H2020 – Secure societies - Protecting freedom and security of Europe and its citizens

SU-DRS02-2018-2019-2020– Technologies for first responders – Research and Innovation Action (RIA)

Emerging technologies for the Early location of Entrapped victims under Collapsed Structures & Advanced Wearables for risk assessment and First Responders Safety in SAR operations

D4.12 Development of PHYSIO DSS component, V2

Workpackage:	WP4 – Data aggregation, Analysis and Decision Support
Authors:	KT
Status:	Final
Due Date:	30/04/2022
Version:	1.00
Submission Date:	30/04/2022
Dissemination Level:	PU

Search and Rescue Project Profile

Grant Agreement No.:	882897
Acronym:	Search and Rescue
Title:	Emerging technologies for the Early location of Entrapped victims under Collapsed Structures & Advanced Wearables for risk assessment and First Responders Safety in SAR operations
URL:	https://search-and-rescue.eu/
Start Date:	01/07/2020
Duration:	36 months

Partners

	NATIONAL TECHNICAL UNIVERSITY OF ATHENS (NTUA) Co-ordinator	Greece
	AIDEAS OÜ (AIDEAS)	Estonia
	SOFTWARE IMAGINATION & VISION S.R.L (SIMAVI)	Romania
	MAGGIOLI SPA (MAG)	Italy
	KONNEKT-ABLE TECHNOLOGIES LIMITED (KT)	Ireland
	THALES ITAIA Italia SPA (THALIT)	Italy
	ATOS IT SOLUTIONS AND SERVICES IBERIA SL (ATOS)	Spain
	ETHNIKO KENTRO EREVNAS KAI TECHNOLOGIKIS ANAPTYXIS (CERTH)	Greece
	UNIVERSITA DEGLI STUDI DI CAGLAIRI (UNICA)	Italy

	UKEMED GLOBAL LTD (UGL)	Cyprus
	PUBLIC SAFETY COMMUNICATION EUROPE FORUM AISBL (PSCE)	Belgium
	UNIVERSITA DEGLI STUDI DI FIRENZE (UNIFI)	Italy
	DEUTSCHES FORSCHUNGSZENTRUM FUR KUNSTLICHE INTELLIGENZ (DFKI)	Germany
	UNIVERSITA CATTOLICA DEL SACRO CUORE (UCSC)	Italy
	VRIJE UNIVERSITEIT BRUSSEL	Belgium
	SYNYO GmbH (SYNYO)	Austria
	UNIVERSITEIT HASSELT (UHASSELT)	Belgium
	SPOLECZNA AKADEMIA NAUK (SAN)	Poland
	GIOUMPITEK MELETI SCHEDIASMOS YLOPOIISI KAI POLISI ERGON PLIROFORIKIS ETAIREIA PERIORISMENIS EFTHYNIS (UBITECH)	Greece
Search and Rescue End-Users		
	ELLINIKI OMADA DIASOSIS SOMATEIO (HRT)	Greece

	ENOSI PTYCHIOYCHON AXIOMATIKON YPAXIOOMATIKON PYROSVESTIR OY SOMATEIO (EPAYPS)	Greece
	JOHANNITER-UNFALL-HILFE EV (JOHAN)	Germany
	JOHANNITER OSTERREICH AUSBLIDUNG UND FORSCHUNG GEMEINNUTZIGE GMBH (JOAFG)	Austria
	CONSIGLIO NAZIONALE DELLE RICERCHE	Italy
	POMPIERS DE L'URGENCE INTERNATIONALE (PUI)	France
	ASOCIATA CLUSTERUL ROAMN RENTRU PROTECTIE SI ECOLOGIE IN DOMENIUL MATERIALELOR CHIMICE, BIOLOGICE, RADIOLOGICE/NUCLEARE SI EXPLOZIVE (PROECO)	Romania
	SERVICIO MADRILENO DE SALUD (SERMAS)	Spain
	FUNDACIÓN PARA LA INVESTIGACIÓN E INNOVACIÓN BIOSANITARIA DE ATENCIÓN PRIMARIA (FIIBAP)	Spain
	ESCUELA ESPANOLA DE SALVAMENTO Y DETECCION CON PERROS (ESDP)	Spain

Document History

Version	Date	Author (Partner)	Remarks/Changes
0.10	01/03/2022	Christos Angelidis (KT) Simona Panunzi (CNR) Alessandro Borri (CNR)	ToC
0.20	20/03/2022	Simona Panunzi (CNR) Alessandro Borri (CNR)	Sections 1, 2, 3
0.30	22/03/2022	Simona Panunzi (CNR) Andrea De Gaetano (CNR)	Update of Section 2 and Revision of all sections
0.31	15/04/2022	Cesar Mediavilla Martinez (ATOS)	Internal review
0.32	18/04/2022	Giulia Sedda (UNICA)	Internal review
0.40	19/04/2022	Christos Angelidis (KT) Simona Panunzi (CNR)	All comments and suggestions were addressed
0.41	19/04/2022	Christodoulos Santorinaios (NTUA)	Quality Control
1.00	30/04/2022	Christos Ntanos (NTUA)	FINAL VERSION TO BE SUBMITTED

Executive Summary

The present deliverable is a technical document which provides the specifications for performing the correct calls to the webservices offered by the second version of the PHYSIO DSS (PHYSIOlogical evolution of the victim Decision Support System) component, which is part of the Decision Support System of the S&R project. The implementation of the PHYSIO DSS, that is the webservices residing on the CNR server, constitutes the real result related to part of the work done in the framework of task T4.5 "Development of DSS components".

This deliverable is strictly connected to the deliverable D4.10 (Design of the PHYSIO DSS component, V2) which provides instead a more detailed description of the functionalities offered by the component, by specifying all the modules and functions constituting the PHYSIO DSS.

During the first period of the project, efforts have been addressed in particular to the design of the entire component, with the definition of most of the modules and functions, with the relative implementation in the testing environment (MATLAB) as well as with the set-up of the production (release) environment (a Webservice with a C++ implementation of the modules). After the release of the first version of the component, therefore, most of the work has been made above all for the implementation of the component in the production environment, with the release of the Webservices, and in the verification process for a complete consistency between the two implementations (MATLAB and C++). Details about the verification process will be given in the relative deliverable D4.13 "DSS Validation, V2" foreseen for August 2022.

The document is structured as follows:

1) section 1: introduction with a description of the main objective of the deliverable and with a brief description of the architecture and implementation of the PHYSIO DSS component in its final version;
2) section 2: technical aspects of the component, the client-server architecture, the PHYSIO DSS WSDL document, the method to call the Web Service and the way to parse the response by the server;
3) section 3: conclusions and next steps.

Table of Contents

List of Figures 8

List of Tables 9

List of Abbreviations 10

1 Introduction 11

1.1 Objectives and Scope 11

1.2 Architecture and implementation of The PHYSIO DSS Component 11

2 The client calls to the PHYSIO DSS Component 15

2.1 General specifications 15

2.2 Settings 19

2.3 Call to the "compute-a-priori" PHYSIO service 21

2.4 Call to the "physioevo" PHYSIO service 24

2.5 Call to the "compute-a-posteriori" PHYSIO service 25

2.6 Call to the "ETD_PIE" PHYSIO service 27

2.7 Call to the "GCS" PHYSIO service 28

2.8 Call to the "Start_Triage" PHYSIO service 29

2.9 Call to the "SORT_Triage" PHYSIO service 30

2.10 Call to the "Sieve_Triage" PHYSIO service 31

2.11 Call to the "JumpStart_Triage" PHYSIO service 32

2.12 Call to the "compute_distribution_given_treatment" PHYSIO service 32

3 Conclusions 34

4 References 35

List of Figures

Figure 2-1: Overview of the client-server architecture of the PHYSIO DSS component 14
Figure 2-2: Code related to the "decompose_PIE.m" script, part 1. 18
Figure 2-3: Code related to the "decompose_PIE.m" script, part 2. 19
Figure 2-4: Code related to the "decompose_PIE.m" script, part 3. 19
Figure 2-5: Code related to the "Settings.m" script. 20
Figure 2-6: Code related to the "parse_results.m" function. 21
Figure 2-7: Code related to the "Services_call" script. 22
Figure 2-5: Call to the "compute_a_priori_remote.m" function. 22
Figure 2-9: Content of the "compute_a_priori_remote.m" function. 23
Figure 2-10: Empirical distribution of the "Heart pump function" PSV 24
Figure 2-11: Call to the "physioevo_remote.m" function. 25
Figure 2-12: Content of the "physioevo_remote.m" function. 25
Figure 2-13: Call to the "compute_a_posteriori_given_health_meas_remote.m" function. 26
Figure 2-14: Content of the "compute_a_posteriori_given_health_meas_remote.m" function. 26
Figure 2-15: Call to the "ETD_PIE_remote.m" function 27
Figure 2-16: Content of the "ETD_PIE_remote.m" function 27
Figure 2-17: Empirical distribution of the ETD 28
Figure 2-18: Call to the "GCS_remote.m" function 29
Figure 2-19: Content of the "GCS_remote.m" function 29
Figure 2-20: Call to the "Start_Triage_remote.m" function 29
Figure 2-21: Content of the "Start_Triage_remote.m" function 30
Figure 2-22: Call to the "SORT_Triage_remote.m" function 30
Figure 2-23: Content of the "SORT_Triage_remote.m" function 31
Figure 2-24: Call to the "SIEVE_Triage_remote.m" function 31
Figure 2-25: Content of the "SIEVE_Triage_remote.m" function 32
Figure 2-26: Call to the "JumpStart_Triage_remote.m" function 32
Figure 2-27: Content of the "JumpStart_Triage_remote.m" function 32
Figure 2-28: Call to the "compute_distribution_given_treatment_remote.m" function 33
Figure 2-29: Content of the "compute_distribution_given_treatment_remote.m" function 33

List of Tables

Table 2-1: PHYSIO DSS Webservices function names 17

List of Abbreviations

Abbreviation	Explanation
S&R	Search & Rescue
DSS	Decision Support System
PSV	Physiological State Variable
ETD	Expected Time to Death
LAMP	Linux, Apache, MySQL and PHP
WSDL	Web Services Description Language
SOAP	Simple Object Access Protocol
XML	Extensible Markup Language
M2M	Machine-to-Machine
START	Simple Triage And Rapid Treatment

1 Introduction

Objectives and Scope

Deliverable D4.12 (The PHYSIO DSS component development, V2) describes the specifications to exploit the services offered by the final version of the PHYSIO DSS component described in the deliverable D4.10 (The PHYSIO DSS component design, V2). Both, the design and the development of the PHYSIO DSS, are part of the results of the work carried out in tasks T4.4 and T4.5, dedicated to the design and development of the Decision Support System of the S&R project. The PHYSIO DSS is the component responsible for the prediction of the evolution of the physiological status of the victims, by means of a series of functions, modules and algorithms, in a Bayesian framework where information from the field enriches, in a real-time modality, the knowledge about the health conditions of the patients. The expected state of the victims in the next few hours or minutes allows for a more efficient allocation of the resources on the field in a situation where there is a disproportion between the needs and the available resources. The final objective of the PHYSIO DSS, in fact, is that of providing suggestions about the dispatch of ambulance, treatments or therapies on the basis of a better triage and victim prioritization. The present deliverable, which accompanies the webservices located on the CNR Service, is that of providing a description, from a technical point of view, of the implemented functions, providing for each function the code for the calls to each single service from a MATLAB client, to make the integration of the PHYSIO component in the S&R platform easier. The first UC1 "Victims trapped under the rubbles", which will take place in Poggioreale old town (Trapani, Sicily, Italy) on April 28th, will be the first occasion for testing the functionalities and the integration of the component in a live setting. Since the delivery of the second version of the component is in concomitance with the first S&R Use Case, from which shortcoming could arise, further work to improve functionalities will be carried out until the subsequent S&R UC5, where the component will be tested again, and until the end of the project if necessary.

