed by other members of the project team. The code should be clearly structured in independent modules connected through well-defined interfaces. In this way subsequent modifications and updates of the code should be supported.

The module of user interface part should comprise all functionalities: graphical user interface, loading and saving of input files and plotting and saving of results. A general template for the modules for the compatibility metrics should be designed. The other members of the team will then implement each

metrics compliant to this template. These metrics modules should communicate with the user interface module through a set of strictly defined functions implementing computation, plotting into GUI and saving of the CM to an output file. Also the CM modules will communicate with each other over given interface functions. A separated module will be formed by auxiliary functions available without limitation in any part of the tool.

The GUI of the tool will be based on so called scenario projects. A project is one complete input setting that can be created, saved and reopen for each simulated scenario. Several template projects will be provided for typical cases according to needs of EC.

The GUI main window will clearly structured into input and output part and will avail direct comparison of computed results with respective input. The input part will contain all input variables sorted into panels according to context. The widget of individual variables will offer tooltips with short description of the meaning, units and range of the variable.

The third part of the thesis will be the testing and verification of the developed software following the best practices of high quality software development.

Lista de Objetivos

Phase 1: Till 1st meeting in Brusel on 16th June
Target: Fullfilled
GasipReduced Delivery: 31st July
Target: Complete SW, .exe and Matlab, complete UserGuide
Phase2: Till end of August, delivery of SW and teleconference
Complete tool, Complete User Guide - Programmer and User
Target: Part

Load and Save input data

Loading and Saving of InputVars of EffDegCn0
Loading and Saving of InputVars of PrnCodes
Loading of Systems and Signals config files

Varible checking

Numerical variables
Files: exists, suffix

Special consistency checks and other checks: error info to user, inform M+D about usage

Gui Design and Implementation

Add Setting to Menu->Settings to set if panels in input panels system are exclusive
Maybe other options to this menu
First prototype for Automatic Creation in RunTime with save

Preview of systems

Way for preview of systems
Finalise preview of systems
Solution for lower resolution,
clean table in case of simple constellation

Preview of signals

Way for preview of signals, implementation
Name of signal as headline to the window

Preview and Edit SSC

Way for preview and editing of SSC values, implementation

ALT SYSTEM: popupmenu with all systems from vSystems, that have at least one signal in vSignal, which refers to them. And the system of the DES signal is excluded
DES signal: popupmenu with all signals
Prototype for Automatic Creation using special program
Way to represent two apps in one in GUI
Decision for the RunTime/Speicial program design solution
Gui design of tab panels - vertical tabs

check compatability for Matlab 2009

Output panels

Output panel design and implementation

Options

Save FoM and Show FoM in Separate Window

Application structure design

Structure of Modules: EffDegCn0 vs. PrnCodes

Gui structure: data structures and functions

EffDegCn0, PrnCodes Gui structure: data structures and functions

Implementation of the directory structure

Implementation of sInputVar full contents - with reading of files: antenna pattern etc.

Documentation

Automatic generation of .m file documenting structInputVars and structFom1-N

Style for modeling of app's structure

Development procurement methodology for the project

Deployment of Scrum

Write GuiFomInterfaceStructure file - documentation purposes

Gasip limited version

generate GUI for limited version with the GUI framework

representation of substructures and respective change to parser

code for subpanels

Deliver setting for the chosen signals of Galileo

Connect the computational functionality

synchronise Gasip full and reduced version (variable names: eg.VarNameFlag convention)

generate executable

generate executable with complete runtime in .msi

ANEXO B

Manuales de FACIL

A continuación se presentan los manuales de uso del *framework* FACIL

Manual de usuario

Institute of Geodesy and Navigation
Institut für Erdmessung und Navigation

Universidad Zaragoza
1542

FACIL

Forget About Complicated Interface Layouts

Manual de usuario

Autor: Eduard Porta Martín-Moreno

Versión: 1.0

Última Revisión: 30 de Agosto de 2011

Tabla de Contenido

Inicio de la Aplicación

Al iniciar la aplicación se presenta al usuario con un dialogo de carga que informa del progreso y la etapa en la que se encuentra (Figura 1).

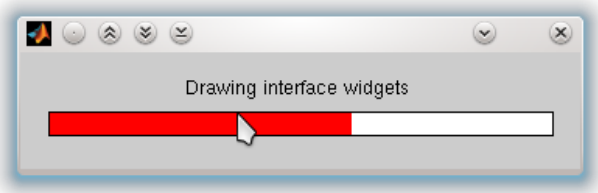

Figura 1 - Diálogo de Carga

Una vez finalizado el proceso de carga, se mostrará una interfaz, cuyo aspecto puede variar, y que en general será muy similar a la Figura 2.

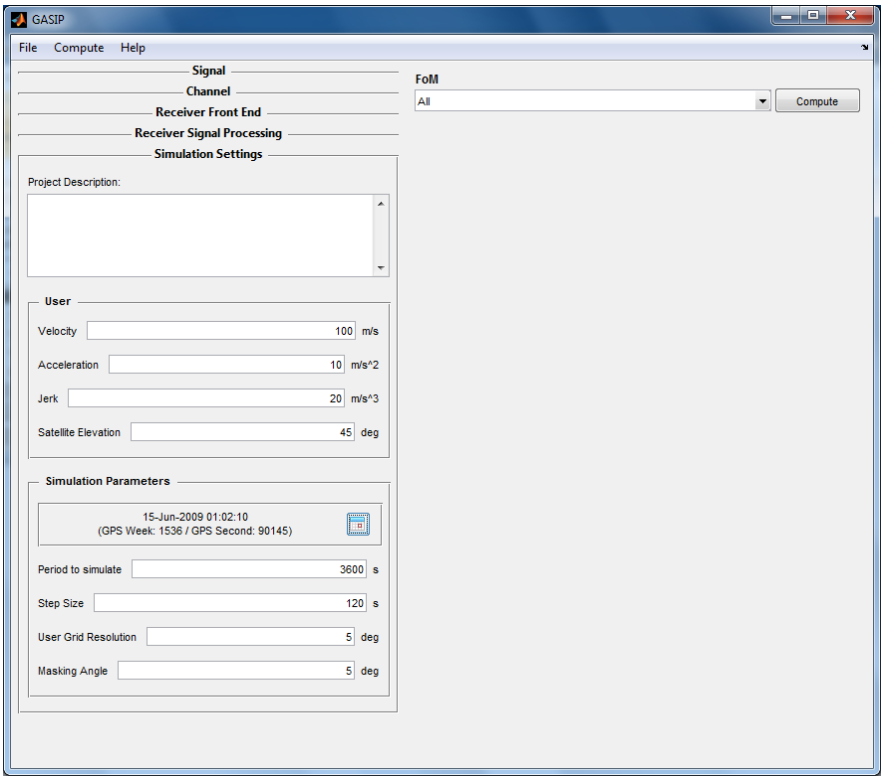

Figura 2 - Interfaz Gráfica

ADVERTENCIA Si se ejecuta la aplicación desde una sesión de MATLAB, es indispensable no cambiar de directorio en ningún momento para asegurar el correcto funcionamiento de la misma.

Proyectos

La aplicación permite guardar y cargar datos en ficheros de proyectos. Puede accederse a estas funciones a través del menú Archivo (Figura 3):

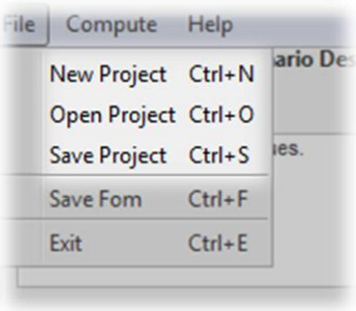

Figura 3 - Menú Archivo: Opciones de Proyectos

- Nuevo Proyecto: Configura el panel de entrada con valores por defecto. En algunas versiones de la aplicación[16] puede permitir al usuario elegir entre una lista de plantillas.
- Abrir proyecto: Carga el panel de entrada con datos que el usuario haya guardado previamente.
- Guardar proyecto: Guarda en un nuevo archivo los datos del panel de entrada.

Los archivos de proyectos emplean un formato XML práctico y sencillo de editar con un editor de texto común (Figura 4).

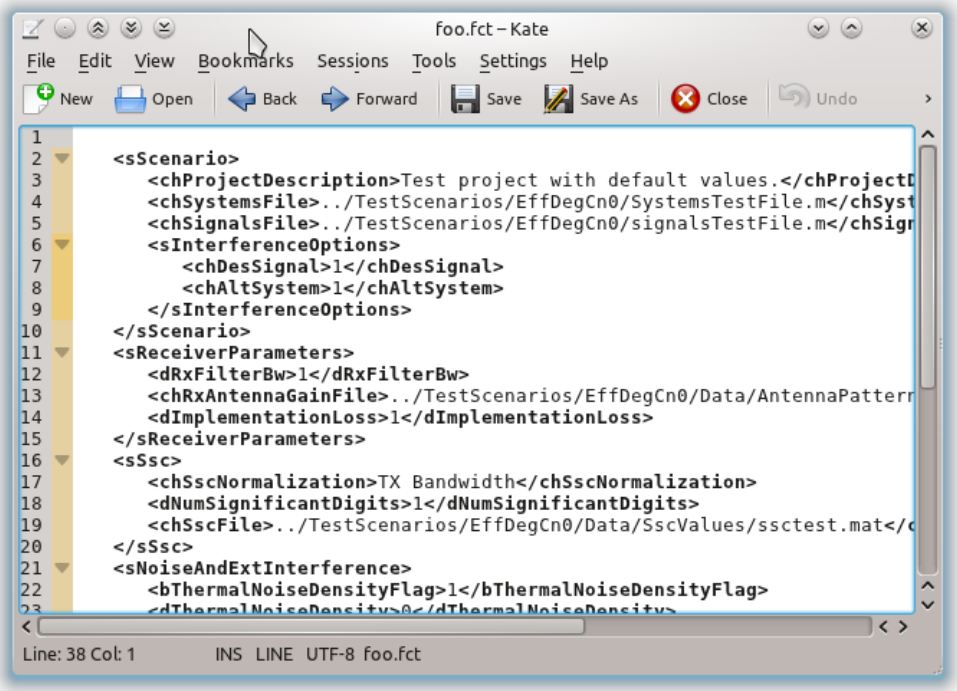

Figura 4 - Editando un Fichero de Proyecto

[16] Esto dependerá de las decisions tomadas por el desarrollador.

El Panel de Entrada

El panel de entrada (Figura 5) de la aplicación se compone de una serie de pestañas. Dichas pestañas contienen controles que permiten modificar los valores de las variables de entrada. Es posible que cuando exista más de una pestaña y dependiendo de la decisión del desarrollador, la apertura de un panel provoque el cierre de los restantes, de forma que en todo momento quede abierto únicamente el panel activo.

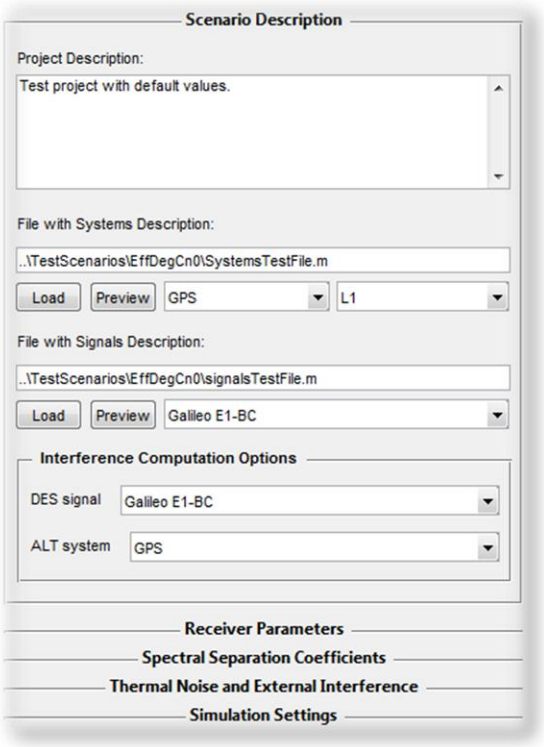

Figura 5 - Ejemplo de Panel de Entrada

Algunos de los controles pueden tener mecanismos de validación (Figura 6). Si los datos de alguno de los controles no son válidos, éste lo indicará mediante algún mecanismo visual como un diálogo y/o un cambio de color, y el botón de cálculo será deshabilitado hasta que el error sea subsanado.