1.2 Architecture and implementation of The PHYSIO DSS Component

In this section we briefly describe the main functionalities of the PHYSIO DSS and the implementation architecture. The PHYSIO DSS component provides modules, functions and algorithms which aim at providing

- o suggestions about which resources to assign to a victim (for example based on the expected time to death corresponding to each possible treatment to deliver)
- o patient prioritization based on either the expected time to death or according to the values of computed scores and known triage algorithms.

The above objectives are achieved by means of the following functionalities:

- Simulation of different Crisis Scenarios

- Generation of the a-priori distributional physiological status based on very few initial information about the victim
- Update of the evolution of the victim distributional physiological status based on health measurements from the field
- Update of the evolution of the victim distributional physiological state based on assigned treatments
- Estimated Time to Death for each victim
- Automatic computation of relevant indices and scores for the assessment of the victim's impairment and triage

All computations are performed in terms of probability distributions and at each time, all the information about a victim is collected in a PIE, which is an encoded long string summarizing where the victim is likely to be in the Physiological State Variables' (PSVs) space. The PSVs are the dimensions along which patients change their physiological conditions following injuries or therapy/treatment administration. For more details about the structure of the PIE refer to deliverable D4.4 and D4.6. The functionalities related to the simulation of a crisis scenario were needed to verify the functioning of the other PHYSIO DSS services. The scenario_generation service in fact, simulates an incident characterized by a certain number of victims affected by a series of injuries. From this point onwards, each generated victim could be a possible victim taken care of by first responders as in a real situation. At this point all the PHYSIO DSS services run in the same way on a simulated or real victim.

After the function **generate_scenario** simulates a new event (i.e. an earthquake) from the **Event** class, a certain number of victims appears in the site of the incident**,** each one affected by a series of **lesions/injuries (**in the **Lesion** class.**)** randomly extracted from a defined probability distribution. In the simpler case, the probability distribution is a discrete distribution assuming only two values: 1 if the lesion occurs, 0 otherwise. The probability of occurrence of the lesion is predefined and set to the maximum probability that a lesion occurs given the event considered. The **defect_generation** function links the injuries to the physiological variables by causing the occurrence of physiological defects along some or all the physiological dimensions (in the **Physiological State Variables (PSVs)** class). Each PSV can be, in fact, affected by the reported injuries, in terms of changes on both the PSV values and their rates of worsening. The **compute_a_priori** function returns the distributional form of the PSVs with respect to both their initial defect and the rate of worsening. The evolution over time of the distributions of each PSV can occur because of the influence of the possible delivered treatments (by means of the **compute_a_posteriori_given_treatment**) or can occurs freely, if no intervention on the victim is performed. The updated of the distributional state may happen also by means of the **compute_a_posteriori_given_hm** function which combines, with a complete Bayesian approach, the a-priori information and the new information coming from the field on the health status of the victims (measurements of health parameters such as pressure, heart rate, etc…). This second version of the PHYSIO DSS offers, in addition to the computation of the Glasgow Coma Scale score [1], also the automatic triaging of the victims according to the SORT [2], START [3], Sieve [4] and JUMP [5] (for paediatric victims) triage algorithms.

All the PHYSIO DSS functions are delivered as Web Services according to the client-server architecture described below.

The PHYSIO DSS Component is based on a client-server architecture (see Figure 1-1) delivered as Web Services. The PHYSIO DSS Webservice runs on a LAMP (Linux, Apache, MySQL and PHP) server located at the CNR-IASI Biomathematics Laboratory, and exposes its functionalities via WSDL (Web Services Description Language), exploiting the SOAP (Simple Object Access Protocol) [6] messaging protocol, at the url:

https://biomatlab.iasi.cnr.it/SearchAndRescue/SearchAndRescue.wsdl

SOAP is a neutral messaging protocol, based on XML (Extensible Markup Language) [7], which allows to exchange structured information.

The PHYSIO DSS Web Service natively supports interoperable machine-to-machine (M2M) interaction over a network, since its interface is described in a machine-processable format. No further interoperability constraints are imposed by our architecture. Virtually any language can be used to program a client, provided that it interrogates the server following the public interface of the webservice, respecting the SOAP specifications, with the requests and the response being exchanged in XML language.

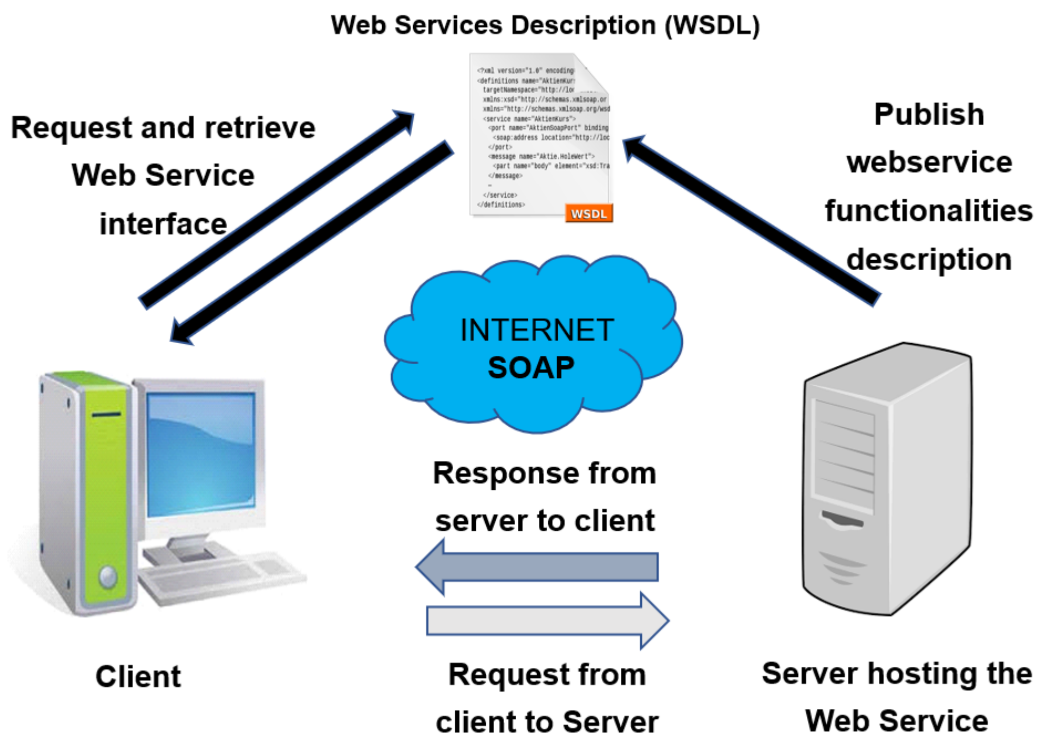

Web Services Description (WSDL)
WSDL
Request and retrieve Web Service interface
Publish webservice functionalities description
INTERNET
SOAP
Response from server to client
Request from client to Server
Client
Server hosting the Web Service

Figure 1-1: Overview of the client-server architecture of the PHYSIO DSS component

In the following section, we report for each PHYSIO DSS service the corresponding client call. The calls to the webservices were programmed in MATLAB, since MATLAB was the programming language used in the testing setting of the component. This choice allowed us to perform a more direct verification process when comparing results returned from the local and remote environments.

2 The client calls to the PHYSIO DSS Component

2.1 General specifications

In the following we describe the implementation of a MATLAB client for calls to PHYSIO DSS services. We will present a main script that makes a series of calls to all the "remote" version functions which, in turn, call the respective PHYSIO DSS services available at the CNR Server, as well as the code for each function. The term "remote" was introduced to distinguish the "local" versions, also developed in MATLAB, and which perform all the computations locally, to the versions running on the Server. The purpose of this deliverable, therefore, is to provide all the technical specifications to allow any client to make calls to each PHYSIO DSS service. The structure of the main script (**Services_call**) is very simple: after running the **Setting.m** script, the necessary inputs are defined for each "remote" function (one for each PHYSIO DSS service) and the "remote" function is called with the corresponding inputs. The content of each MATLAB "remote" function is standard: the inputs are inserted into a string vector (PieString) which is then enlarged to include the mandatory string '**Rescue function_name**'. The other important element is the instruction

```
Results_server=parse_results(CNR_services_instance,test_string);
```

The above instruction calls the "`parse_results`" MATLAB function which performs the actual call to the webservices:

```
return_string=CallSearchAndRescueCNR(CNR_services_instance,test_string);
```

The calls to the webservices differentiate according to the content of the "test_string". `CallSearchAndRescueCNR` is instead a web service method of the class constructor "CNR_services" (see the subsection below). The next subsections report the MATLAB client program for calls to each PHYSIO DSS service. Actually, even though the Physio DSS offers a series of services, all the services are "embedded" into a unique service **Rescue** which, according to the **function_name** returns the output of a specific service (compute_a_priori, compute_a_posteriori_given_treatment, etc…).