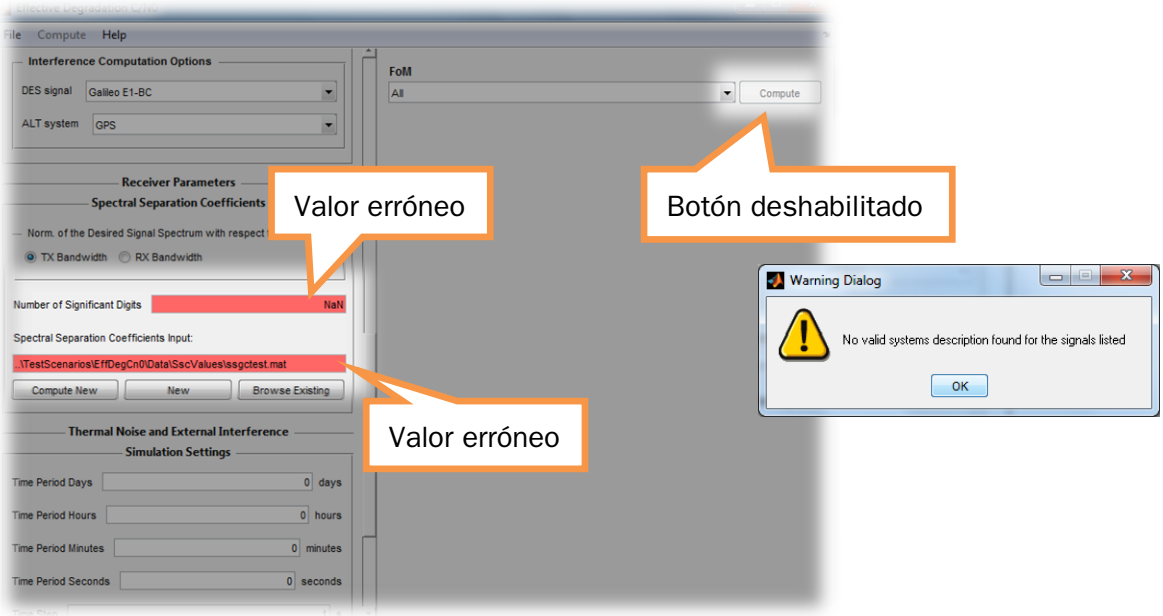

Figura 6 - Mecanismos de validación y notificación al usuario

Cálculo de Resultados

Si todos los campos del panel de entrada presentan datos válidos, el botón y el menú de cálculo estarán activos (nótese que su aplicación podría no presentar el menú en caso de existir una única función de cálculo). Para seleccionar la función a calcular emplearemos la lista o listas desplegables del panel de salida y pulsaremos el botón calcular, o bien seleccionaremos la función deseada en el menú calcular (Figura 7).

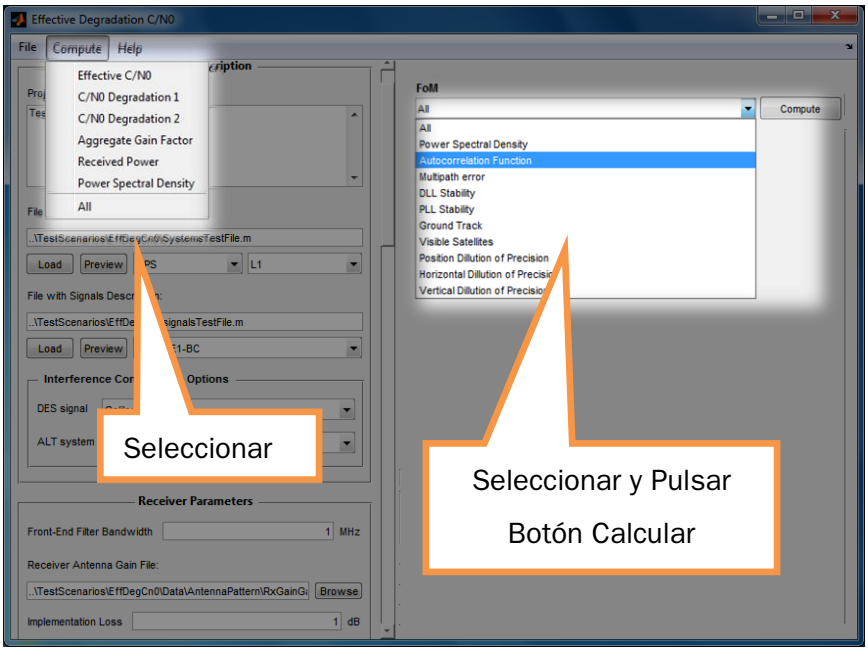

Figura 7 - Menú Calcular y desplegable junto al Botón Calcular

Si el algoritmo de cálculo es costoso en tiempo es probable que aparezca una barra de progreso como la de la Figura 8. Si pulsamos sobre el botón rojo con una cruz de dicho diálogo se cancelará el proceso de cálculo.

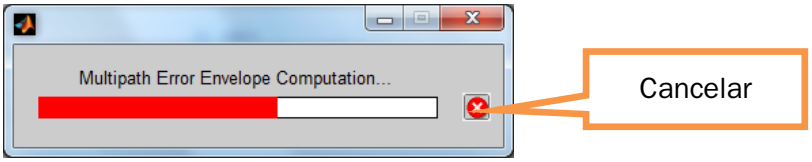

Figura 8 – Diálogo de Progreso del Cálculo con Botón de Cancelación

Una vez se complete el proceso de cálculo con éxito se mostrarán los resultados en el panel de salida. Dependiendo del algoritmo ejecutado es posible que se muestre un panel con opciones para controlar la visualización (Figura 9).

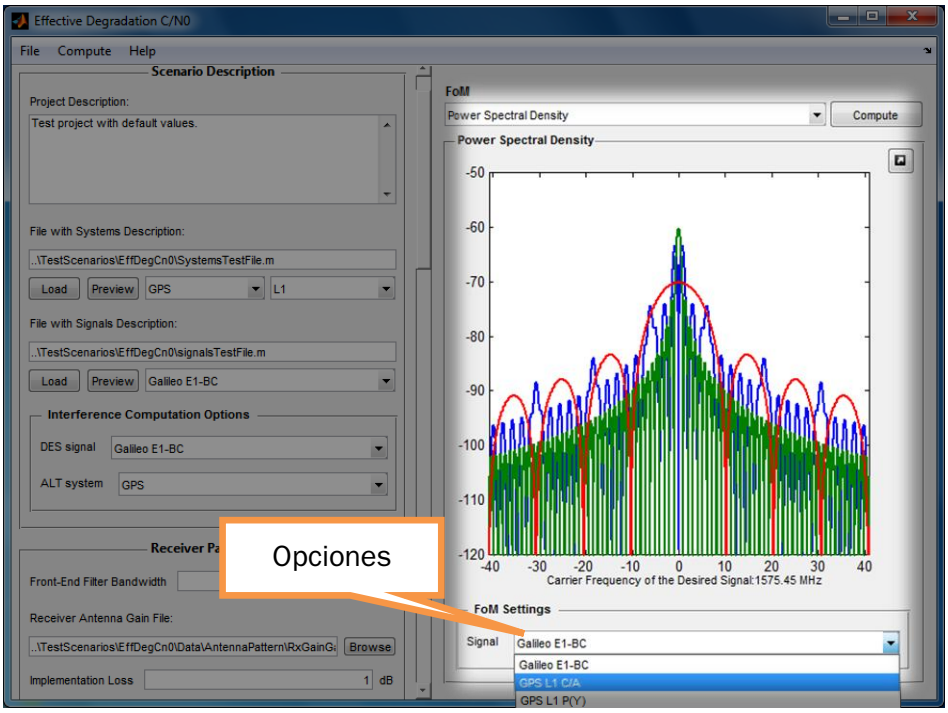

Figura 9 - Panel de salida Mostrando el Resultado y Opciones de Visualización

Existe también la opción de mostrar los resultados en una vista separada, para ello se pulsará sobre el botón 'Ver en ventana separada', disponible en la esquina superior derecha del panel de salida (Figura 10).

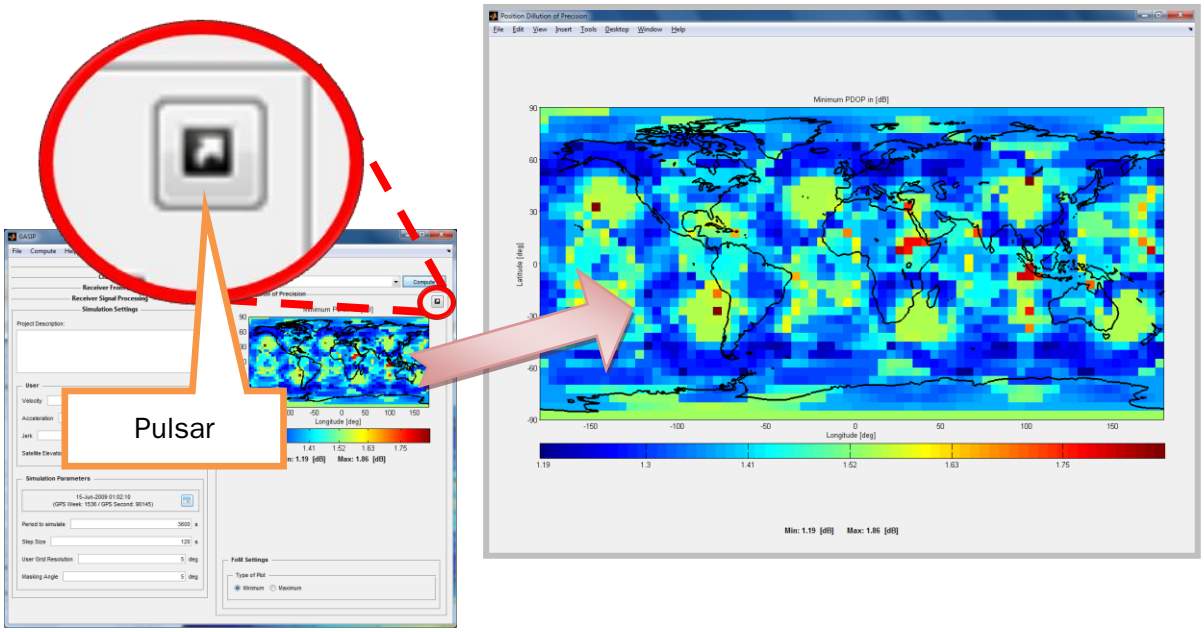

Figura 10 - Visualización de Resultados en Ventana Separada

Para guardar los resultados en un archivo emplearemos la opción disponible en el menú archivo (Figura 11). Los formatos disponibles son:

- **Fig**: Almacena una copia del resultado en pantalla en formato figura, el cual puede ser abierto desde MATLAB.
- **Mat**: Todos los valores calculados (y no su representación) se almacenan en un fichero de datos en formato MAT, dichos valores pueden ser cargados de nuevo en MATLAB para trabajar con ellos.
- **Csv**: Almacena todos los valores calculados en un fichero de valores separados por comas que puede ser abierto con un editor de texto simple o con una aplicación de hojas de cálculo.
- **Imagen (tiff, png, jpg, bmp, ppm, eps)**: Guarda como imagen una captura de la representación de los resultados mostrada.

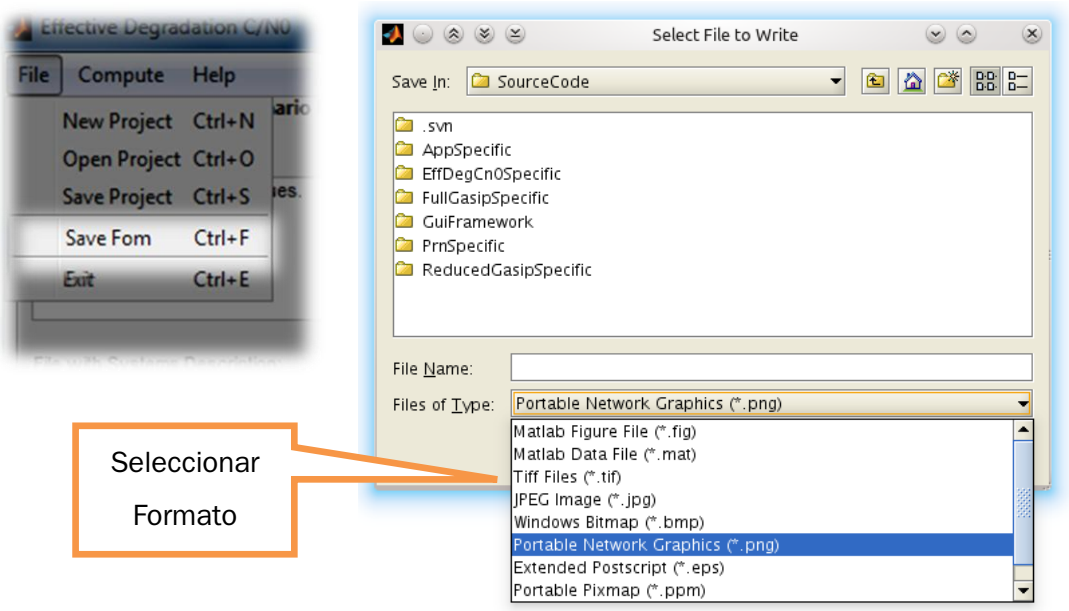

Figura 11 - Menú y Diálogo para Guardar los Resultados

Manual del programador

Institute of Geodesy and Navigation
Institut für Erdmessung und Navigation

Universidad Zaragoza
1542

FACIL

Forget About Complicated Interface Layouts

Manual del Programador

Autor: Eduard Porta Martín-Moreno

Versión: 1.0

Última Revisión: 30 de Agosto de 2011

Tabla de Contenido

Implementación de una Aplicación Básica

Con FACIL es posible implementar una aplicación completa con tan solo definir tres elementos: las variables de entrada, la descripción de las funciones de cálculo y las interfaces con las funciones de cálculo.