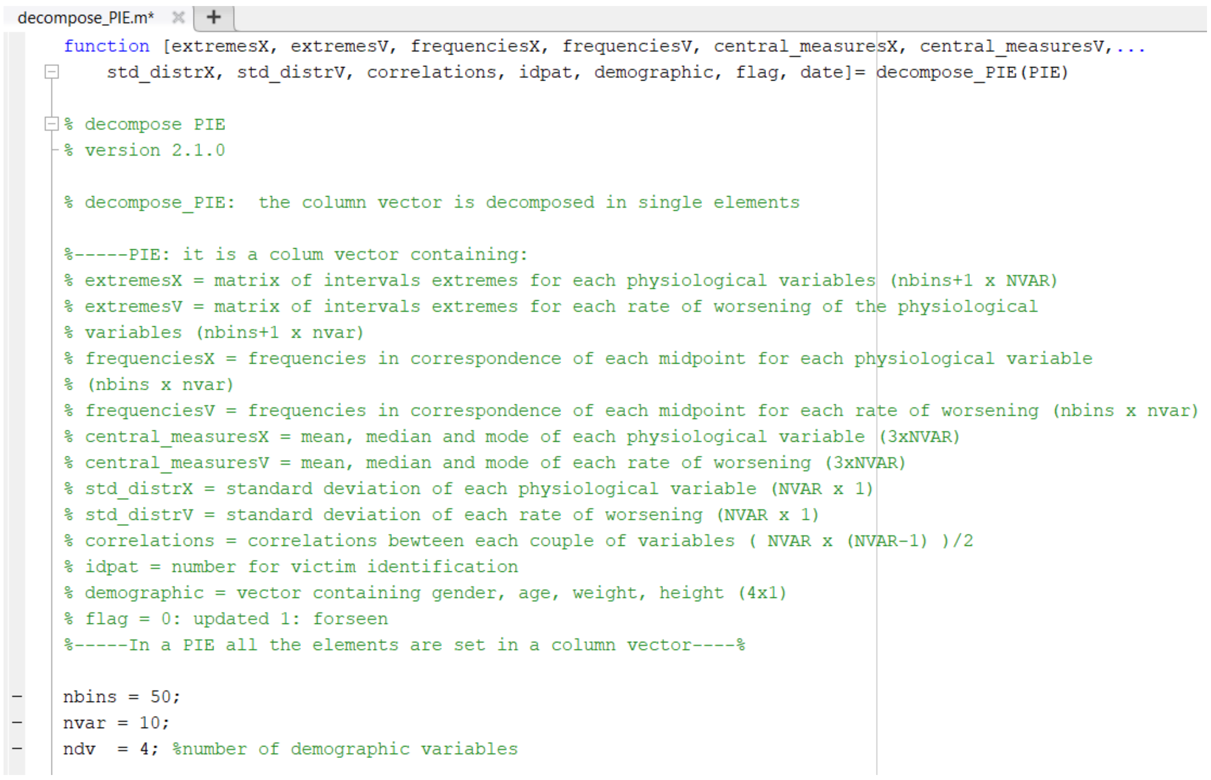
Therefore, each client, must know the exact name of the functions residing on the Server and called by the **Rescue** service. The following table shows the exact name of the functions, the necessary input parameters, the outputs, as well as a brief description of the service offered. The functions reported in the table are only the "external" ones, that is those which return a PHYSIO DSS service as output. Therefore, the table does not show all the other utility ("internal") functions, that is functions that support the functioning of the services but that cannot be called remotely.

Function name on the Server	Description	Input parameters	Output
ComputeAprioriGimelTuple	Function generating the a-priori distribution of the physiological variables from the very few initial information on the victim	*idpat* *demographic* (gender, age, weight, height) *longitudinal_deviation* *latitudinal_deviation* *event_type* *event_dimension* *date*	*PIE_remote* (a long coded string, a vector)
ETDPIEVector	Function that computes the empirical distribution of the expected time to death for each victim of the incident, taking in input the current PIE	*PIE_remote*	*extremeETD* (a 51 element vector of double, representing the extremes of the intervals of the empirical distribution) *frequenciesETD* (a 50 element vector of double containing the frequencies related to each interval whose extremes are contained in the extremeETD variable) *central_measuresETD* (a 3 element vector of double containing mean, median and mode) *std_distrETD* (a double element)
ComputeAposterioriGivenHealthMeas	Function generating the a-posteriori empirical distributions of the PSVs by updating the current PIE with values of health measurements from the field	*PIE_remote* *Health_measurement_ID* *Health_measurement_Value*	*PIE_remote* (a long coded string, a 2297 element vector)
compute_distribution_given_treatment	Function generating the empirical distributions of the PSVs by updating the current PIE with values of the delivered	*PIE* *Treatment_ID* *Treatment_Value*	*PIE_remote* (a long coded string, a 2297 element vector)

	treatment		
PhysioEVO	Function updating the current PIE on the basis of a free evolution of the system	*PIE_remote* *new_date*	*PIE_remote* (a long coded string, a 2297 element vector)
GlascowComaScaleComputation	Function computing the Glascow Coma Scale and returning a score between 3 and 15	*EO* (Eye Opening) *VR* (Verbal response) *MR* (Motor Response)	*GCS_score* (an integer number ranging from 3 to 15)
JumpStartTriage	Function returning the triage code according to the Jump Start algorithm	*RR* (Respiratory Rate) *PR* (Pulse Rate) *W* (can walk?) *AVPU* (Alert Verbal Pain Unresponsive)	*jumpstart_color_code* (an integer number ranging from 0 to 3)
SieveTriage	Function returning the triage code according to the Sieve algorithm	*RR* (Respiratory Rate) *PR* (Pulse Rate) *W* (can Walk?)	*sieve_color_code* (an integer number ranging from 0 to 3)
SortTriage	Function returning the triage code according to the Sort algorithm	*RR* (Respiratory Rate) *BP* (Blood Pressure) *EO* (Eye Opening) *VR* (Verbal response) *MR* (Motor Response)	*triage_sort* (an integer number ranging from 1 to 3)
StartTriage	Function returning the triage code according to the Start algorithm	*RR* (respiratory rate) *CR* (Capillary Refill) *W* (Can Walk?) *F* (Can Follow simple command)	*start_color_code* (an integer number ranging from 0 to 3)

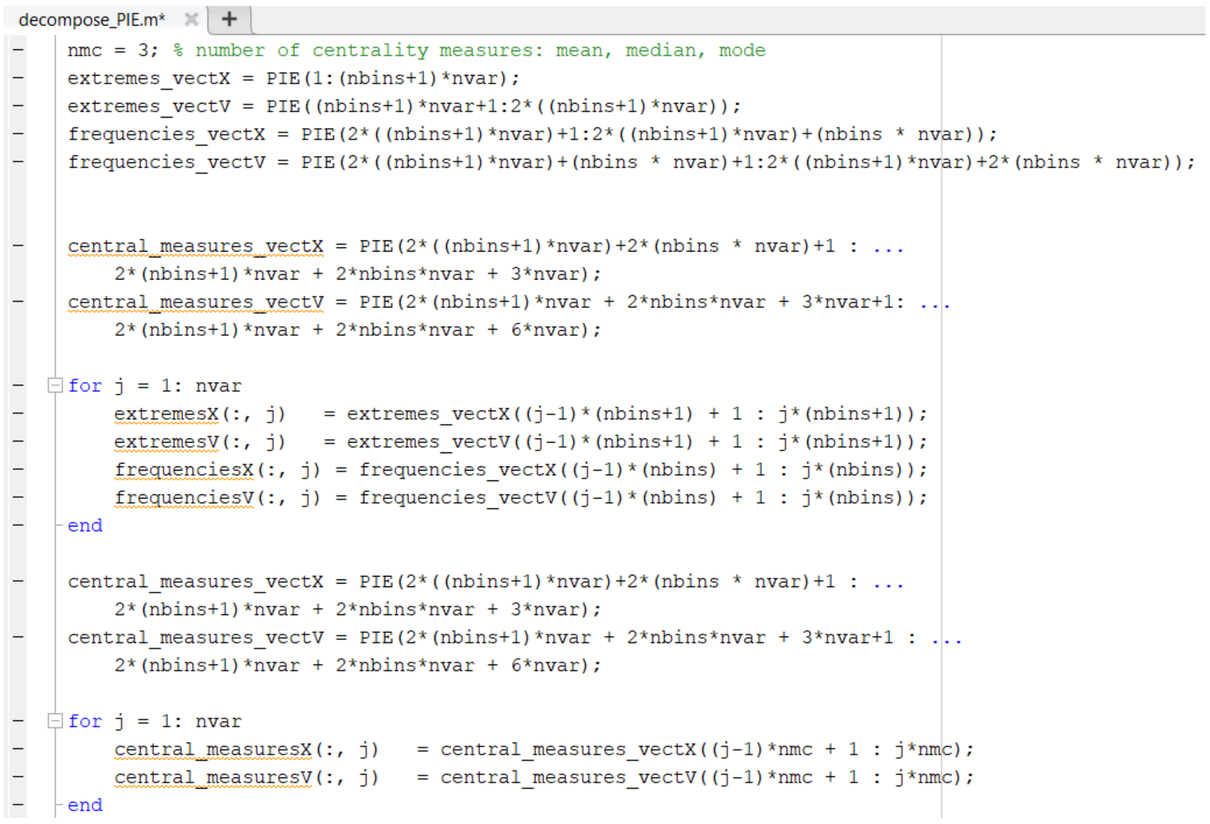
Table 2-1: PHYSIO DSS Webservices function names

The function outputs are self-explicit, with the exception of the PIE output which contains 2297 elements. A MATLAB function is then provided (the **decompose_PIE**) which is shown below. This function transforms the PIE vector into separate elements, some of which are necessary for the construction of the empirical distributions of the values of the PSVs and of the relative worsening rates. Furthermore, as for the distribution of the expected time of death, the PIE contains some elements of synthesis of the distributions such as mean, median, mode and standard deviation. Other elements are the correlations between each couple of variables, the victim's identification ID, some demographic information such as gender, age, weight and height, a flag and the date to which the PIE refers.



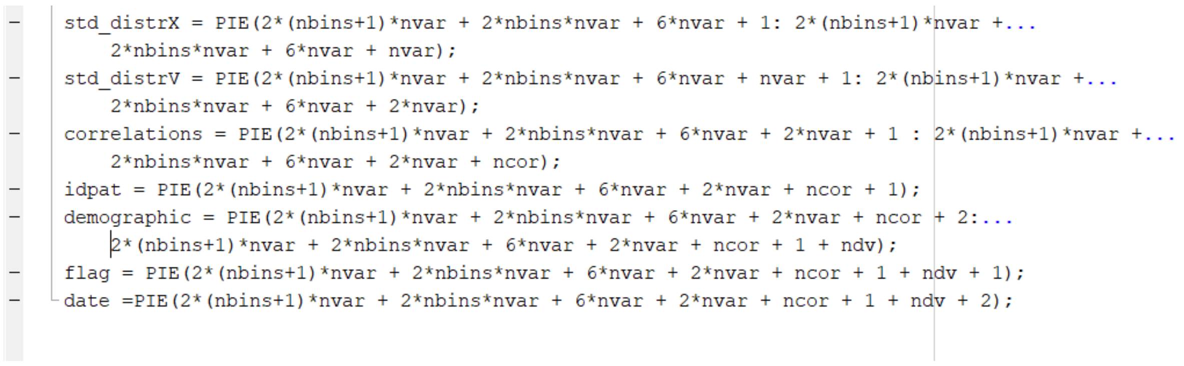
```
function [extremesX, extremesV, frequenciesX, frequenciesV, central_measuresX, central_measuresV,...
    std_distrX, std_distrV, correlations, idpat, demographic, flag, date]= decompose_PIE(PIE)

% decompose PIE
% version 2.1.0

% decompose_PIE:  the column vector is decomposed in single elements

%-----PIE: it is a colum vector containing:
% extremesX = matrix of intervals extremes for each physiological variables (nbins+1 x NVAR)
% extremesV = matrix of intervals extremes for each rate of worsening of the physiological
% variables (nbins+1 x nvar)
% frequenciesX = frequencies in correspondence of each midpoint for each physiological variable
% (nbins x nvar)
% frequenciesV = frequencies in correspondence of each midpoint for each rate of worsening (nbins x nvar)
% central_measuresX = mean, median and mode of each physiological variable (3xNVAR)
% central_measuresV = mean, median and mode of each rate of worsening (3xNVAR)
% std_distrX = standard deviation of each physiological variable (NVAR x 1)
% std_distrV = standard deviation of each rate of worsening (NVAR x 1)
% correlations = correlations bewteen each couple of variables ( NVAR x (NVAR-1) )/2
% idpat = number for victim identification
% demographic = vector containing gender, age, weight, height (4x1)
% flag = 0: updated 1: forseen
%-----In a PIE all the elements are set in a column vector----%

nbins = 50;
nvar = 10;
ndv  = 4; %number of demographic variables
```

Figure 2-2: Code related to the "decompose_PIE.m" script, part 1.


decompose_PIE.m*

```
nmc = 3; % number of centrality measures: mean, median, mode
extremes_vectX = PIE(1:(nbins+1)*nvar);
extremes_vectV = PIE((nbins+1)*nvar+1:2*((nbins+1)*nvar));
frequencies_vectX = PIE(2*((nbins+1)*nvar)+1:2*((nbins+1)*nvar)+(nbins * nvar));
frequencies_vectV = PIE(2*((nbins+1)*nvar)+(nbins * nvar)+1:2*((nbins+1)*nvar)+2*(nbins * nvar));


central_measures_vectX = PIE(2*((nbins+1)*nvar)+2*(nbins * nvar)+1 : ...
    2*(nbins+1)*nvar + 2*nbins*nvar + 3*nvar);
central_measures_vectV = PIE(2*(nbins+1)*nvar + 2*nbins*nvar + 3*nvar+1: ...
    2*(nbins+1)*nvar + 2*nbins*nvar + 6*nvar);

for j = 1: nvar
    extremesX(:, j)   = extremes_vectX((j-1)*(nbins+1) + 1 : j*(nbins+1));
    extremesV(:, j)   = extremes_vectV((j-1)*(nbins+1) + 1 : j*(nbins+1));
    frequenciesX(:, j) = frequencies_vectX((j-1)*(nbins) + 1 : j*(nbins));
    frequenciesV(:, j) = frequencies_vectV((j-1)*(nbins) + 1 : j*(nbins));
end

central_measures_vectX = PIE(2*((nbins+1)*nvar)+2*(nbins * nvar)+1 : ...
    2*(nbins+1)*nvar + 2*nbins*nvar + 3*nvar);
central_measures_vectV = PIE(2*(nbins+1)*nvar + 2*nbins*nvar + 3*nvar+1 : ...
    2*(nbins+1)*nvar + 2*nbins*nvar + 6*nvar);

for j = 1: nvar
    central_measuresX(:, j)   = central_measures_vectX((j-1)*nmc + 1 : j*nmc);
    central_measuresV(:, j)   = central_measures_vectV((j-1)*nmc + 1 : j*nmc);
end
```

Figure 2-3: Code related to the “decompose_PIE.m” script, part 2.



```
std_distrX = PIE(2*(nbins+1)*nvar + 2*nbins*nvar + 6*nvar + 1: 2*(nbins+1)*nvar +...
    2*nbins*nvar + 6*nvar + nvar);
std_distrV = PIE(2*(nbins+1)*nvar + 2*nbins*nvar + 6*nvar + nvar + 1: 2*(nbins+1)*nvar +...
    2*nbins*nvar + 6*nvar + 2*nvar);
correlations = PIE(2*(nbins+1)*nvar + 2*nbins*nvar + 6*nvar + 2*nvar + 1 : 2*(nbins+1)*nvar +...
    2*nbins*nvar + 6*nvar + 2*nvar + ncor);
idpat = PIE(2*(nbins+1)*nvar + 2*nbins*nvar + 6*nvar + 2*nvar + ncor + 1);
demographic = PIE(2*(nbins+1)*nvar + 2*nbins*nvar + 6*nvar + 2*nvar + ncor + 2:...
    2*(nbins+1)*nvar + 2*nbins*nvar + 6*nvar + 2*nvar + ncor + 1 + ndv);
flag = PIE(2*(nbins+1)*nvar + 2*nbins*nvar + 6*nvar + 2*nvar + ncor + 1 + ndv + 1);
date =PIE(2*(nbins+1)*nvar + 2*nbins*nvar + 6*nvar + 2*nvar + ncor + 1 + ndv + 2);
```

Figure 2-4: Code related to the “decompose_PIE.m” script, part 3.

2.2Settings

Before calling the services, it is necessary to define some variables and parameters for the correct connection to the webservice, in particular it is necessary to set the URL of the WSDL interface. MATLAB is able to create a Web Service class based on a WSDL application programming interface (API).

The createClassFromWsdl() creates a MATLAB class based on the service name defined as input in the function. The execution of the createClassFromWsdl creates a class folder, @servicename, in the current folder. The class folder contains:

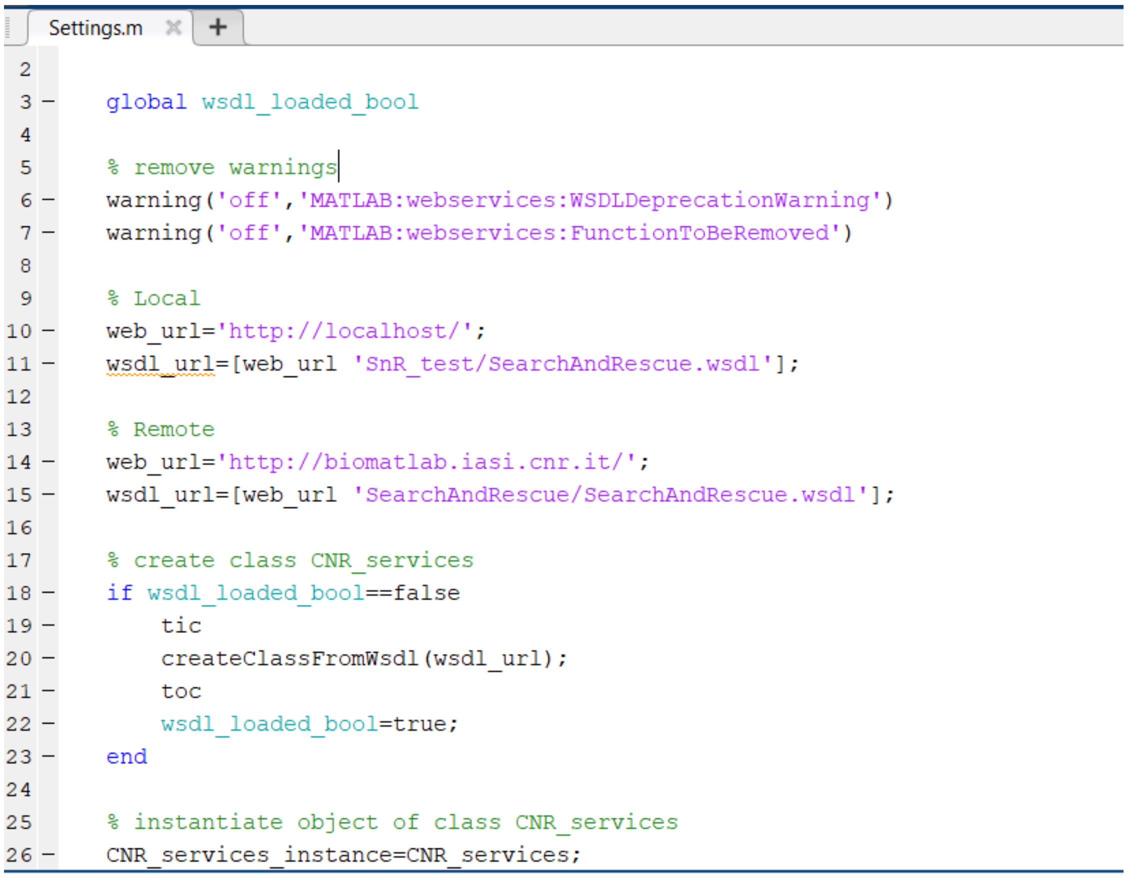
- A method file for each Web service operation.
- A display method, display.m.
- A constructor, servicename.m.

The latter is used to create an instance of the class, which we call CNR_services_instance and which contains the following two objects:

endpoint: 'http://biomatlab.iasi.cnr.it/SearchAndRescue/services.php'

wsdl: 'http://biomatlab.iasi.cnr.it/SearchAndRescue/SearchAndRescue.wsdl'

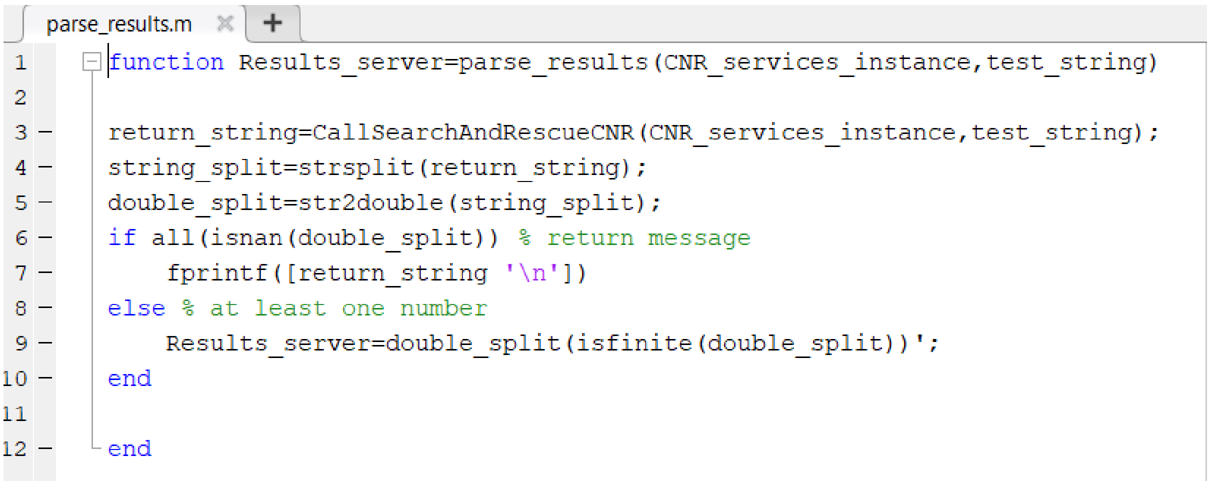
Figure 2-5: Code related to the “Settings.m” script.shows the MATLAB code included into the “Setting.m” script.



```
Settings.m

    global wsdl_loaded_bool

    % remove warnings
    warning('off','MATLAB:webservices:WSDLDeprecationWarning')
    warning('off','MATLAB:webservices:FunctionToBeRemoved')

    % Local
    web_url='http://localhost/';
    wsdl_url=[web_url 'SnR_test/SearchAndRescue.wsdl'];

    % Remote
    web_url='http://biomatlab.iasi.cnr.it/';
    wsdl_url=[web_url 'SearchAndRescue/SearchAndRescue.wsdl'];

    % create class CNR_services
    if wsdl_loaded_bool==false
        tic
        createClassFromWsdl(wsdl_url);
        toc
        wsdl_loaded_bool=true;
    end

    % instantiate object of class CNR_services
    CNR_services_instance=CNR_services;
```

Figure 2-5: Code related to the “Settings.m” script.