En primer lugar suponiendo que la aplicación vaya a llamarse 'MyApp' crearemos una estructura de directorios para nuestra nueva aplicación como muestra la Figura 1. Otra opción es copiar el directorio AppSpecific y la función `startApp.m` incluidas en los ficheros de ejemplo de FACIL y modificar estos siguiendo esta guía.

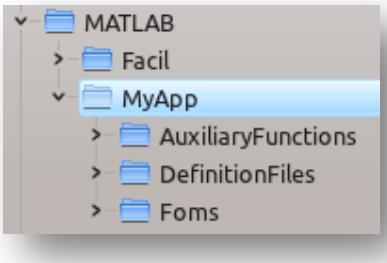

Figura 1 - Directorios Base

Es necesaria una función que lance la aplicación, para ello crearemos un archivo llamado `startMyApp.m` con el siguiente contenido:

```
1   function [] = startMyApp( )

2   %clean the working space of Matlab
3   matlabrc; clear all; clc; close all;

4   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5   %Change this value to the application specific directory
6   appSpecificDir='MyApp';
7   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

8   %add Facil to the path variable
9   addpath(strcat(cd, [filesep 'Facil']));

10  %start the graphical user interface
11  mainGui(appSpecificDir);

12  end
```

También es necesario crear un fichero con configuración específica de la aplicación llamado configureApplication.m en el directorio AuxiliaryFunctions de la carpeta de la aplicación, con el siguiente contenido:

```
function [ sConfig ] = configureApplication ()

% Extensión para proyectos de la aplicación
sConfig.chInputVarsFileExtension = '*.app';

% Descripción para proyectos de la aplicación
sConfig.chInputVarsFileDescription = 'Application Input File';

% Título de la ventana de la aplicación
sConfig.chApplicationTitle = 'My Application';

% Pestañas del panel de entrada funcionan de modo exclusivo (exclusive) o es
posible abrir más de uno a la vez (multiple)
sConfig.TypeOfInputTabs = 'multiple';

end
```

Definición de las Variables de Entrada

Las variables de entrada deben ser definidas en un fichero llamado `InputVarsDescription.txt` dentro del directorio `DefinitionFiles` de la aplicación.

A continuación se descompone la sintaxis de dicho fichero:

1. La cabecera: Se compone de la etiqueta Input Parameters, el nombre de la aplicación, una descripción de ésta, una línea de longitud arbitraria de asteriscos y la declaración de la estructura base, la cual siempre debe aparecer:

```
16 Input Parameters: My Application [Descripcion de MyApp]
17 ****************************************************************************
18 sInputVars: Estructura con las variables de entrada
19 {
```

2. Una o más declaraciones de estructuras conteniendo las variables de entrada. Se declara entre corchetes el título que la pestaña o panel generado por la estructura tendrá en la interfaz. La estructura a su vez puede albergar más estructuras y/o definiciones de variables escalares.

```
20   sTest [Titulo para la Pestaña]: Descripción de lo que contiene
21   {
           <...>
22   }
```

3. Una o más definiciones de variables escalares, especificando entre corchetes las unidades de la variable, el tipo, el rango o contenido válido, configuraciones específicas del GUI y La etiqueta a mostrar en la interfaz gráfica. Además se usa el comentario sobre la variable como comentario emergente o tooltip en la interfaz. Para una especificación completa de las distintas posibilidades puede consultarse la documentación de la función `getWidgetDefinitions`.

```
23 chVariable1 [unitless, string, any, , Variable Ejemplo]: Comentario Emergente
```

4. La llave de cierre de la estructura base y otra línea de asteriscos de longitud arbitraria.

Descripción de las Funciones de Cálculo

La descripción de las funciones de cálculo se realiza en un fichero muy similar al anterior llamado `FomsDescription.txt` que debe crearse también entro del directorio `DefinitionFiles` de la aplicación.

El fichero se descompone como sigue:

5. La cabecera: Se compone de la etiqueta 'LIST OF FOMS', el nombre de la aplicación, una descripción de ésta y una línea de longitud arbitraria de asteriscos:

```
24 LIST OF FOMS: My Application [Descripcion de MyApp]
25 ************************************************************************
```

6. Una o más definiciones de funciones compuestas por los nombres corto y largo de la misma, un entero especificando el número de opciones y, de ser distinto de cero, declaraciones de parámetros con el mismo formato que las variables escalares del fichero anterior.

```
26 Short Name of the Fom: 'TestFunction'
27 Complete Name: 'Test function for sample application'
28 Options: 0
```

7. Otra línea de asteriscos de longitud arbitraria.

Las Interfaces con las Funciones de Cálculo

Con FACIL la interacción con las funciones de cálculo se realiza en dos pasos, el cálculo propiamente dicho y la representación. Si hemos generado los ficheros anteriores correctamente, al ejecutar nuestro `startMyApp` deberíamos tener una interfaz gráfica completa, con los widgets correspondientes a las variables que hemos definido en el panel de entrada, y la lista de funciones de cálculo en el panel de salida y menú calcular.

Cálculo

Cuando tratemos de ejecutar una de las funciones de cálculo, FACIL esperará encontrar en algún subdirectorio de la carpeta `MyApp` (preferiblemente en Foms), una función denominada `get<NombreCorto>`, donde `<NombreCorto>` designa el nombre corto definido para la función en el punto anterior. En caso de no encontrar la función se buscará una llamada `getFoms`, que de existir, debería calcular todas las funciones de la aplicación.

Tanto la función específica como la general deberían aceptar como único argumento de entrada una estructura que respondería a la descrita en el fichero de descripción de variables de entrada. Como salida deben devolver un valor booleano indicando si la operación ha sido cancelada por el usuario o abortada por un error y una estructura, que en caso de las funciones específicas contendrá los datos calculados y en el caso de la función general una estructura un nivel superior agrupando las anteriores.

Representación

La representación de una función de cálculo se realizará cuando dicha función esté seleccionada en el panel de salida y haya datos disponibles para la misma. Esto puede suceder porque acabemos de calcularla o porque ya lo hicimos en un momento anterior y la hemos vuelto a seleccionar en la lista desplegable.

Para este paso FACIL buscará una función con el nombre `plot<NombreCorto>`, que debe aceptar como argumentos tres handles, correspondientes respectivamente a los ejes, campo de texto y tabla del panel de salida, una estructura con los datos calculados previamente y tantos argumentos adicionales como opciones se hayan especificado para la función.

Esta función será la encargada de emplear estos handles para representar los datos, teniendo en cuenta:

1. La función no debería tratar de controlar el tamaño y posición de los elementos.

2. Los elementos no usados deberían ocultarse con `set(hElemento,'Visible','off')`.

3. El objeto axes tiene dos propiedades adicionales. La propiedad `PlotType` acepta los valores '`normal`' y '`earth`'. La propiedad `AspRatio` controla la relación de aspecto de la gráfica, y, de no fijarse, será `[1 1]` para graficas normales y `[2 1]` cuando `PlotType` sea '`earth`'.

4. El uso del objeto uitable correspondiente al elemento tabla puede ser consultado en la documentación de MATLAB.

Funcionalidad Avanzada

Gran parte de la funcionalidad avanzada de FACIL reside en el hecho de que es posible implementar complementos para algunos tipos de widget, y lo que es más importante, es posible implementar nuestros propios widgets.

Para ello se empleará la herencia de objetos, partiendo habitualmente de las clases plugin y widget o alguno de sus descendientes. Toda la documentación necesaria acerca de sus propiedades, métodos y ejemplos de uso pueden ser consultados en el código fuente de FACIL.

ANEXO C

Detalles de Implementación

Este anexo se compone de documentos que extienden o explican algunos detalles de la implementación de FACIL y las aplicaciones que se han desarrollado con el mismo.

Estudio sobre el uso de constantes y variables globales

A lo largo de la experiencia del autor como alumno de ingeniería siempre se le ha desaconsejado el uso de variables globales por ser esta una de las peores fuentes de acoplamiento y deteriorar la modularidad de la aplicación.

Sin embargo para este proyecto resultó prácticamente imposible evitar el uso de globales en los casos en los que la propia naturaleza del problema lo requería, como pueden ser las constantes de cálculo científico o el bus de comunicación entre widgets. Para estos casos se han evaluado las posibles alternativas e implementado aquellas menos perjudiciales o más eficientes.

Constantes

Principalmente, las constantes globales se han utilizado para disponer de constantes científicas usadas en los algoritmos de cálculo. En primer lugar se elaboró una lista de posibilidades:

- La alternativa actual, un fichero que define una variable global de tipo estructura. Las constantes se almacenan en sus campos y pueden ser obtenidas haciendo visible la variable global y accediendo a dichos campos.
- Elaborar un script que crea e inicializa todas las constantes, ejecutar dicho script al inicio de cualquier función que use estas constantes. Este sistema se consideró inseguro por la posibilidad de introducir código arbitrario en este archivo que pudiese alterar la estabilidad de la aplicación.
- Escribir una función para cada constante, de este modo son de solo lectura y, al ser funciones, la introducción de código defectuoso solo afecta a la ejecución de la misma y el valor devuelto. Esta alternativa es usada por MATLAB, pero se descartó por el excesivo coste debido al elevado número de constantes.

Tras el estudio se concluyo que era mejor no modificar el tratamiento de constantes globales, pues el coste invertido sería mayor que el beneficio obtenido, debido a que no se eliminaría ninguna dependencia o acoplamiento.

Variables

La introducción de FACIL hizo que se plantease esta cuestión que no había surgido anteriormente. Según la documentación de MATLAB un programa con variables globales puede presentar problemas para generar un ejecutable o que dicho ejecutable funcione correctamente.

El principal problema en este caso fue encontrar un modo de almacenar información persistente (para el bus, los resultados de los cálculos, algunos widgets), que no precisara del uso de una variable global real. La solución al problema consistió en usar los datos de aplicación. Los datos de aplicación consisten en un mecanismo de MATLAB que permite acceder a una única variable relacionada con la figura (ventana) actual, desde cualquiera de sus objetos hijo. Dado que esta funcionalidad era necesaria para los widgets, siempre se disponía de un handle apropiado, con lo que únicamente fue necesario emplear dicha variable como una estructura, almacenando cada dato en un campo diferente.

Estudio sobre los diferentes sistemas de OOP en Matlab

MATLAB, a diferencia de otros lenguajes de programación no dispone de un framework unificado para la programación orientada a objetos. En la versión del cliente, MATLAB 2007a se documenta la implementación de clases mediante funciones que utilizan en su cuerpo una sentencia especial “class”. Sin embargo esta funcionalidad resulta insuficiente pues las posibilidades de herencia y abstracción son altamente limitadas.

En principio en dicha versión del software no existe forma alguna de heredar a partir de elementos uicontrol o uipanel. Tampoco es posible usar la sentencia classdef como en versiones más modernas en lugar de usar una función constructor. En resumen, MATLAB dispone, a lo largo de sus versiones, de una colección de métodos de implementar programación orientada a objetos, pero todavía en las últimas versiones no parece haber un estándar consensuado sobre la forma de hacerlo.