The function below takes in input the object CNR_services_instance and the actual service name saved into the “test_string” variable and performs the call to the server throw the **CallSearchAndRescueCNR.m** function automatically created by MATLAB with the execution of the createClassFromWsdl MATLAB function. The result is stored into the Results_server variable, which is the numeric version of the server response. The output of each call to the webservices is returned in fact into a string which is parsed inside the **parse_results** function and returned to the calling function as floating-point numbers with double precision.

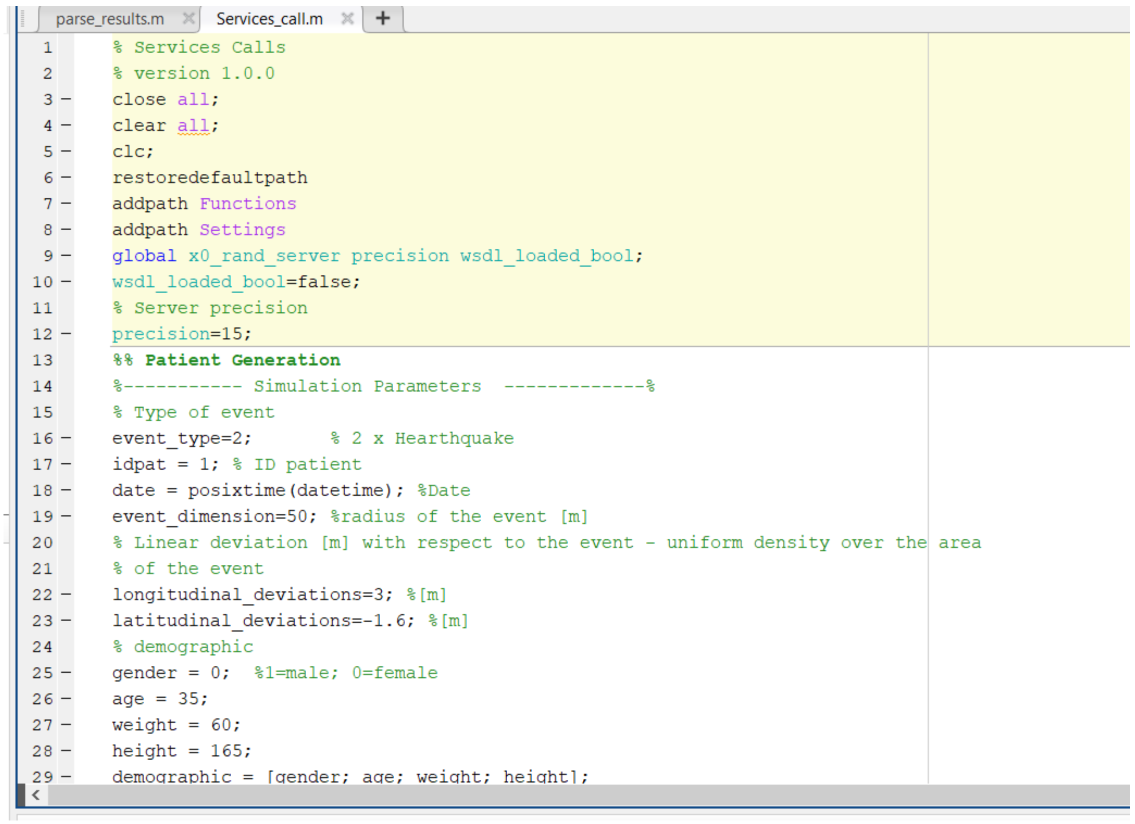

parse_results.m

```
function Results_server=parse_results(CNR_services_instance,test_string)

return_string=CallSearchAndRescueCNR(CNR_services_instance,test_string);
string_split=strsplit(return_string);
double_split=str2double(string_split);
if all(isnan(double_split)) % return message
    fprintf([return_string '\n'])
else % at least one number
    Results_server=double_split(isfinite(double_split))';
end

end
```

Figure 2-6: Code related to the "parse_results.m" function.

2.3 Call to the "compute-a-priori" PHYSIO service

The initial part of the script **Services_call.m**, reported in Figure 2-7: Code related to the "Services_call" script., set the values of a series of parameters necessary for the call to the **compute_a_priori_remote** function. This represents the very first step for the PHYSIO DSS functioning. With this call the initial (a-priori) PIE is generated based on minimal information, as the event type (i.e earthquake); the patient identification code (idpat); the exact time when the call is made (date); the extent of the event (event_dimension); the position of the victim, in terms of longitudinal and latitudinal deviations, expressed in meters, from the event location; the gender, age, weight and height of the victim.

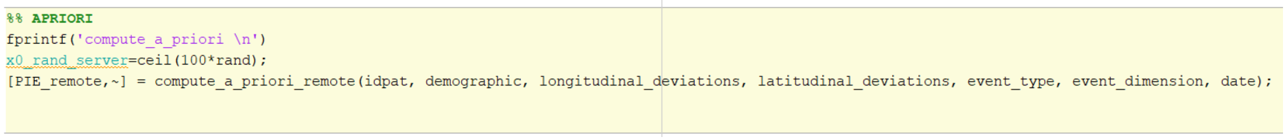


parse_results.m   Services_call.m

```
% Services Calls
% version 1.0.0
close all;
clear all;
clc;
restoredefaultpath
addpath Functions
addpath Settings
global x0_rand_server precision wsdl_loaded_bool;
wsdl_loaded_bool=false;
% Server precision
precision=15;
%% Patient Generation
%----------- Simulation Parameters  -------------%
% Type of event
event_type=2;       % 2 x Hearthquake
idpat = 1; % ID patient
date = posixtime(datetime); %Date
event_dimension=50; %radius of the event [m]
% Linear deviation [m] with respect to the event - uniform density over the area
% of the event
longitudinal_deviations=3; %[m]
latitudinal_deviations=-1.6; %[m]
% demographic
gender = 0;  %1=male; 0=female
age = 35;
weight = 60;
height = 165;
demographic = [gender; age; weight; height];
```

Figure 2-7: Code related to the "Services_call" script.

After having defined the necessary input parameters, the **Services_call.m** script performs a call to the **compute_a_priori_remote** MATLAB function:

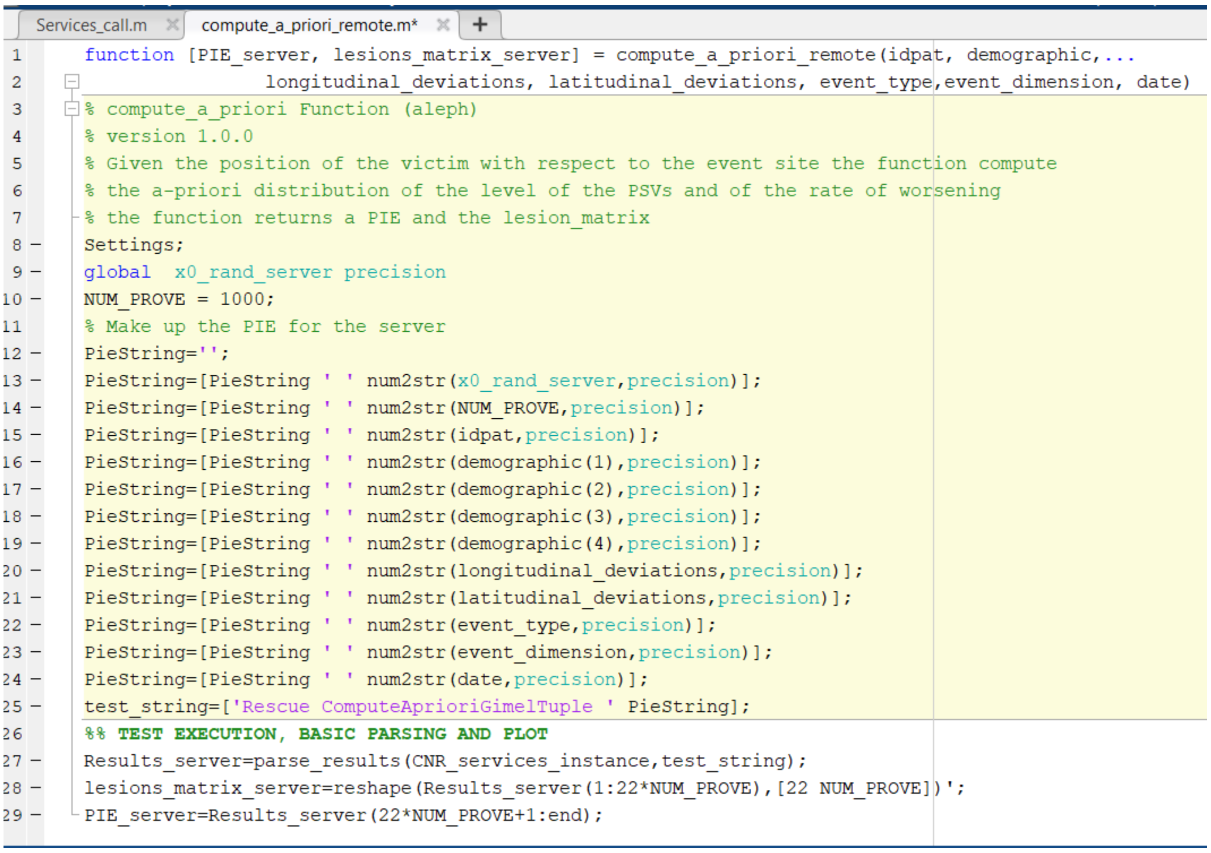


```
%% APRIORI
fprintf('compute_a_priori \n')
x0_rand_server=ceil(100*rand);
[PIE_remote,~] = compute_a_priori_remote(idpat, demographic, longitudinal_deviations, latitudinal_deviations, event_type, event_dimension, date);
```

Figure 2-8: Call to the "compute_a_priori_remote.m" function.

Note that, before each call to the "remote" function, in the current version of the component it is necessary to generate an integer random number. The instruction is represented by the code line "x0_rand_server=ceil(100*rand)". This will no longer be needed in the future and it was necessary for the sake of repeatability evaluation, i.e. to make possible the comparisons between results obtained locally and those returned remotely (webservices).

Figure 2-9: Content of the "compute_a_priori_remote.m" function. shows the content of the function. As can also be seen in the functions related to the other PHYSIO DSS services, within the "remote" type functions it is always necessary to set those parameters and variables necessary for connecting to the Server. This is done by executing the instructions contained in the **Settings.m** script, described above, which must be executed every time a call is made to a webservice. After that, a string is created, containing all the input parameters. The first part of the string to be passed to the parse_results function must contain the call to the exact function residing on the server: **Recsue ComputeAprioriGimelTuple** for the service generating the a priori distribution of the physiological state of the victim. As described above it will be the **parse_results** function to perform the actual call to the service by invoking the **CallSearchAndRescueCNR** method.