ANEXO D

Optimización del Cálculo de la Correlación Cruzada de Códigos PRN

Este anexo recopila las diferentes versiones del bucle trabajado así como los resultados experimentales obtenidos

Código del Bucle Original

```
1   % Initialization of variables
2   vCorrelationMagnitudeLog = [-100 (-80:0.5:1)];
3   vCorrelationMagnitudeNat = 0:1:dTotLength;
4   sCorrelationHistogramLog.mOdd = ...
        zeros(dNumDesCodes*dNumIntCodes, ...
        length(vCorrelationMagnitudeLog),dTotNumCombinations);
5   sCorrelationHistogramNat.vOdd =
        zeros(dTotNumCombinations,length(vCorrelationMagnitudeNat));
6   count = 0;

7   % Cycle computing the cross-correlation histogram
8   for i=1:dNumDesCodes
9      vFftDesCode = fft(mDesiredCodeExt(i,:));
10     for j=1:dNumIntCodes
11        count = count + 1;
12        for n = 1:dTotNumCombinations

13           vOddCorrelation = ...
                real(ifft(vFftDesCode .* ...
                conj(fft(squeeze(mInterferingCodeExt(j,:,n)) .* ...
                vDoppler))));

14           sCorrelationHistogramLog.mOdd(count,:,n) = ...
                hist(20*log10(abs(vOddCorrelation) / ...
                length(vOddCorrelation)),vCorrelationMagnitudeLog)...
                ./ length(vOddCorrelation);

15           sCorrelationHistogramNat.vOdd(n,:) = ...
                sCorrelationHistogramNat.vOdd(n,:) + ...
                hist(abs(vOddCorrelation),vCorrelationMagnitudeNat);

16        end
17     end
18  end
```

Código Vectorizado

```
% Clear unused variables
clear sInputVars mDesiredCode mInterferingCode dNumBits dNumComb ...
mBitCombinations dNumRows vTempBitComb ...
vTempShiftedBitComb dIndex vIndex mTempFlips mTempInterferingCode

% This packs variables in contiguous space
save('fjklq4fji4qof4jqiofpqf9q40fuq9fquf9q4uf90qFRDSGREGER54');
clear;
load('fjklq4fji4qof4jqiofpqf9q40fuq9fquf9q4uf90qFRDSGREGER54');
delete('fjklq4fji4qof4jqiofpqf9q40fuq9fquf9q4uf90qFRDSGREGER54.mat');

% Compute the cross-correlation histogram (vectorized version)

mOddCorrelation = reshape(shiftdim(real(ifft(bsxfun(@times, ...
    shiftdim(fft(mDesiredCodeExt,[],2),-1), ...
    reshape(conj(fft(bsxfun(@times, ...
    mInterferingCodeExt,vDoppler),[],2)), ...
    dNumIntCodes,1,dTotLength,dTotNumCombinations)),[],3)),2), ...
    dTotLength,[]);

sCorrelationHistogramLog.mOdd = ...
    shiftdim(reshape( hist( ...
    20*log10(abs(mOddCorrelation)./dTotLength), ...
    vCorrelationMagnitudeLog), ...
    [],dTotNumCombinations,dNumIntCodes*dNumDesCodes),2) ...
    ./ dTotLength;

sCorrelationHistogramNat.vOdd = sum(reshape( ...
    hist(abs(mOddCorrelation),vCorrelationMagnitudeNat),...
    [],dTotNumCombinations,dNumIntCodes*dNumDesCodes),3)';
```

Código Paralelizado

```
% Clear unused variables
clear sInputVars mDesiredCode mInterferingCode dNumBits dNumComb ...
mBitCombinations dNumRows vTempBitComb ...
vTempShiftedBitComb dIndex vIndex mTempFlips mTempInterferingCode

% This packs variables in contiguous space
save('fjklq4fji4qof4jqiofpqf9q40fuq9fquf9q4uf90qFRDSGREGER54');
clear;
load('fjklq4fji4qof4jqiofpqf9q40fuq9fquf9q4uf90qFRDSGREGER54');
delete('fjklq4fji4qof4jqiofpqf9q40fuq9fquf9q4uf90qFRDSGREGER54.mat');

% Get biggest array size and Max amount of data that fits into memory
dBiggestArray = dNumDesCodes* dNumIntCodes* dTotNumCombinations* dTotLength;
cTemp = 1 + 1i; dSize = whos('cTemp'); sMemory = memory; %#ok<NASGU>
dMaxSize = sMemory.MaxPossibleArrayBytes / (4 * dSize.bytes); %Enough

% Calculate number of chunks to partition the computation into from
% the maximum size an array can be
if dBiggestArray > dMaxSize
    dNumChunks = floor(dBiggestArray / dMaxSize);
else
   dNumChunks = 1;
end

% if PCT is active make sure we have at least one chunk per worker
dNumWorkers = matlabpool('size');
if dNumChunks < dNumWorkers
   dNumChunks = dNumWorkers;
end

% Try to avoid remainder computation because it slows overall process,
% This will work properly unless dNumDescodes is a product of big primes
while mod(dNumDesCodes,dNumChunks)
   dNumChunks = dNumChunks + 1;
end

% Final chunk size and remainder size
dChunkSize = floor(dNumDesCodes / dNumChunks);

% Memory allocation
mOdd = zeros(dNumIntCodes*dChunkSize,dNumChunks, ...
   length(vCorrelationMagnitudeLog),dTotNumCombinations);
vOdd = zeros(dTotNumCombinations,length(vCorrelationMagnitudeNat));

% Reshape for the parfor
mDesiredCodeExtTemp = reshape(mDesiredCodeExt(1:dNumDesCodes,:), ...
   dChunkSize,dNumChunks,dTotLength);

% Compute the cross-correlation histogram (vectorized version)
mInterferingCodeExtDoppler = reshape(conj(fft(bsxfun(@times, ...
   mInterferingCodeExt,vDoppler),[],2)), ...
   dNumIntCodes,1,dTotLength,dTotNumCombinations);
```

```
% Loop though all chunks, in parallel if possible
parfor i1 = 1:dNumChunks

    mOddCorrelation = reshape(shiftdim(real(ifft(bsxfun(@times, ...
       reshape(fft(mDesiredCodeExtTemp(:,i1,:),[],3),1,dChunkSize, ...
       dTotLength), mInterferingCodeExtDoppler),[],3)),2),dTotLength,[]);

    mOdd(:,i1,:,:) = shiftdim(reshape( ...
       hist( 20.*log10(abs(mOddCorrelation)./dTotLength), ...
       vCorrelationMagnitudeLog), [],dTotNumCombinations, ...
       dNumIntCodes*dChunkSize),2) ...
       ./ dTotLength;

    vOdd = vOdd + sum(reshape( ...
       hist(abs(mOddCorrelation),vCorrelationMagnitudeNat),...
       [],dTotNumCombinations,dNumIntCodes*dChunkSize),3)';

end

% Save computed histograms to output structures
sCorrelationHistogramLog.mOdd = reshape(mOdd,dNumIntCodes * ...
   (dNumDesCodes), length(vCorrelationMagnitudeLog),dTotNumCombinations);
sCorrelationHistogramNat.vOdd = vOdd;
```

Registro Detallado de Pruebas

Galileo E1OS vs Código de prueba

Size of Desired Code: 100 * 4092 = 409200
Size of Interfering Code: 10 * 4092 = 40920
Number of operations: 16368000

Original function test: iteration 1
Test took 7.204349e+000 seconds

Original function test: iteration 2
Test took 7.116012e+000 seconds

Original function test: iteration 3
Test took 7.113576e+000 seconds

Original function test: iteration 4
Test took 7.120285e+000 seconds

Original function test: iteration 5
Test took 7.113468e+000 seconds

Original function test: iteration 6
Test took 7.104463e+000 seconds

Original function test: iteration 7
Test took 7.101905e+000 seconds

Original function test: iteration 8
Test took 7.112294e+000 seconds

Original function test: iteration 9
Test took 7.106759e+000 seconds

Original function test: iteration 10
Test took 7.113659e+000 seconds

Mean time for original function: 7.120677e+000 seconds

Vectorization test: iteration 1
Test took 6.074601e+000 seconds

Vectorization test: iteration 2
Test took 6.032805e+000 seconds

Vectorization test: iteration 3
Test took 6.018464e+000 seconds

Vectorization test: iteration 4
Test took 6.038645e+000 seconds

Vectorization test: iteration 5
Test took 6.061127e+000 seconds

Vectorization test: iteration 6
Test took 6.046472e+000 seconds

Vectorization test: iteration 7
Test took 6.021827e+000 seconds

Vectorization test: iteration 8
Test took 6.039734e+000 seconds

Vectorization test: iteration 9
Test took 6.023767e+000 seconds

Vectorization test: iteration 10
Test took 6.053223e+000 seconds

Mean time for vectorized function: 6.041067e+000 seconds

Paralelization test (1 workers): iteration 1
Test took 8.350339e+000 seconds

Paralelization test (1 workers): iteration 2
Test took 8.024923e+000 seconds

Paralelization test (1 workers): iteration 3
Test took 8.041908e+000 seconds

Paralelization test (1 workers): iteration 4
Test took 8.042631e+000 seconds

Paralelization test (1 workers): iteration 5
Test took 8.056062e+000 seconds

Paralelization test (1 workers): iteration 6

Test took 8.037763e+000 seconds

Paralelization test (1 workers): iteration 7
Test took 8.061843e+000 seconds

Paralelization test (1 workers): iteration 8
Test took 8.050014e+000 seconds

Paralelization test (1 workers): iteration 9
Test took 8.053528e+000 seconds

Paralelization test (1 workers): iteration 10
Test took 8.044153e+000 seconds

Mean time for paralelized function (1 workers): 8.076316e+000 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 2 labs.
Paralelization test (2 workers): iteration 1
Test took 5.188011e+000 seconds

Paralelization test (2 workers): iteration 2
Test took 4.859721e+000 seconds

Paralelization test (2 workers): iteration 3
Test took 4.834535e+000 seconds

Paralelization test (2 workers): iteration 4
Test took 4.951344e+000 seconds

Paralelization test (2 workers): iteration 5
Test took 4.880815e+000 seconds

Paralelization test (2 workers): iteration 6
Test took 4.913995e+000 seconds

Paralelization test (2 workers): iteration 7
Test took 4.909172e+000 seconds

Paralelization test (2 workers): iteration 8
Test took 4.906089e+000 seconds

Paralelization test (2 workers): iteration 9
Test took 4.901155e+000 seconds

Paralelization test (2 workers): iteration 10
Test took 4.891437e+000 seconds

Mean time for paralelized function (2 workers): 4.923628e+000 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 3 labs.
Paralelization test (3 workers): iteration 1
Test took 5.115134e+000 seconds

Paralelization test (3 workers): iteration 2
Test took 4.727467e+000 seconds

Paralelization test (3 workers): iteration 3
Test took 4.753545e+000 seconds

Paralelization test (3 workers): iteration 4
Test took 4.616276e+000 seconds

Paralelization test (3 workers): iteration 5
Test took 4.875292e+000 seconds

Paralelization test (3 workers): iteration 6
Test took 4.580108e+000 seconds

Paralelization test (3 workers): iteration 7
Test took 4.899534e+000 seconds

Paralelization test (3 workers): iteration 8
Test took 4.681787e+000 seconds

Paralelization test (3 workers): iteration 9
Test took 4.560740e+000 seconds

Paralelization test (3 workers): iteration 10
Test took 4.598574e+000 seconds

Mean time for paralelized function (3 workers): 4.740846e+000 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 4 labs.
Paralelization test (4 workers): iteration 1
Test took 4.014569e+000 seconds

Paralelization test (4 workers): iteration 2
Test took 3.534832e+000 seconds

Paralelization test (4 workers): iteration 3
Test took 3.543032e+000 seconds

Paralelization test (4 workers): iteration 4
Test took 3.555466e+000 seconds

Paralelization test (4 workers): iteration 5
Test took 3.571426e+000 seconds

Paralelization test (4 workers): iteration 6
Test took 3.546217e+000 seconds

Paralelization test (4 workers): iteration 7
Test took 3.572331e+000 seconds

Paralelization test (4 workers): iteration 8
Test took 3.565209e+000 seconds

Paralelization test (4 workers): iteration 9
Test took 3.649263e+000 seconds

Paralelization test (4 workers): iteration 10
Test took 3.600512e+000 seconds

Mean time for paralelized function (4 workers): 3.615286e+000 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 6 labs.
Paralelization test (6 workers): iteration 1
Test took 4.149970e+000 seconds

Paralelization test (6 workers): iteration 2
Test took 3.702282e+000 seconds

Paralelization test (6 workers): iteration 3
Test took 3.778356e+000 seconds

Paralelization test (6 workers): iteration 4
Test took 3.901571e+000 seconds

Paralelization test (6 workers): iteration 5
Test took 3.782397e+000 seconds

Paralelization test (6 workers): iteration 6
Test took 3.730523e+000 seconds

Paralelization test (6 workers): iteration 7
Test took 3.828897e+000 seconds

Paralelization test (6 workers): iteration 8
Test took 3.539646e+000 seconds

Paralelization test (6 workers): iteration 9
Test took 3.735222e+000 seconds

Paralelization test (6 workers): iteration 10
Test took 3.842354e+000 seconds

Mean time for paralelized function (6 workers): 3.799122e+000 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 8 labs.
Paralelization test (8 workers): iteration 1
Test took 4.587747e+000 seconds

Paralelization test (8 workers): iteration 2
Test took 3.799349e+000 seconds

Paralelization test (8 workers): iteration 3
Test took 3.922482e+000 seconds

Paralelization test (8 workers): iteration 4
Test took 3.644648e+000 seconds

Paralelization test (8 workers): iteration 5
Test took 3.804318e+000 seconds

Paralelization test (8 workers): iteration 6
Test took 3.913598e+000 seconds

Paralelization test (8 workers): iteration 7
Test took 3.638799e+000 seconds

Paralelization test (8 workers): iteration 8
Test took 3.679321e+000 seconds

Paralelization test (8 workers): iteration 9
Test took 3.789654e+000 seconds

Paralelization test (8 workers): iteration 10
Test took 3.926501e+000 seconds

Mean time for paralelized function (8 workers): 3.870642e+000 seconds

Galileo E1OS vs Galileo E1OS

Size of Desired Code: 100 * 4092 = 409200
Size of Interfering Code: 100 * 4092 = 409200
Number of operations: 163680000

Original function test: iteration 1
Test took 7.337345e+001 seconds

Original function test: iteration 2
Test took 7.317807e+001 seconds

Original function test: iteration 3
Test took 7.361047e+001 seconds

Original function test: iteration 4
Test took 7.337703e+001 seconds

Original function test: iteration 5
Test took 7.314042e+001 seconds

Original function test: iteration 6
Test took 7.359485e+001 seconds

Original function test: iteration 7
Test took 7.336724e+001 seconds

Original function test: iteration 8
Test took 7.313567e+001 seconds

Original function test: iteration 9
Test took 7.359770e+001 seconds

Original function test: iteration 10
Test took 7.338579e+001 seconds

Mean time for original function: 7.337607e+001 seconds

Vectorization test: iteration 1
Test took 5.824876e+001 seconds

Vectorization test: iteration 2
Test took 5.822643e+001 seconds

Vectorization test: iteration 3
Test took 5.836071e+001 seconds

Vectorization test: iteration 4
Test took 5.832150e+001 seconds

Vectorization test: iteration 5
Test took 5.829883e+001 seconds

Vectorization test: iteration 6
Test took 5.827940e+001 seconds

Vectorization test: iteration 7
Test took 5.836256e+001 seconds

Vectorization test: iteration 8
Test took 5.835939e+001 seconds

Vectorization test: iteration 9
Test took 5.827665e+001 seconds

Vectorization test: iteration 10
Test took 5.834078e+001 seconds

Mean time for vectorized function: 5.830750e+001 seconds

Starting matlabpool using the 'local' configuration ...
connected to 1 labs.
Paralelization test (1 workers): iteration 1
Test took 7.788909e+001 seconds

Paralelization test (1 workers): iteration 2
Test took 7.739555e+001 seconds

Paralelization test (1 workers): iteration 3
Test took 7.750922e+001 seconds

Paralelization test (1 workers): iteration 4
Test took 7.755277e+001 seconds

Paralelization test (1 workers): iteration 5
Test took 7.749703e+001 seconds

Paralelization test (1 workers): iteration 6
Test took 7.751428e+001 seconds

Paralelization test (1 workers): iteration 7
Test took 7.752486e+001 seconds

Paralelization test (1 workers): iteration 8
Test took 7.752803e+001 seconds

Paralelization test (1 workers): iteration 9
Test took 7.758727e+001 seconds

Paralelization test (1 workers): iteration 10
Test took 7.757774e+001 seconds

Mean time for paralelized function (1 workers): 7.755758e+001 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 2 labs.
Paralelization test (2 workers): iteration 1
Test took 4.501418e+001 seconds

Paralelization test (2 workers): iteration 2
Test took 4.473560e+001 seconds

Paralelization test (2 workers): iteration 3
Test took 4.544476e+001 seconds

Paralelization test (2 workers): iteration 4
Test took 4.558927e+001 seconds

Paralelization test (2 workers): iteration 5
Test took 4.459673e+001 seconds

Paralelization test (2 workers): iteration 6
Test took 4.447794e+001 seconds

Paralelization test (2 workers): iteration 7
Test took 4.579807e+001 seconds

Paralelization test (2 workers): iteration 8
Test took 4.533772e+001 seconds

Paralelization test (2 workers): iteration 9
Test took 4.630345e+001 seconds

Paralelization test (2 workers): iteration 10
Test took 4.593055e+001 seconds

Mean time for paralelized function (2 workers): 4.532283e+001 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 3 labs.
Paralelization test (3 workers): iteration 1
Test took 3.680194e+001 seconds

Paralelization test (3 workers): iteration 2
Test took 3.490535e+001 seconds

Paralelization test (3 workers): iteration 3
Test took 3.452238e+001 seconds

Paralelization test (3 workers): iteration 4
Test took 3.502171e+001 seconds

Paralelization test (3 workers): iteration 5
Test took 3.670276e+001 seconds

Paralelization test (3 workers): iteration 6
Test took 3.521943e+001 seconds

Paralelization test (3 workers): iteration 7
Test took 3.736271e+001 seconds

Paralelization test (3 workers): iteration 8
Test took 3.474632e+001 seconds

Paralelization test (3 workers): iteration 9
Test took 3.478934e+001 seconds

Paralelization test (3 workers): iteration 10
Test took 3.591892e+001 seconds

Mean time for paralelized function (3 workers): 3.559909e+001 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 4 labs.
Paralelization test (4 workers): iteration 1
Test took 3.329196e+001 seconds

Paralelization test (4 workers): iteration 2
Test took 3.324737e+001 seconds

Paralelization test (4 workers): iteration 3
Test took 3.330796e+001 seconds

Paralelization test (4 workers): iteration 4

Test took 3.288827e+001 seconds

Paralelization test (4 workers): iteration 5
Test took 3.414778e+001 seconds

Paralelization test (4 workers): iteration 6
Test took 3.347865e+001 seconds

Paralelization test (4 workers): iteration 7
Test took 3.401685e+001 seconds

Paralelization test (4 workers): iteration 8
Test took 3.307125e+001 seconds

Paralelization test (4 workers): iteration 9
Test took 3.278782e+001 seconds

Paralelization test (4 workers): iteration 10
Test took 3.269050e+001 seconds

Mean time for paralelized function (4 workers): 3.329284e+001 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ... connected to 6 labs.
Paralelization test (6 workers): iteration 1
Test took 3.321327e+001 seconds

Paralelization test (6 workers): iteration 2
Test took 3.307876e+001 seconds

Paralelization test (6 workers): iteration 3
Test took 3.263791e+001 seconds

Paralelization test (6 workers): iteration 4
Test took 3.320412e+001 seconds

Paralelization test (6 workers): iteration 5
Test took 3.380434e+001 seconds

Paralelization test (6 workers): iteration 6
Test took 3.318892e+001 seconds

Paralelization test (6 workers): iteration 7
Test took 3.244019e+001 seconds

Paralelization test (6 workers): iteration 8
Test took 3.356368e+001 seconds

Paralelization test (6 workers): iteration 9
Test took 3.346966e+001 seconds

Paralelization test (6 workers): iteration 10
Test took 3.354680e+001 seconds

Mean time for paralelized function (6 workers): 3.321477e+001 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ... connected to 8 labs.
Paralelization test (8 workers): iteration 1
Test took 5.629993e+001 seconds

Paralelization test (8 workers): iteration 2
Test took 4.066622e+001 seconds

Paralelization test (8 workers): iteration 3
Test took 3.433007e+001 seconds

Paralelization test (8 workers): iteration 4
Test took 3.600077e+001 seconds

Paralelization test (8 workers): iteration 5
Test took 3.479797e+001 seconds

Paralelization test (8 workers): iteration 6
Test took 3.446430e+001 seconds

Paralelization test (8 workers): iteration 7
Test took 3.440415e+001 seconds

Paralelization test (8 workers): iteration 8
Test took 3.570771e+001 seconds

Paralelization test (8 workers): iteration 9
Test took 3.343650e+001 seconds

Paralelization test (8 workers): iteration 10
Test took 3.378412e+001 seconds

Mean time for paralelized function (8 workers): 3.738917e+001 seconds

Galileo E1OS vs Galileo E5a I

>> Size of Desired Code: 100 * 4092 = 409200
Size of Interfering Code: 50 * 10232 = 511600
Number of operations: 204640000

Original function test: iteration 1
Test took 9.945295e+001 seconds

Original function test: iteration 2
Test took 1.000775e+002 seconds

Original function test: iteration 3
Test took 9.937495e+001 seconds

Original function test: iteration 4
Test took 9.936818e+001 seconds

Original function test: iteration 5
Test took 9.927409e+001 seconds

Original function test: iteration 6
Test took 1.000828e+002 seconds

Original function test: iteration 7
Test took 9.941552e+001 seconds

Original function test: iteration 8
Test took 9.939544e+001 seconds

Original function test: iteration 9
Test took 9.931905e+001 seconds

Original function test: iteration 10
Test took 1.000842e+002 seconds

Mean time for original function: 9.958446e+001 seconds

Vectorization test: iteration 1
Test took 9.192417e+001 seconds

Vectorization test: iteration 2
Test took 9.174146e+001 seconds

Vectorization test: iteration 3
Test took 9.195457e+001 seconds

Vectorization test: iteration 4
Test took 9.191531e+001 seconds

Vectorization test: iteration 5
Test took 9.181038e+001 seconds

Vectorization test: iteration 6
Test took 9.194220e+001 seconds

Vectorization test: iteration 7
Test took 9.184585e+001 seconds

Vectorization test: iteration 8
Test took 9.189747e+001 seconds

Vectorization test: iteration 9
Test took 9.173335e+001 seconds

Vectorization test: iteration 10
Test took 9.183731e+001 seconds

Mean time for vectorized function: 9.186020e+001 seconds

Starting matlabpool using the 'local' configuration ...
connected to 1 labs.
Paralelization test (1 workers): iteration 1
Test took 1.103470e+002 seconds

Paralelization test (1 workers): iteration 2
Test took 1.101855e+002 seconds

Paralelization test (1 workers): iteration 3
Test took 1.101071e+002 seconds

Paralelization test (1 workers): iteration 4
Test took 1.101545e+002 seconds

Paralelization test (1 workers): iteration 5
Test took 1.102035e+002 seconds

Paralelization test (1 workers): iteration 6
Test took 1.102073e+002 seconds

Paralelization test (1 workers): iteration 7
Test took 1.101820e+002 seconds

Paralelization test (1 workers): iteration 8
Test took 1.102654e+002 seconds

Paralelization test (1 workers): iteration 9
Test took 1.102012e+002 seconds

Paralelization test (1 workers): iteration 10
Test took 1.102005e+002 seconds

Mean time for paralelized function (1 workers): 1.102054e+002 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 2 labs.
Paralelization test (2 workers): iteration 1
Test took 6.444053e+001 seconds

Paralelization test (2 workers): iteration 2
Test took 6.435601e+001 seconds

Paralelization test (2 workers): iteration 3
Test took 6.317444e+001 seconds

Paralelization test (2 workers): iteration 4
Test took 6.375847e+001 seconds

Paralelization test (2 workers): iteration 5
Test took 6.341425e+001 seconds

Paralelization test (2 workers): iteration 6
Test took 6.422422e+001 seconds

Paralelization test (2 workers): iteration 7
Test took 6.300108e+001 seconds

Paralelization test (2 workers): iteration 8
Test took 6.285403e+001 seconds

Paralelization test (2 workers): iteration 9
Test took 6.287318e+001 seconds

Paralelization test (2 workers): iteration 10
Test took 6.337323e+001 seconds

Mean time for paralelized function (2 workers): 6.354694e+001 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 3 labs.
Paralelization test (3 workers): iteration 1
Test took 5.062535e+001 seconds

Paralelization test (3 workers): iteration 2
Test took 5.198562e+001 seconds

Paralelization test (3 workers): iteration 3
Test took 5.068817e+001 seconds

Paralelization test (3 workers): iteration 4
Test took 5.088636e+001 seconds

Paralelization test (3 workers): iteration 5
Test took 5.120127e+001 seconds

Paralelization test (3 workers): iteration 6
Test took 5.079045e+001 seconds

Paralelization test (3 workers): iteration 7
Test took 4.889649e+001 seconds

Paralelization test (3 workers): iteration 8
Test took 4.867341e+001 seconds

Paralelization test (3 workers): iteration 9
Test took 5.159129e+001 seconds

Paralelization test (3 workers): iteration 10
Test took 5.162893e+001 seconds

Mean time for paralelized function (3 workers): 5.069673e+001 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 4 labs.
Paralelization test (4 workers): iteration 1
Test took 4.783033e+001 seconds