Services_call.m   compute_a_priori_remote.m*

```
function [PIE_server, lesions_matrix_server] = compute_a_priori_remote(idpat, demographic,...
                longitudinal_deviations, latitudinal_deviations, event_type,event_dimension, date)
% compute_a_priori Function (aleph)
% version 1.0.0
% Given the position of the victim with respect to the event site the function compute
% the a-priori distribution of the level of the PSVs and of the rate of worsening
% the function returns a PIE and the lesion_matrix
Settings;
global  x0_rand_server precision
NUM_PROVE = 1000;
% Make up the PIE for the server
PieString='';
PieString=[PieString ' ' num2str(x0_rand_server,precision)];
PieString=[PieString ' ' num2str(NUM_PROVE,precision)];
PieString=[PieString ' ' num2str(idpat,precision)];
PieString=[PieString ' ' num2str(demographic(1),precision)];
PieString=[PieString ' ' num2str(demographic(2),precision)];
PieString=[PieString ' ' num2str(demographic(3),precision)];
PieString=[PieString ' ' num2str(demographic(4),precision)];
PieString=[PieString ' ' num2str(longitudinal_deviations,precision)];
PieString=[PieString ' ' num2str(latitudinal_deviations,precision)];
PieString=[PieString ' ' num2str(event_type,precision)];
PieString=[PieString ' ' num2str(event_dimension,precision)];
PieString=[PieString ' ' num2str(date,precision)];
test_string=['Rescue ComputeAprioriGimelTuple ' PieString];
%% TEST EXECUTION, BASIC PARSING AND PLOT
Results_server=parse_results(CNR_services_instance,test_string);
lesions_matrix_server=reshape(Results_server(1:22*NUM_PROVE),[22 NUM_PROVE])';
PIE_server=Results_server(22*NUM_PROVE+1:end);
```

Figure 2-9: Content of the "compute_a_priori_remote.m" function.

Once the response from the server has been parsed and transformed into an array of double (operation performed by the **parse_results** function which returns the vector "Results_server"), the last two code lines arrange in the correct way the content of the Results_server vector into the final output of the **compute_a_priori_remote** function. The main output is the PIE_server vector, which will constitute the input for the other PHYSIO DSS services.

At this point, the **decompose_PIE** function must be called to decompose the PIE into the elements necessary to visualize the output and to obtain measures of synthesis of the a-priori distribution.

```
[extremesX, extremesV, frequenciesX, frequenciesV, central_measuresX, cen-
tral_measuresV, std_distrX, std_distrV, correlations, idpat, demographic, flag,
date]= decompose_PIE(PIE_server)
```

The `extremesX` matrix and the `frequenciesX` matrix, as well as the matrices `extremesV` and `frequenciesV`, include, for each variable, the extremes of the intervals and the relative frequencies for building the histograms. In fact, the midpoint of each interval can be computed and a bar graph (for each variable) can be built with bases equal to the widths of the intervals and heights equal to the frequencies. In MATLAB we used the following code:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[nbins, nvar] = size(frequenciesX);
for j = 1 : nvar
    for i = 1 : nbins
          midpointsX(i, j)  =  extremesX (i, j) + (extremesX (i+1, j) - extremesX
(i, j))/2;
    end
bar(midpointsX(:,j), frequenciesX(:,j))
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Below we report the empirical distribution of the physiological variable PSV5 "Heart pump function":

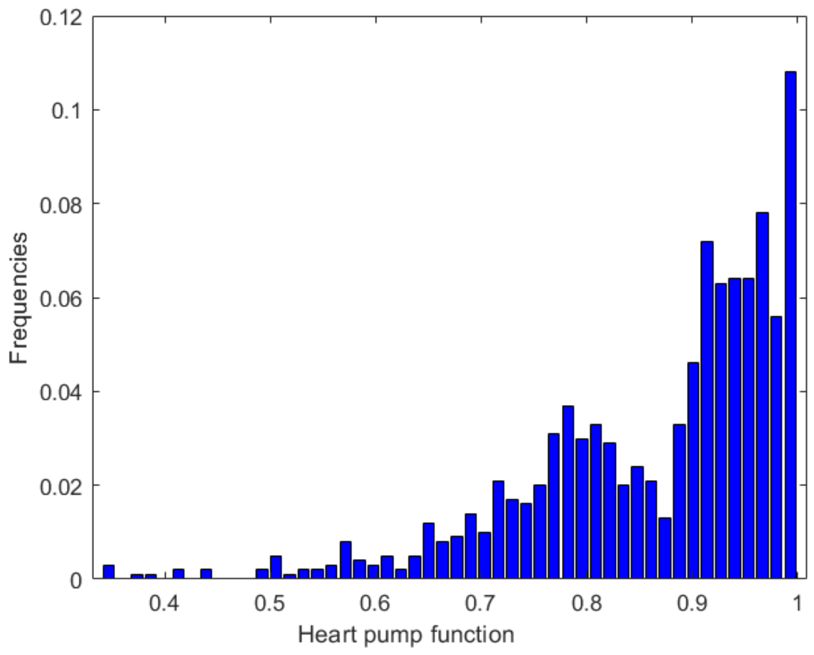
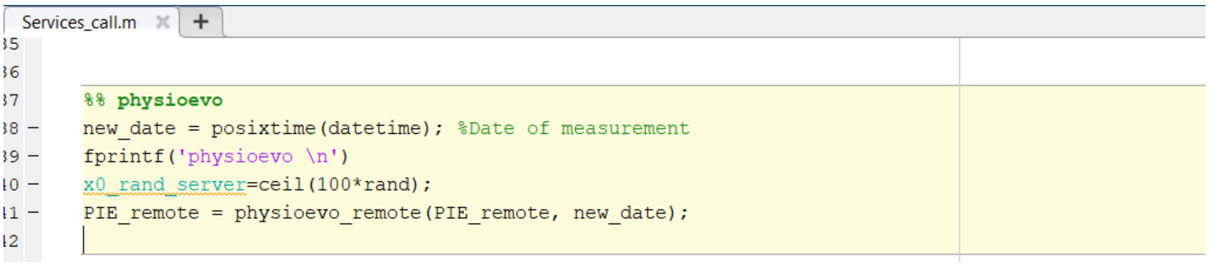


Figure 2-10: Empirical distribution of the "Heart pump function" PSV

2.4 Call to the "physioevo" PHYSIO service

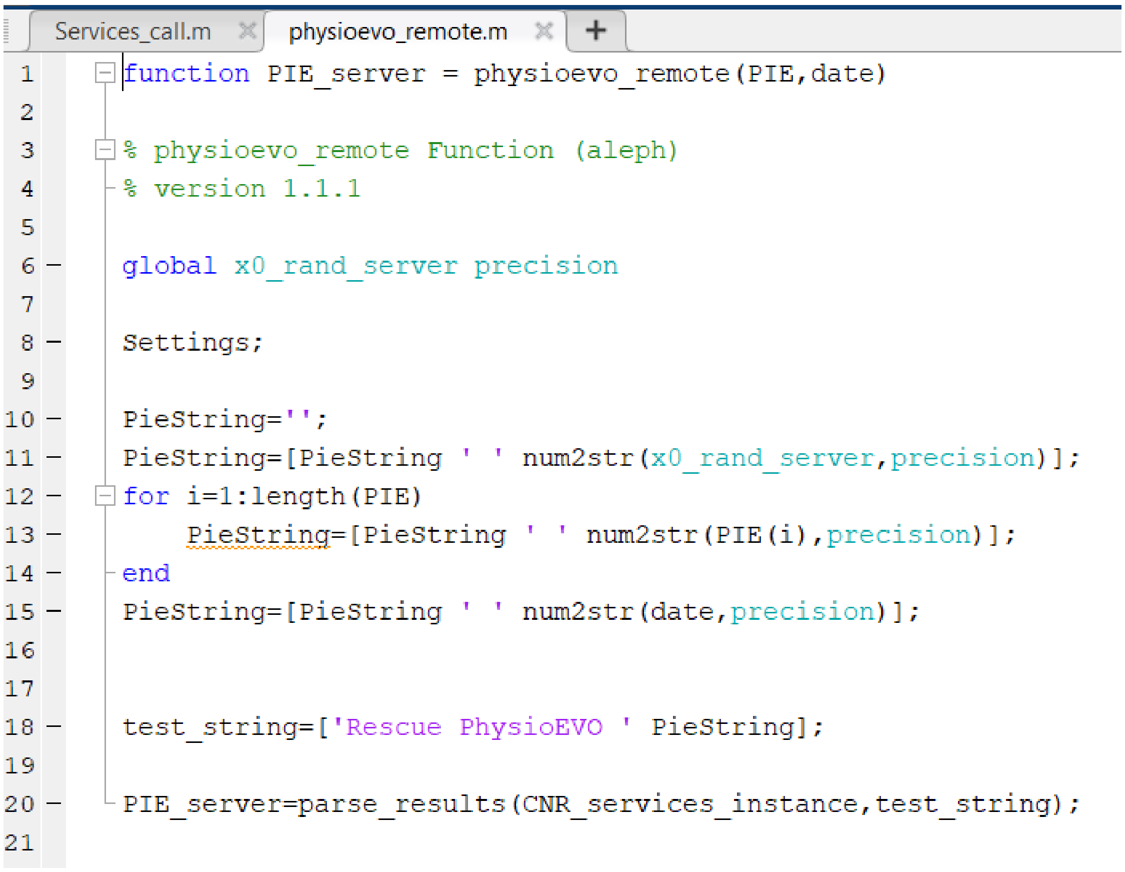
The "physioevo" service computes the new physiological distributional state of the victim based on a free evolution of the system. Given the distribution of the values of the physiological variables and of their rates of worsening, after some times, the victims will exhibit a new distribution of the values of the PSVs which, given the respective rates of worsening, and without any treatment administration, will decay to represent more impaired conditions. The inputs for the function are the actual PIE (a PIE is a long coded string, see deliverable D4.4 and D4.6 for more details) and the time at which the forecast is to be obtained (see Figure 2-11).


Services_call.m

```
%% physioevo
new_date = posixtime(datetime); %Date of measurement
fprintf('physioevo \n')
x0_rand_server=ceil(100*rand);
PIE_remote = physioevo_remote(PIE_remote, new_date);
```

Figure 2-11: Call to the "physioevo_remote.m" function.

The code of the "remote" function is reported in the figure below.


Services_call.m
physioevo_remote.m

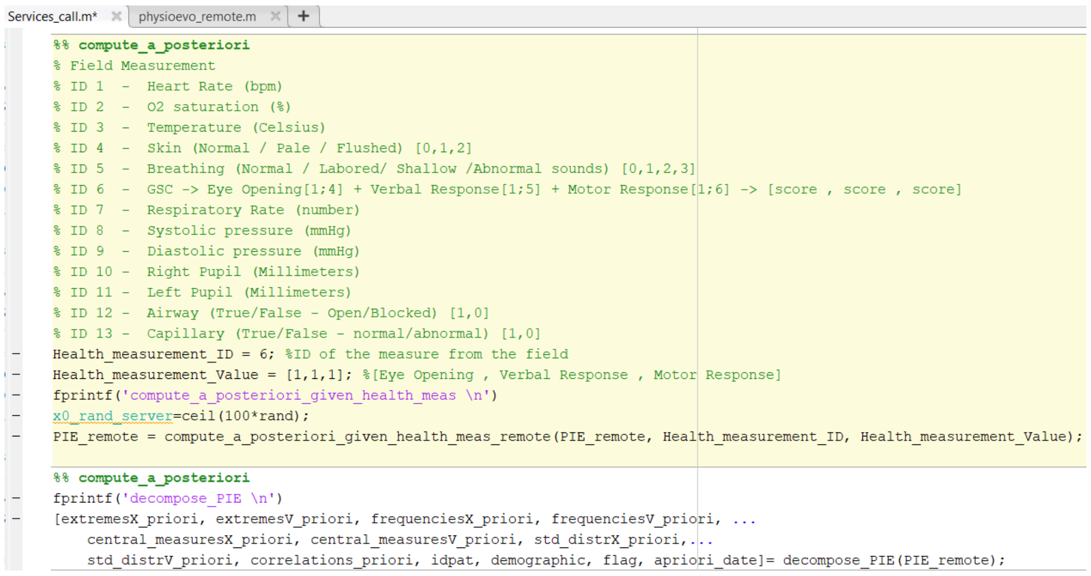
```
function PIE_server = physioevo_remote(PIE,date)

% physioevo_remote Function (aleph)
% version 1.1.1

global x0_rand_server precision

Settings;

PieString='';
PieString=[PieString ' ' num2str(x0_rand_server,precision)];
for i=1:length(PIE)
    PieString=[PieString ' ' num2str(PIE(i),precision)];
end
PieString=[PieString ' ' num2str(date,precision)];


test_string=['Rescue PhysioEVO ' PieString];

PIE_server=parse_results(CNR_services_instance,test_string);
```

Figure 2-12: Content of the "physioevo_remote.m" function.

The string to be passed to the **parse_results** function is constructed with the necessary inputs and with the call to the service **Rescue PhysioEVO**

2.5 Call to the "compute-a-posteriori" PHYSIO service

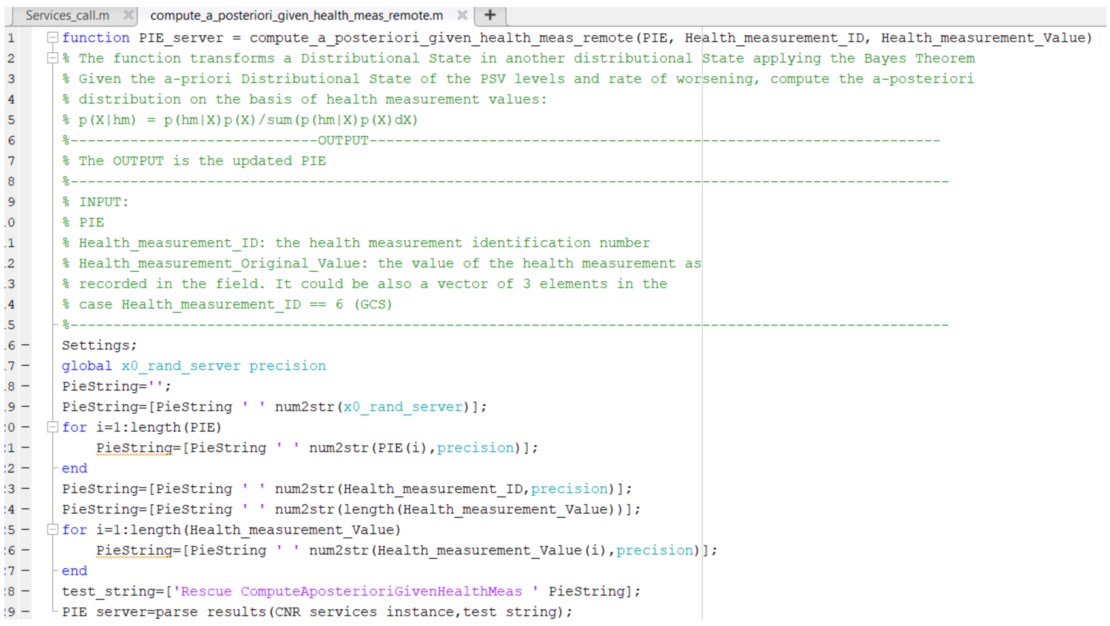
The **compute_a_aposteriori_given_health_meas** function generates the a-posteriori empirical distribution of both the values and worsening rates of the PSVs by updating the a-priori distribution, or the latest available distribution, with information from the field on the victim's health status. It takes as inputs the current PIE, the ID of one of the measured parameters (heart rate, respiratory rate, etc…) and the corresponding value. Note that each PIE includes as its last element the exact date (day of the year; hour, minute and second) to which the PIE refers, and the PIE to be passed for the computation of the a-posteriori distribution should be the PIE returned by the **PhysioEVO** service immediately before the call to the a-posteriori service.



```
%% compute_a_posteriori
% Field Measurement
% ID 1  -  Heart Rate (bpm)
% ID 2  -  O2 saturation (%)
% ID 3  -  Temperature (Celsius)
% ID 4  -  Skin (Normal / Pale / Flushed) [0,1,2]
% ID 5  -  Breathing (Normal / Labored/ Shallow /Abnormal sounds) [0,1,2,3]
% ID 6  -  GSC -> Eye Opening[1;4] + Verbal Response[1;5] + Motor Response[1;6] -> [score , score , score]
% ID 7  -  Respiratory Rate (number)
% ID 8  -  Systolic pressure (mmHg)
% ID 9  -  Diastolic pressure (mmHg)
% ID 10 -  Right Pupil (Millimeters)
% ID 11 -  Left Pupil (Millimeters)
% ID 12 -  Airway (True/False - Open/Blocked) [1,0]
% ID 13 -  Capillary (True/False - normal/abnormal) [1,0]
Health_measurement_ID = 6; %ID of the measure from the field
Health_measurement_Value = [1,1,1]; %[Eye Opening , Verbal Response , Motor Response]
fprintf('compute_a_posteriori_given_health_meas \n')
x0_rand_server=ceil(100*rand);
PIE_remote = compute_a_posteriori_given_health_meas_remote(PIE_remote, Health_measurement_ID, Health_measurement_Value);

%% compute_a_posteriori
fprintf('decompose_PIE \n')
[extremesX_priori, extremesV_priori, frequenciesX_priori, frequenciesV_priori, ...
    central_measuresX_priori, central_measuresV_priori, std_distrX_priori,...
    std_distrV_priori, correlations_priori, idpat, demographic, flag, apriori_date]= decompose_PIE(PIE_remote);
```

Figure 2-13: Call to the "compute_a_posteriori_given_health_meas_remote.m" function.

The **decompose_PIE.m** function is an utility function which encodes the content of a PIE returning all the necessary elements to construct the empirical distributions of the values and worsening rates of each PSV. This MATLAB function runs only locally and has been made available to the technical partners along with all the scripts and functions described in the present deliverable.



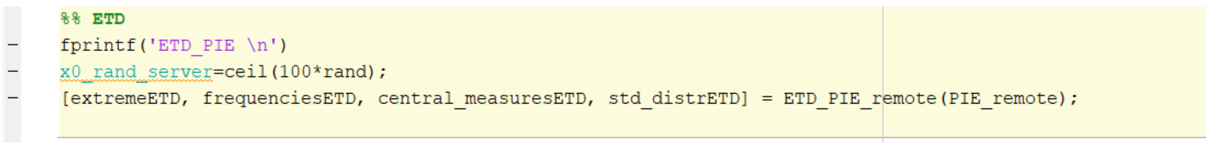
```
function PIE_server = compute_a_posteriori_given_health_meas_remote(PIE, Health_measurement_ID, Health_measurement_Value)
% The function transforms a Distributional State in another distributional State applying the Bayes Theorem
% Given the a-priori Distributional State of the PSV levels and rate of worsening, compute the a-posteriori
% distribution on the basis of health measurement values:
% p(X|hm) = p(hm|X)p(X)/sum(p(hm|X)p(X)dX)
%-----------------------------OUTPUT-----------------------------------------------------------------
% The OUTPUT is the updated PIE
%----------------------------------------------------------------------------------------------------
% INPUT:
% PIE
% Health_measurement_ID: the health measurement identification number
% Health_measurement_Original_Value: the value of the health measurement as
% recorded in the field. It could be also a vector of 3 elements in the
% case Health_measurement_ID == 6 (GCS)
%----------------------------------------------------------------------------------------------------
Settings;
global x0_rand_server precision
PieString='';
PieString=[PieString ' ' num2str(x0_rand_server)];
for i=1:length(PIE)
    PieString=[PieString ' ' num2str(PIE(i),precision)];
end
PieString=[PieString ' ' num2str(Health_measurement_ID,precision)];
PieString=[PieString ' ' num2str(length(Health_measurement_Value))];
for i=1:length(Health_measurement_Value)
    PieString=[PieString ' ' num2str(Health_measurement_Value(i),precision)];
end
test_string=['Rescue ComputeAposterioriGivenHealthMeas ' PieString];
PIE server=parse results(CNR services instance,test string);
```

Figure 2-14: Content of the "compute_a_posteriori_given_health_meas_remote.m" function.

As it is shown in the Figure 2-14 the name of the service is **Rescue ComputeAposterioriGivenHealthMeas**.