Paralelization test (4 workers): iteration 2
Test took 4.779371e+001 seconds

Paralelization test (4 workers): iteration 3
Test took 4.773536e+001 seconds

Paralelization test (4 workers): iteration 4

Test took 4.709349e+001 seconds

Paralelization test (4 workers): iteration 5
Test took 4.690194e+001 seconds

Paralelization test (4 workers): iteration 6
Test took 4.715136e+001 seconds

Paralelization test (4 workers): iteration 7
Test took 4.725443e+001 seconds

Paralelization test (4 workers): iteration 8
Test took 4.745618e+001 seconds

Paralelization test (4 workers): iteration 9
Test took 4.611957e+001 seconds

Paralelization test (4 workers): iteration 10
Test took 4.610300e+001 seconds

Mean time for paralelized function (4 workers): 4.714394e+001 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 6 labs.
Paralelization test (6 workers): iteration 1
Test took 4.708668e+001 seconds

Paralelization test (6 workers): iteration 2
Test took 4.645192e+001 seconds

Paralelization test (6 workers): iteration 3
Test took 5.027155e+001 seconds

Paralelization test (6 workers): iteration 4
Test took 4.640104e+001 seconds

Paralelization test (6 workers): iteration 5
Test took 4.926839e+001 seconds

Paralelization test (6 workers): iteration 6
Test took 4.723789e+001 seconds

Paralelization test (6 workers): iteration 7
Test took 4.631054e+001 seconds

Paralelization test (6 workers): iteration 8
Test took 4.634271e+001 seconds

Paralelization test (6 workers): iteration 9
Test took 4.572396e+001 seconds

Paralelization test (6 workers): iteration 10
Test took 4.680543e+001 seconds

Mean time for paralelized function (6 workers): 4.719001e+001 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 8 labs.
Paralelization test (8 workers): iteration 1
Test took 8.949727e+001 seconds

Paralelization test (8 workers): iteration 2
Test took 5.240460e+001 seconds

Paralelization test (8 workers): iteration 3
Test took 4.770685e+001 seconds

Paralelization test (8 workers): iteration 4
Test took 7.082699e+001 seconds

Paralelization test (8 workers): iteration 5
Test took 5.840942e+001 seconds

Paralelization test (8 workers): iteration 6
Test took 4.702396e+001 seconds

Paralelization test (8 workers): iteration 7
Test took 4.943002e+001 seconds

Paralelization test (8 workers): iteration 8
Test took 4.728819e+001 seconds

Paralelization test (8 workers): iteration 9
Test took 8.109415e+001 seconds

Paralelization test (8 workers): iteration 10
Test took 6.297037e+001 seconds

Mean time for paralelized function (8 workers): 6.066518e+001 seconds

Galileo E1OS vs GPS L1C

Size of Desired Code: 100 * 4092 = 409200
Size of Interfering Code: 126 * 10230 = 1288980
Number of operations: 515592000

Original function test: iteration 1
Test took 2.287853e+002 seconds

Original function test: iteration 2
Test took 2.280156e+002 seconds

Original function test: iteration 3
Test took 2.299133e+002 seconds

Original function test: iteration 4
Test took 2.275880e+002 seconds

Original function test: iteration 5
Test took 2.278776e+002 seconds

Original function test: iteration 6
Test took 2.299542e+002 seconds

Original function test: iteration 7
Test took 2.264995e+002 seconds

Original function test: iteration 8
Test took 2.285086e+002 seconds

Original function test: iteration 9
Test took 2.278742e+002 seconds

Original function test: iteration 10
Test took 2.265526e+002 seconds

Mean time for original function: 2.281569e+002 seconds

Vectorization test: iteration 1
Test took 2.213902e+002 seconds

Vectorization test: iteration 2
Test took 2.224804e+002 seconds

Vectorization test: iteration 3
Test took 2.219157e+002 seconds

Vectorization test: iteration 4
Test took 2.215253e+002 seconds

Vectorization test: iteration 5
Test took 2.216899e+002 seconds

Vectorization test: iteration 6
Test took 2.217287e+002 seconds

Vectorization test: iteration 7
Test took 2.217840e+002 seconds

Vectorization test: iteration 8
Test took 2.215695e+002 seconds

Vectorization test: iteration 9
Test took 2.218146e+002 seconds

Vectorization test: iteration 10
Test took 2.215394e+002 seconds

Mean time for vectorized function: 2.217438e+002 seconds

Starting matlabpool using the 'local' configuration ...
connected to 1 labs.
Paralelization test (1 workers): iteration 1
Test took 2.675471e+002 seconds

Paralelization test (1 workers): iteration 2
Test took 2.672191e+002 seconds

Paralelization test (1 workers): iteration 3
Test took 2.668871e+002 seconds

Paralelization test (1 workers): iteration 4
Test took 2.670801e+002 seconds

Paralelization test (1 workers): iteration 5
Test took 2.668658e+002 seconds

Paralelization test (1 workers): iteration 6
Test took 2.671958e+002 seconds

Paralelization test (1 workers): iteration 7
Test took 2.670605e+002 seconds

Paralelization test (1 workers): iteration 8
Test took 2.670635e+002 seconds

Paralelization test (1 workers): iteration 9
Test took 2.673304e+002 seconds

Paralelization test (1 workers): iteration 10
Test took 2.673593e+002 seconds

Mean time for paralelized function (1 workers): 2.671609e+002 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 2 labs.
Paralelization test (2 workers): iteration 1
Test took 1.573133e+002 seconds

Paralelization test (2 workers): iteration 2
Test took 1.559092e+002 seconds

Paralelization test (2 workers): iteration 3
Test took 1.540356e+002 seconds

Paralelization test (2 workers): iteration 4
Test took 1.527646e+002 seconds

Paralelization test (2 workers): iteration 5
Test took 1.535915e+002 seconds

Paralelization test (2 workers): iteration 6
Test took 1.534741e+002 seconds

Paralelization test (2 workers): iteration 7
Test took 1.625441e+002 seconds

Paralelization test (2 workers): iteration 8
Test took 1.562429e+002 seconds

Paralelization test (2 workers): iteration 9
Test took 1.559047e+002 seconds

Paralelization test (2 workers): iteration 10
Test took 1.545841e+002 seconds

Mean time for paralelized function (2 workers): 1.556364e+002 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 3 labs.
Paralelization test (3 workers): iteration 1
Test took 1.233047e+002 seconds

Paralelization test (3 workers): iteration 2
Test took 1.233821e+002 seconds

Paralelization test (3 workers): iteration 3
Test took 1.228636e+002 seconds

Paralelization test (3 workers): iteration 4
Test took 1.203328e+002 seconds

Paralelization test (3 workers): iteration 5
Test took 1.203512e+002 seconds

Paralelization test (3 workers): iteration 6
Test took 1.207012e+002 seconds

Paralelization test (3 workers): iteration 7
Test took 1.212253e+002 seconds

Paralelization test (3 workers): iteration 8
Test took 1.210576e+002 seconds

Paralelization test (3 workers): iteration 9
Test took 1.209943e+002 seconds

Paralelization test (3 workers): iteration 10
Test took 1.208156e+002 seconds

Mean time for paralelized function (3 workers): 1.215028e+002 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 4 labs.
Paralelization test (4 workers): iteration 1
Test took 1.183733e+002 seconds

Paralelization test (4 workers): iteration 2
Test took 1.163429e+002 seconds

Paralelization test (4 workers): iteration 3
Test took 1.152734e+002 seconds

Paralelization test (4 workers): iteration 4

Test took 1.146903e+002 seconds

Paralelization test (4 workers): iteration 5
Test took 1.198839e+002 seconds

Paralelization test (4 workers): iteration 6
Test took 1.170180e+002 seconds

Paralelization test (4 workers): iteration 7
Test took 1.162003e+002 seconds

Paralelization test (4 workers): iteration 8
Test took 1.171831e+002 seconds

Paralelization test (4 workers): iteration 9
Test took 1.156322e+002 seconds

Paralelization test (4 workers): iteration 10
Test took 1.161668e+002 seconds

Mean time for paralelized function (4 workers): 1.166764e+002 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 6 labs.
Paralelization test (6 workers): iteration 1
Test took 1.390532e+002 seconds

Paralelization test (6 workers): iteration 2
Test took 1.283608e+002 seconds

Paralelization test (6 workers): iteration 3
Test took 1.188729e+002 seconds

Paralelization test (6 workers): iteration 4
Test took 1.183490e+002 seconds

Paralelization test (6 workers): iteration 5
Test took 1.196858e+002 seconds

Paralelization test (6 workers): iteration 6
Test took 1.197489e+002 seconds

Paralelization test (6 workers): iteration 7
Test took 1.180056e+002 seconds

Paralelization test (6 workers): iteration 8
Test took 1.185418e+002 seconds

Paralelization test (6 workers): iteration 9
Test took 1.172187e+002 seconds

Paralelization test (6 workers): iteration 10
Test took 1.189358e+002 seconds

Mean time for paralelized function (6 workers): 1.216772e+002 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 8 labs.
Paralelization test (8 workers): iteration 1
Test took 1.774428e+002 seconds

Paralelization test (8 workers): iteration 2
Test took 2.689427e+002 seconds

Paralelization test (8 workers): iteration 3
Test took 1.739691e+002 seconds

Paralelization test (8 workers): iteration 4
Test took 2.794010e+002 seconds

Paralelization test (8 workers): iteration 5
Test took 2.403673e+002 seconds

Paralelization test (8 workers): iteration 6
Test took 1.349971e+002 seconds

Paralelization test (8 workers): iteration 7
Test took 2.210490e+002 seconds

Paralelization test (8 workers): iteration 8
Test took 3.013925e+002 seconds

Paralelization test (8 workers): iteration 9
Test took 2.500660e+002 seconds

Paralelization test (8 workers): iteration 10
Test took 1.736543e+002 seconds

Mean time for paralelized function (8 workers): 2.221282e+002 secondsSending a stop signal to all the labs ... stopped.

Galileo E5a I vs Galileo E5a I

Size of Desired Code: 50 * 10232 = 511600
Size of Interfering Code: 50 * 10232 = 511600
Number of operations: 102320000

Original function test: iteration 1
Test took 5.181407e+001 seconds

Original function test: iteration 2
Test took 5.033496e+001 seconds

Original function test: iteration 3
Test took 5.028617e+001 seconds

Original function test: iteration 4
Test took 5.029432e+001 seconds

Original function test: iteration 5
Test took 5.028732e+001 seconds

Original function test: iteration 6
Test took 5.028755e+001 seconds

Original function test: iteration 7
Test took 5.049856e+001 seconds

Original function test: iteration 8
Test took 5.038577e+001 seconds

Original function test: iteration 9
Test took 5.036845e+001 seconds

Original function test: iteration 10
Test took 5.036897e+001 seconds

Mean time for original function: 5.049261e+001 seconds

Vectorization test: iteration 1
Test took 4.641750e+001 seconds

Vectorization test: iteration 2
Test took 4.654186e+001 seconds

Vectorization test: iteration 3
Test took 4.644476e+001 seconds

Vectorization test: iteration 4
Test took 4.646189e+001 seconds

Vectorization test: iteration 5
Test took 4.663313e+001 seconds

Vectorization test: iteration 6
Test took 4.650950e+001 seconds

Vectorization test: iteration 7
Test took 4.657960e+001 seconds

Vectorization test: iteration 8
Test took 4.652975e+001 seconds

Vectorization test: iteration 9
Test took 4.642395e+001 seconds

Vectorization test: iteration 10
Test took 4.659548e+001 seconds

Mean time for vectorized function: 4.651374e+001 seconds

Starting matlabpool using the 'local' configuration ...
connected to 1 labs.
Paralelization test (1 workers): iteration 1
Test took 5.642238e+001 seconds

Paralelization test (1 workers): iteration 2
Test took 5.606958e+001 seconds

Paralelization test (1 workers): iteration 3
Test took 5.606929e+001 seconds

Paralelization test (1 workers): iteration 4
Test took 5.610106e+001 seconds

Paralelization test (1 workers): iteration 5
Test took 5.611729e+001 seconds

Paralelization test (1 workers): iteration 6
Test took 5.611052e+001 seconds

Paralelization test (1 workers): iteration 7
Test took 5.609363e+001 seconds

Paralelization test (1 workers): iteration 8
Test took 5.609663e+001 seconds

Paralelization test (1 workers): iteration 9
Test took 5.612877e+001 seconds

Paralelization test (1 workers): iteration 10
Test took 5.613337e+001 seconds

Mean time for paralelized function (1 workers): 5.613425e+001 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 2 labs.
Paralelization test (2 workers): iteration 1
Test took 3.439288e+001 seconds

Paralelization test (2 workers): iteration 2
Test took 3.391733e+001 seconds

Paralelization test (2 workers): iteration 3
Test took 3.332480e+001 seconds

Paralelization test (2 workers): iteration 4
Test took 3.377591e+001 seconds

Paralelization test (2 workers): iteration 5
Test took 3.363847e+001 seconds

Paralelization test (2 workers): iteration 6
Test took 3.407207e+001 seconds

Paralelization test (2 workers): iteration 7
Test took 3.383244e+001 seconds

Paralelization test (2 workers): iteration 8
Test took 3.310050e+001 seconds

Paralelization test (2 workers): iteration 9
Test took 3.362828e+001 seconds

Paralelization test (2 workers): iteration 10
Test took 3.392414e+001 seconds

Mean time for paralelized function (2 workers): 3.376068e+001 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 3 labs.
Paralelization test (3 workers): iteration 1
Test took 2.880869e+001 seconds

Paralelization test (3 workers): iteration 2
Test took 2.843752e+001 seconds

Paralelization test (3 workers): iteration 3
Test took 2.814015e+001 seconds

Paralelization test (3 workers): iteration 4
Test took 2.823386e+001 seconds

Paralelization test (3 workers): iteration 5
Test took 2.875909e+001 seconds

Paralelization test (3 workers): iteration 6
Test took 2.874833e+001 seconds

Paralelization test (3 workers): iteration 7
Test took 2.844465e+001 seconds

Paralelization test (3 workers): iteration 8
Test took 2.800746e+001 seconds

Paralelization test (3 workers): iteration 9
Test took 2.830076e+001 seconds

Paralelization test (3 workers): iteration 10
Test took 2.835741e+001 seconds

Mean time for paralelized function (3 workers): 2.842379e+001 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 4 labs.
Paralelization test (4 workers): iteration 1
Test took 2.553546e+001 seconds

Paralelization test (4 workers): iteration 2
Test took 2.558699e+001 seconds

Paralelization test (4 workers): iteration 3
Test took 2.686825e+001 seconds

Paralelization test (4 workers): iteration 4

Test took 2.616707e+001 seconds

Paralelization test (4 workers): iteration 5
Test took 2.737150e+001 seconds

Paralelization test (4 workers): iteration 6
Test took 2.707540e+001 seconds

Paralelization test (4 workers): iteration 7
Test took 2.551317e+001 seconds

Paralelization test (4 workers): iteration 8
Test took 2.749703e+001 seconds

Paralelization test (4 workers): iteration 9
Test took 2.626759e+001 seconds

Paralelization test (4 workers): iteration 10
Test took 2.543616e+001 seconds

Mean time for paralelized function (4 workers): 2.633186e+001 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 6 labs.
Paralelization test (6 workers): iteration 1
Test took 2.689319e+001 seconds

Paralelization test (6 workers): iteration 2
Test took 2.633820e+001 seconds

Paralelization test (6 workers): iteration 3
Test took 2.739175e+001 seconds

Paralelization test (6 workers): iteration 4
Test took 2.643628e+001 seconds

Paralelization test (6 workers): iteration 5
Test took 2.730447e+001 seconds

Paralelization test (6 workers): iteration 6
Test took 2.723840e+001 seconds

Paralelization test (6 workers): iteration 7
Test took 2.582240e+001 seconds

Paralelization test (6 workers): iteration 8
Test took 2.677658e+001 seconds

Paralelization test (6 workers): iteration 9
Test took 2.700600e+001 seconds

Paralelization test (6 workers): iteration 10
Test took 2.512888e+001 seconds

Mean time for paralelized function (6 workers): 2.663361e+001 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 8 labs.
Paralelization test (8 workers): iteration 1
Test took 8.331908e+001 seconds

Paralelization test (8 workers): iteration 2
Test took 3.430029e+001 seconds

Paralelization test (8 workers): iteration 3
Test took 3.000163e+001 seconds

Paralelization test (8 workers): iteration 4
Test took 3.057931e+001 seconds

Paralelization test (8 workers): iteration 5
Test took 2.996362e+001 seconds

Paralelization test (8 workers): iteration 6
Test took 2.863304e+001 seconds

Paralelization test (8 workers): iteration 7
Test took 2.643518e+001 seconds

Paralelization test (8 workers): iteration 8
Test took 2.878104e+001 seconds

Paralelization test (8 workers): iteration 9
Test took 2.937917e+001 seconds

Paralelization test (8 workers): iteration 10
Test took 2.645141e+001 seconds

Mean time for paralelized function (8 workers): 3.478438e+001 secondsSending a stop signal to all the labs ... stopped.

Galileo E5a I vs GPS L1C

Size of Desired Code: 50 * 10232 = 511600
Size of Interfering Code: 126 * 10230 = 1288980
Number of operations: 386694000

Original function test: iteration 1
Test took 1.743246e+002 seconds

Original function test: iteration 2
Test took 1.720125e+002 seconds

Original function test: iteration 3
Test took 1.716207e+002 seconds

Original function test: iteration 4
Test took 1.714756e+002 seconds

Original function test: iteration 5
Test took 1.716048e+002 seconds

Original function test: iteration 6
Test took 1.717638e+002 seconds

Original function test: iteration 7
Test took 1.717011e+002 seconds

Original function test: iteration 8
Test took 1.716649e+002 seconds

Original function test: iteration 9
Test took 1.724741e+002 seconds

Original function test: iteration 10
Test took 1.718331e+002 seconds

Mean time for original function: 1.720475e+002 seconds

Vectorization test: iteration 1
Test took 1.699758e+002 seconds

Vectorization test: iteration 2
Test took 1.698246e+002 seconds

Vectorization test: iteration 3
Test took 1.709799e+002 seconds

Vectorization test: iteration 4
Test took 1.721551e+002 seconds

Vectorization test: iteration 5
Test took 1.702890e+002 seconds

Vectorization test: iteration 6
Test took 1.707572e+002 seconds

Vectorization test: iteration 7
Test took 1.702272e+002 seconds

Vectorization test: iteration 8
Test took 1.704596e+002 seconds

Vectorization test: iteration 9
Test took 1.705003e+002 seconds

Vectorization test: iteration 10
Test took 1.704313e+002 seconds

Mean time for vectorized function: 1.705600e+002 seconds

Starting matlabpool using the 'local' configuration ...
connected to 1 labs.
Paralelization test (1 workers): iteration 1
Test took 2.051717e+002 seconds

Paralelization test (1 workers): iteration 2
Test took 2.069255e+002 seconds

Paralelization test (1 workers): iteration 3
Test took 2.068237e+002 seconds

Paralelization test (1 workers): iteration 4
Test took 2.065896e+002 seconds

Paralelization test (1 workers): iteration 5
Test took 2.071652e+002 seconds

Paralelization test (1 workers): iteration 6
Test took 2.068305e+002 seconds

Paralelization test (1 workers): iteration 7
Test took 2.070528e+002 seconds

Paralelization test (1 workers): iteration 8
Test took 2.069012e+002 seconds

Paralelization test (1 workers): iteration 9
Test took 2.067473e+002 seconds

Paralelization test (1 workers): iteration 10
Test took 2.068076e+002 seconds

Mean time for paralelized function (1 workers): 2.067015e+002 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 2 labs.
Paralelization test (2 workers): iteration 1
Test took 1.209077e+002 seconds

Paralelization test (2 workers): iteration 2
Test took 1.205673e+002 seconds

Paralelization test (2 workers): iteration 3
Test took 1.187218e+002 seconds

Paralelization test (2 workers): iteration 4
Test took 1.224316e+002 seconds

Paralelization test (2 workers): iteration 5
Test took 1.216477e+002 seconds

Paralelization test (2 workers): iteration 6
Test took 1.166889e+002 seconds

Paralelization test (2 workers): iteration 7
Test took 1.172392e+002 seconds

Paralelization test (2 workers): iteration 8
Test took 1.163622e+002 seconds

Paralelization test (2 workers): iteration 9
Test took 1.178131e+002 seconds

Paralelization test (2 workers): iteration 10
Test took 1.176245e+002 seconds

Mean time for paralelized function (2 workers): 1.190004e+002 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 3 labs.
Paralelization test (3 workers): iteration 1
Test took 9.529279e+001 seconds

Paralelization test (3 workers): iteration 2
Test took 9.675169e+001 seconds

Paralelization test (3 workers): iteration 3
Test took 9.530056e+001 seconds

Paralelization test (3 workers): iteration 4
Test took 9.716705e+001 seconds

Paralelization test (3 workers): iteration 5
Test took 9.776782e+001 seconds

Paralelization test (3 workers): iteration 6
Test took 9.534260e+001 seconds

Paralelization test (3 workers): iteration 7
Test took 9.533364e+001 seconds

Paralelization test (3 workers): iteration 8
Test took 9.590262e+001 seconds

Paralelization test (3 workers): iteration 9
Test took 9.483588e+001 seconds

Paralelization test (3 workers): iteration 10
Test took 9.652747e+001 seconds

Mean time for paralelized function (3 workers): 9.602221e+001 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 4 labs.
Paralelization test (4 workers): iteration 1
Test took 9.178305e+001 seconds

Paralelization test (4 workers): iteration 2
Test took 9.230038e+001 seconds

Paralelization test (4 workers): iteration 3
Test took 9.290222e+001 seconds

Paralelization test (4 workers): iteration 4

Test took 8.885372e+001 seconds

Paralelization test (4 workers): iteration 5
Test took 9.019942e+001 seconds

Paralelization test (4 workers): iteration 6
Test took 9.186076e+001 seconds

Paralelization test (4 workers): iteration 7
Test took 9.077629e+001 seconds

Paralelization test (4 workers): iteration 8
Test took 9.153314e+001 seconds

Paralelization test (4 workers): iteration 9
Test took 9.027877e+001 seconds

Paralelization test (4 workers): iteration 10
Test took 9.210942e+001 seconds

Mean time for paralelized function (4 workers): 9.125972e+001 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 6 labs.
Paralelization test (6 workers): iteration 1
Test took 9.650475e+001 seconds

Paralelization test (6 workers): iteration 2
Test took 1.308240e+002 seconds

Paralelization test (6 workers): iteration 3
Test took 9.437661e+001 seconds

Paralelization test (6 workers): iteration 4
Test took 9.143342e+001 seconds

Paralelization test (6 workers): iteration 5
Test took 9.214043e+001 seconds

Paralelization test (6 workers): iteration 6
Test took 9.049071e+001 seconds

Paralelization test (6 workers): iteration 7
Test took 9.147385e+001 seconds

Paralelization test (6 workers): iteration 8
Test took 9.066120e+001 seconds

Paralelization test (6 workers): iteration 9
Test took 9.141388e+001 seconds

Paralelization test (6 workers): iteration 10
Test took 8.976024e+001 seconds

Mean time for paralelized function (6 workers): 9.590791e+001 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 8 labs.
Paralelization test (8 workers): iteration 1
Test took 1.461562e+002 seconds

Paralelization test (8 workers): iteration 2
Test took 1.491652e+002 seconds

Paralelization test (8 workers): iteration 3
Test took 1.767950e+002 seconds

Paralelization test (8 workers): iteration 4
Test took 1.142838e+002 seconds

Paralelization test (8 workers): iteration 5
Test took 1.681327e+002 seconds

Paralelization test (8 workers): iteration 6
Test took 1.479509e+002 seconds

Paralelization test (8 workers): iteration 7
Test took 1.147910e+002 seconds

Paralelization test (8 workers): iteration 8
Test took 1.750491e+002 seconds

Paralelization test (8 workers): iteration 9
Test took 2.422198e+002 seconds

Paralelization test (8 workers): iteration 10
Test took 1.733375e+002 seconds

Mean time for paralelized function (8 workers): 1.607881e+002 seconds

>> Sending a stop signal to all the labs ... stopped.

GPS L1C vs GPS L1C

Size of Desired Code: 126 * 10230 = 1288980
Size of Interfering Code: 126 * 10230 = 1288980
Number of operations: 649645920

Original function test: iteration 1
Test took 2.875912e+002 seconds

Original function test: iteration 2
Test took 2.873912e+002 seconds

Original function test: iteration 3
Test took 2.888755e+002 seconds

Original function test: iteration 4
Test took 2.870722e+002 seconds

Original function test: iteration 5
Test took 2.874815e+002 seconds

Original function test: iteration 6
Test took 2.876207e+002 seconds

Original function test: iteration 7
Test took 2.865208e+002 seconds

Original function test: iteration 8
Test took 2.874344e+002 seconds

Original function test: iteration 9
Test took 2.888326e+002 seconds

Original function test: iteration 10
Test took 2.872084e+002 seconds

Mean time for original function: 2.876028e+002 seconds

Vectorization test: iteration 1
Test took 2.794826e+002 seconds

Vectorization test: iteration 2
Test took 2.793366e+002 seconds

Vectorization test: iteration 3
Test took 2.793242e+002 seconds

Vectorization test: iteration 4
Test took 2.792545e+002 seconds

Vectorization test: iteration 5
Test took 2.797900e+002 seconds

Vectorization test: iteration 6
Test took 2.799262e+002 seconds

Vectorization test: iteration 7
Test took 2.796397e+002 seconds

Vectorization test: iteration 8
Test took 2.795021e+002 seconds

Vectorization test: iteration 9
Test took 2.793599e+002 seconds

Vectorization test: iteration 10
Test took 2.792613e+002 seconds

Mean time for vectorized function: 2.794877e+002 seconds

Starting matlabpool using the 'local' configuration ...
connected to 1 labs.
Paralelization test (1 workers): iteration 1
Test took 3.361030e+002 seconds

Paralelization test (1 workers): iteration 2
Test took 3.367494e+002 seconds

Paralelization test (1 workers): iteration 3
Test took 3.367094e+002 seconds

Paralelization test (1 workers): iteration 4
Test took 3.366485e+002 seconds

Paralelization test (1 workers): iteration 5
Test took 3.369333e+002 seconds

Paralelization test (1 workers): iteration 6
Test took 3.366385e+002 seconds

Paralelization test (1 workers): iteration 7
Test took 3.366464e+002 seconds

Paralelization test (1 workers): iteration 8
Test took 3.364871e+002 seconds

Paralelization test (1 workers): iteration 9
Test took 3.364841e+002 seconds

Paralelization test (1 workers): iteration 10
Test took 3.365270e+002 seconds

Mean time for paralelized function (1 workers): 3.365927e+002 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 2 labs.
Paralelization test (2 workers): iteration 1
Test took 1.965989e+002 seconds

Paralelization test (2 workers): iteration 2
Test took 1.978874e+002 seconds

Paralelization test (2 workers): iteration 3
Test took 1.969736e+002 seconds

Paralelization test (2 workers): iteration 4
Test took 1.941810e+002 seconds

Paralelization test (2 workers): iteration 5
Test took 2.034588e+002 seconds

Paralelization test (2 workers): iteration 6
Test took 1.964041e+002 seconds

Paralelization test (2 workers): iteration 7
Test took 1.942174e+002 seconds

Paralelization test (2 workers): iteration 8
Test took 1.940118e+002 seconds

Paralelization test (2 workers): iteration 9
Test took 1.959643e+002 seconds

Paralelization test (2 workers): iteration 10
Test took 1.948945e+002 seconds

Mean time for paralelized function (2 workers): 1.964592e+002 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 3 labs.
Paralelization test (3 workers): iteration 1
Test took 1.535526e+002 seconds

Paralelization test (3 workers): iteration 2
Test took 1.532379e+002 seconds

Paralelization test (3 workers): iteration 3
Test took 1.527538e+002 seconds

Paralelization test (3 workers): iteration 4
Test took 1.544432e+002 seconds

Paralelization test (3 workers): iteration 5
Test took 1.521218e+002 seconds

Paralelization test (3 workers): iteration 6
Test took 1.535084e+002 seconds

Paralelization test (3 workers): iteration 7
Test took 1.539611e+002 seconds

Paralelization test (3 workers): iteration 8
Test took 1.523736e+002 seconds

Paralelization test (3 workers): iteration 9
Test took 1.532851e+002 seconds

Paralelization test (3 workers): iteration 10
Test took 1.580339e+002 seconds

Mean time for paralelized function (3 workers): 1.537271e+002 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 4 labs.
Paralelization test (4 workers): iteration 1
Test took 1.444728e+002 seconds

Paralelization test (4 workers): iteration 2
Test took 1.471491e+002 seconds

Paralelization test (4 workers): iteration 3
Test took 1.442544e+002 seconds

Paralelization test (4 workers): iteration 4

Test took 1.438231e+002 seconds

Paralelization test (4 workers): iteration 5
Test took 1.448373e+002 seconds

Paralelization test (4 workers): iteration 6
Test took 1.443854e+002 seconds

Paralelization test (4 workers): iteration 7
Test took 1.440917e+002 seconds

Paralelization test (4 workers): iteration 8
Test took 1.462611e+002 seconds

Paralelization test (4 workers): iteration 9
Test took 1.442672e+002 seconds

Paralelization test (4 workers): iteration 10
Test took 1.439537e+002 seconds

Mean time for paralelized function (4 workers): 1.447496e+002 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 6 labs.
Paralelization test (6 workers): iteration 1
Test took 1.486149e+002 seconds

Paralelization test (6 workers): iteration 2
Test took 1.548810e+002 seconds

Paralelization test (6 workers): iteration 3
Test took 1.478249e+002 seconds

Paralelization test (6 workers): iteration 4
Test took 1.442996e+002 seconds

Paralelization test (6 workers): iteration 5
Test took 1.463060e+002 seconds

Paralelization test (6 workers): iteration 6
Test took 1.443569e+002 seconds

Paralelization test (6 workers): iteration 7
Test took 1.437713e+002 seconds

Paralelization test (6 workers): iteration 8
Test took 1.442409e+002 seconds

Paralelization test (6 workers): iteration 9
Test took 1.470929e+002 seconds

Paralelization test (6 workers): iteration 10
Test took 1.444710e+002 seconds

Mean time for paralelized function (6 workers): 1.465859e+002 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 8 labs.
Paralelization test (8 workers): iteration 1
Test took 1.936859e+002 seconds

Paralelization test (8 workers): iteration 2
Test took 2.675851e+002 seconds

Paralelization test (8 workers): iteration 3
Test took 2.699862e+002 seconds

Paralelization test (8 workers): iteration 4
Test took 3.678074e+002 seconds

Paralelization test (8 workers): iteration 5
Test took 2.543949e+002 seconds

Paralelization test (8 workers): iteration 6
Test took 2.659066e+002 seconds

Paralelization test (8 workers): iteration 7
Test took 2.379135e+002 seconds

Paralelization test (8 workers): iteration 8
Test took 2.636676e+002 seconds

Paralelization test (8 workers): iteration 9
Test took 2.913445e+002 seconds

Paralelization test (8 workers): iteration 10
Test took 2.359072e+002 seconds

Mean time for paralelized function (8 workers): 2.648199e+002 seconds

GPS L1C vs Galileo E1OS

Size of Desired Code: 126 * 10230 = 1288980
Size of Interfering Code: 100 * 4092 = 409200
Number of operations: 824947200

Original function test: iteration 1
Test took 3.670512e+002 seconds

Vectorization test: iteration 1
Test took 3.429448e+002 seconds

Mean time for vectorized function: 3.429448e+002 seconds

Starting matlabpool using the 'local' configuration ...
connected to 1 labs.
Paralelization test (1 workers): iteration 1
Test took 4.152047e+002 seconds

Mean time for paralelized function (1 workers): 4.152047e+002 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 2 labs.
Paralelization test (2 workers): iteration 1
Test took 2.386202e+002 seconds

Mean time for paralelized function (2 workers): 2.386202e+002 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 3 labs.
Paralelization test (3 workers): iteration 1
Test took 1.886200e+002 seconds

Mean time for paralelized function (3 workers): 1.886200e+002 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 4 labs.
Paralelization test (4 workers): iteration 1
Test took 1.755502e+002 seconds

Mean time for paralelized function (4 workers): 1.755502e+002 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 6 labs.
Paralelization test (6 workers): iteration 1
Test took 1.768018e+002 seconds

Mean time for paralelized function (6 workers): 1.768018e+002 seconds

Sending a stop signal to all the labs ... stopped.
Starting matlabpool using the 'local' configuration ...
connected to 8 labs.
Paralelization test (8 workers): iteration 1
Test took 3.252544e+002 seconds

Mean time for paralelized function (8 workers): 3.252544e+002 seconds

>>

ANEXO E

Glosario

A continuación se definen y explican algunos términos con los que el lector puede no estar familiarizado con el objetivo de facilitar la comprensión de esta memoria

CAJA BLANCA

Elemento estudiado desde el punto de vista de su mecánica interna, al contrario que la perspectiva de caja negra, esta se centra en observar cómo el modelo hace las cosas en lugar de qué es lo que hace.

CAJA NEGRA

Elemento estudiado desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin la obligación de conocer su funcionamiento interno. Lo importante de una caja negra es lo bien definidas que estén sus entradas y salidas, es decir, su interfaz, no el cómo funciona.

CONTROL

Un control es un objeto que reside en un panel accesible para el usuario y que permite visualizar información

Los controles en este proyecto, son aquellos generados mediante las sentencias `uicontrol` o `javacomponent` e integrados en el propio MATLAB o la biblioteca Swing de Java.

EXPRESIÓN REGULAR

Expresión que describe un conjunto de cadenas sin enumerar sus elementos. Una expresión regular es una forma de representar a los lenguajes regulares (finitos o infinitos) y se construye utilizando caracteres del alfabeto sobre el cual se define el lenguaje. Sirven para buscar un texto dentro de otro de una forma sofisticada.

EXTREME PROGRAMMING

Es un enfoque de la ingeniería de software formulado por Kent Beck, autor del primer libro sobre la materia, Extreme Programming Explained: Embrace Change (1999).

La programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad considerarandose como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo con el proyecto, y aplicarlo de manera dinámica durante el ciclo de vida del software.

La simplicidad y la comunicación son extraordinariamente complementarias. Con más comunicación resulta más fácil identificar qué se debe y qué no se debe hacer. Cuanto más simple es el sistema, menos tendrá que comunicar sobre éste, lo que lleva a una comunicación más completa, especialmente si se puede reducir el equipo de programadores.

FIGURA DE MÉRITO

Cantidad utilizada para caracterizar el desempeño de un dispositivo, sistema o método, en comparación con sus alternativas. En ingeniería, figuras de mérito se definen a menudo para determinados materiales o dispositivos con el fin de determinar su utilidad relativa para una aplicación.

FRAMEWORK

Estructura conceptual y tecnológica de soporte que permite que otro proyecto de software pueda ser organizado y desarrollado. Puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

HANDLE

Un tipo particular de punteros "inteligentes". Los handles son utilizados cuando un programa hace referencia a bloques de memoria u objetos controlados por otros sistemas, tales como una base de datos o un sistema operativo.

LEY DE AMDAHL

Indica la mejora de rendimiento que se puede esperar incrementando los elementos de procesamiento. La ley de Amdahl acota de una forma muy sencilla el incremento de prestaciones sostenido en un sistema como consecuencia de la mejora de una o varias partes del mismo.

Esta mejora representada como una mejora del rendimiento, va a depender tanto de la calidad de las mejoras efectuadas como del tiempo que éstas utilicen. De forma abreviada podemos decir que este incremento de prestaciones dará la medida de cómo un equipo rinde en relación con un rendimiento previo después de efectuar en él una o varias mejoras.

PSEUDO-RANDOM NOISE

Es una señal similar al ruido que satisface una o más de las pruebas estándar de aleatoriedad estadística. A pesar de no presentar un patrón definido a priori, el pseudo-random noise consiste en una secuencia determinada de pulsos, repetida durante n periodos.

SCRUM

Scrum es un modelo de referencia, un marco de trabajo para la gestión y desarrollo de software basada en un proceso iterativo e incremental, que puede tomarse como punto de partida para definir el proceso de desarrollo que se ejecutará durante un proyecto, comúnmente en entornos basados en el desarrollo ágil de software.

Scrum permite la creación de equipos autoorganizados impulsando la co-localización de todos los miembros del equipo, y la comunicación verbal entre todos los miembros y disciplinas involucrados en el proyecto.

Adopta una aproximación pragmática, aceptando que el problema no puede ser completamente entendido o definido, y centrándose en maximizar la capacidad del equipo de entregar rápidamente y responder a requisitos emergentes.

SIGNALS Y SLOTS

Es parte del lenguaje de programación introducido en Qt, lo que facilita su implementación con el patrón Observer, evitando así el código repetitivo.

El concepto fundamental es que los widgets tienen la capacidad de enviar señales que contengan información sobre el evento, la cual es recibida por otros controles de uso de funciones especiales denominados slots.

El sistema signal/slot encaja perfectamente con el diseño de las interfaces gráficas para usuarios, así como para su aplicación en E/S asíncronas.

No es necesario generar códigos de registros ya que MetaQt es un objeto perteneciente al compilador MOC, quien genera automáticamente la infraestructura necesaria.

VECTORIZACIÓN

En programación, se trata de una técnica mediante la cual se transforman bucles iterativos en operaciones matriciales.