2.6 Call to the “ETD_PIE” PHYSIO service

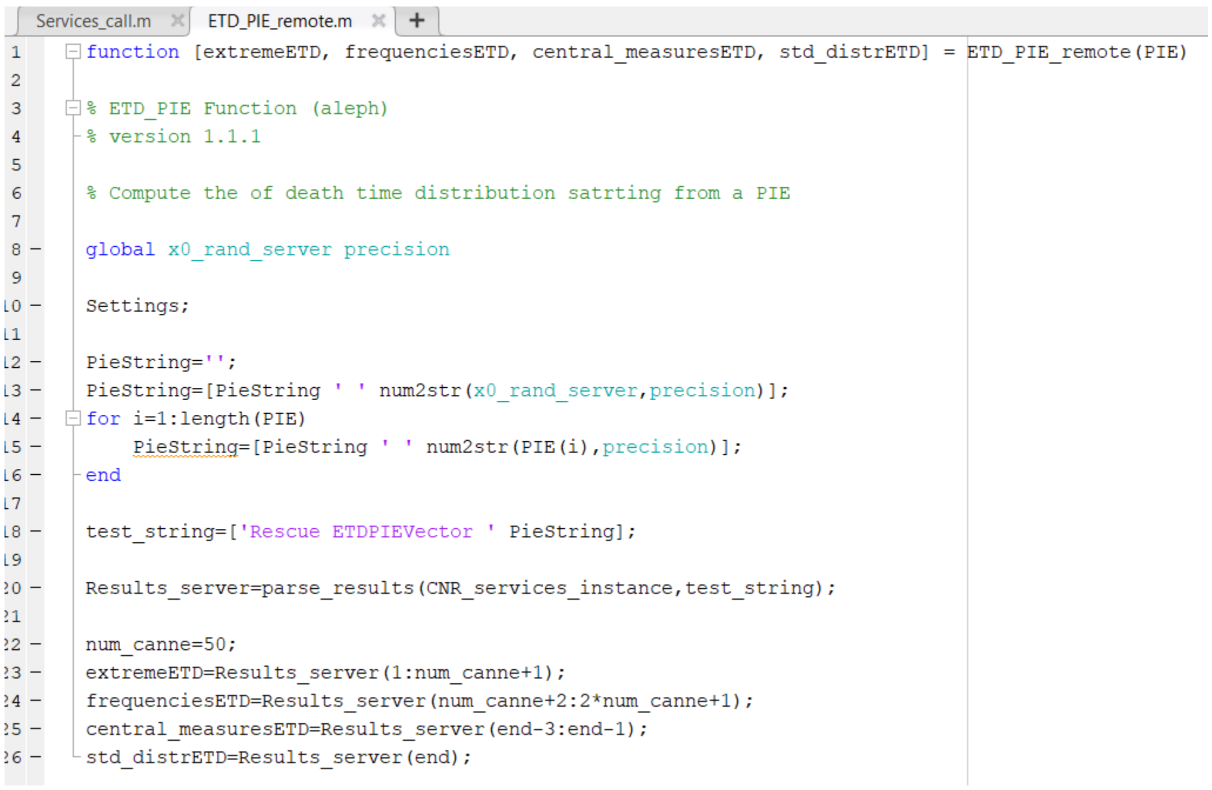
For each victim, from the PIE values, the ETD_PIE PHYSIO service computes the expected time to death. The Expected Time to Death represents the minimum time necessary for the subject to reach a value incompatible with life, by considering all the physiological variables. It is not intended in substitution to the triage algorithms used in practice, which are however provided by the PHYSIO DSS component, but can be considered as additional information for the prioritization of the victims, particularly in the situation where more victims have been triaged as red but some of them could present with more serious conditions than others. The function takes in input the latest available PIE and returns the distribution of the expected time to death in terms of extremes and frequencies of the value classes for the histogram construction. The output includes also a series of synthesis measures of the empirical distributions (mean, median, mode, standard deviation).



```
%% ETD
fprintf('ETD_PIE \n')
x0_rand_server=ceil(100*rand);
[extremeETD, frequenciesETD, central_measuresETD, std_distrETD] = ETD_PIE_remote(PIE_remote);
```

Figure 2-15: Call to the “ETD_PIE_remote.m” function

The content of the function is simple, and the service is offered by **Rescue ETDPIEVector**. After the response has been parsed, the “Results_server” vector is split into the appropriate outputs.



Services_call.m ETD_PIE_remote.m

```
function [extremeETD, frequenciesETD, central_measuresETD, std_distrETD] = ETD_PIE_remote(PIE)

% ETD_PIE Function (aleph)
% version 1.1.1

% Compute the of death time distribution satrting from a PIE

global x0_rand_server precision

Settings;

PieString='';
PieString=[PieString ' ' num2str(x0_rand_server,precision)];
for i=1:length(PIE)
    PieString=[PieString ' ' num2str(PIE(i),precision)];
end

test_string=['Rescue ETDPIEVector ' PieString];

Results_server=parse_results(CNR_services_instance,test_string);

num_canne=50;
extremeETD=Results_server(1:num_canne+1);
frequenciesETD=Results_server(num_canne+2:2*num_canne+1);
central_measuresETD=Results_server(end-3:end-1);
std_distrETD=Results_server(end);
```

Figure 2-16: Content of the “ETD_PIE_remote.m” function

The ETD output can be summarized by visualizing the central measures (mean, median and mode) and standard deviation and by plotting the empirical frequency distribution. In MATLAB, the code for creating the graph is the following:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Compute ETD middle-points
nbins = length(frequenciesETD);
for i = 1 : nbins
    midpointsETD(i)  =  extremeETD(i) + (extremeETD(i+1) - extremeETD(i))/2;
end

%% Figure ETD
bar(midpointsETD,frequenciesETD)
xlabel('Hours [h]')
ylabel('Frequencies')
title('ETD')
```

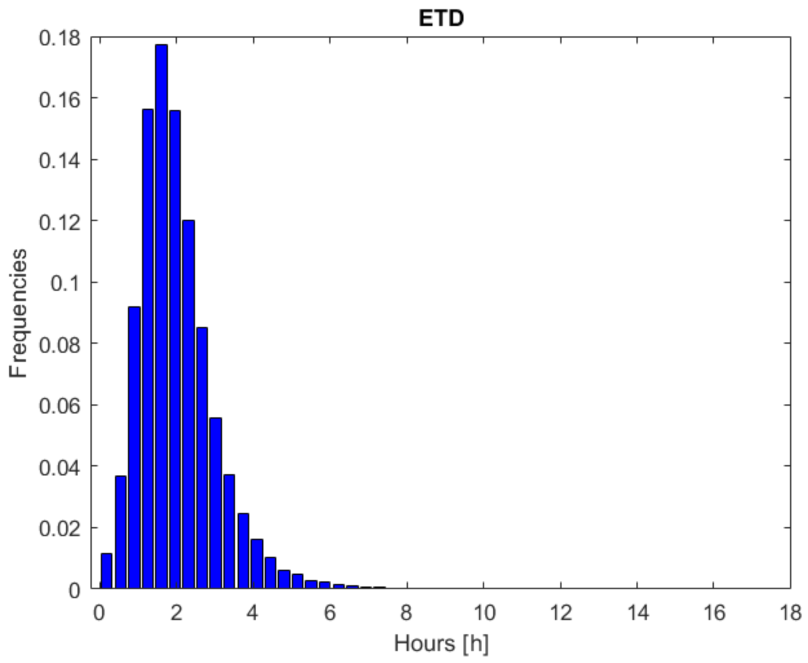
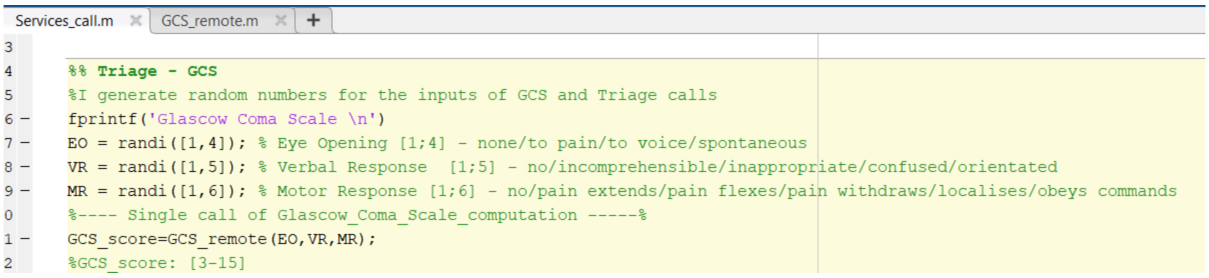


Figure 2-17: Empirical distribution of the ETD

2.7 Call to the "GCS" PHYSIO service

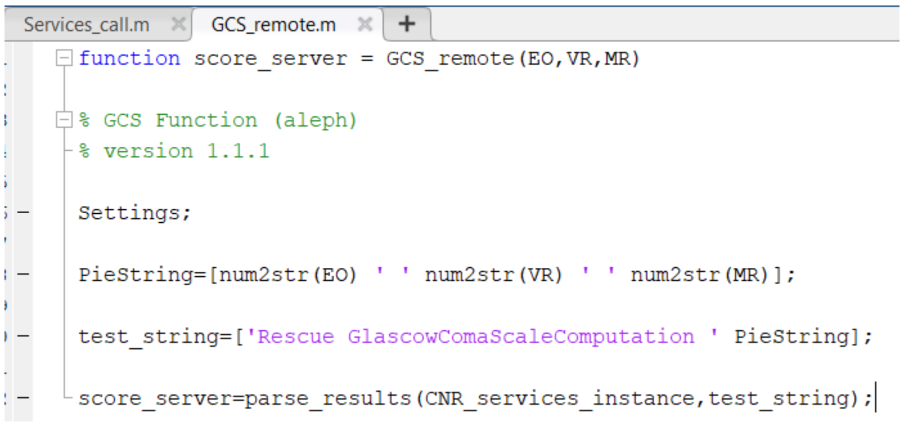
The PHYSIO DSS component offers a series of automatic computations of scores for the triage of the victims. The Glascow Coma Scale [1] is commonly used to provide an initial assessment score for patients as a marker of the severity of head injury in adults. It is widely used in the field of emergency medicine and acute trauma. This service along with other TRIAGE algorithms returns a completely deterministic response.



```
%% Triage - GCS
%I generate random numbers for the inputs of GCS and Triage calls
fprintf('Glascow Coma Scale \n')
EO = randi([1,4]); % Eye Opening [1;4] - none/to pain/to voice/spontaneous
VR = randi([1,5]); % Verbal Response  [1;5] - no/incomprehensible/inappropriate/confused/orientated
MR = randi([1,6]); % Motor Response [1;6] - no/pain extends/pain flexes/pain withdraws/localises/obeys commands
%---- Single call of Glascow_Coma_Scale_computation -----%
GCS_score=GCS_remote(EO,VR,MR);
%GCS_score: [3-15]
```

Figure 2-18: Call to the "GCS_remote.m" function

The service takes the name of **Rescue GlascowComaScaleComputation** and returns a score.

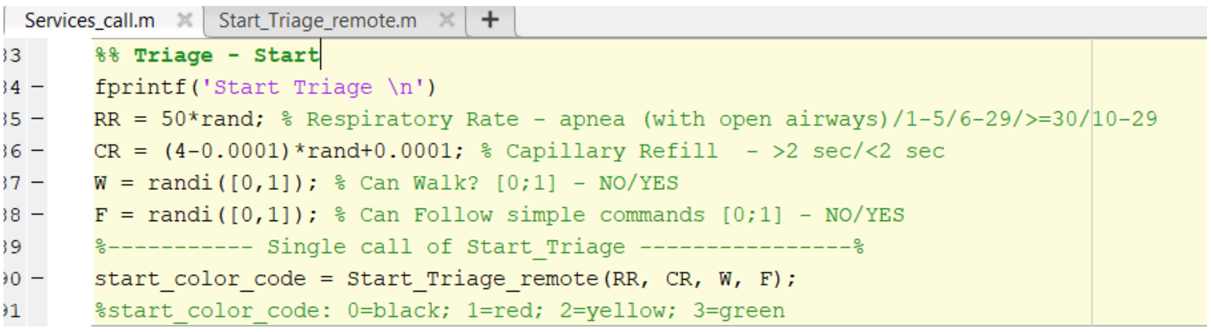


```
function score_server = GCS_remote(EO,VR,MR)

% GCS Function (aleph)
% version 1.1.1

Settings;

PieString=[num2str(EO) ' ' num2str(VR) ' ' num2str(MR)];

test_string=['Rescue GlascowComaScaleComputation ' PieString];

score_server=parse_results(CNR_services_instance,test_string);
```

Figure 2-19: Content of the "GCS_remote.m" function

2.8 Call to the "Start_Triage" PHYSIO service

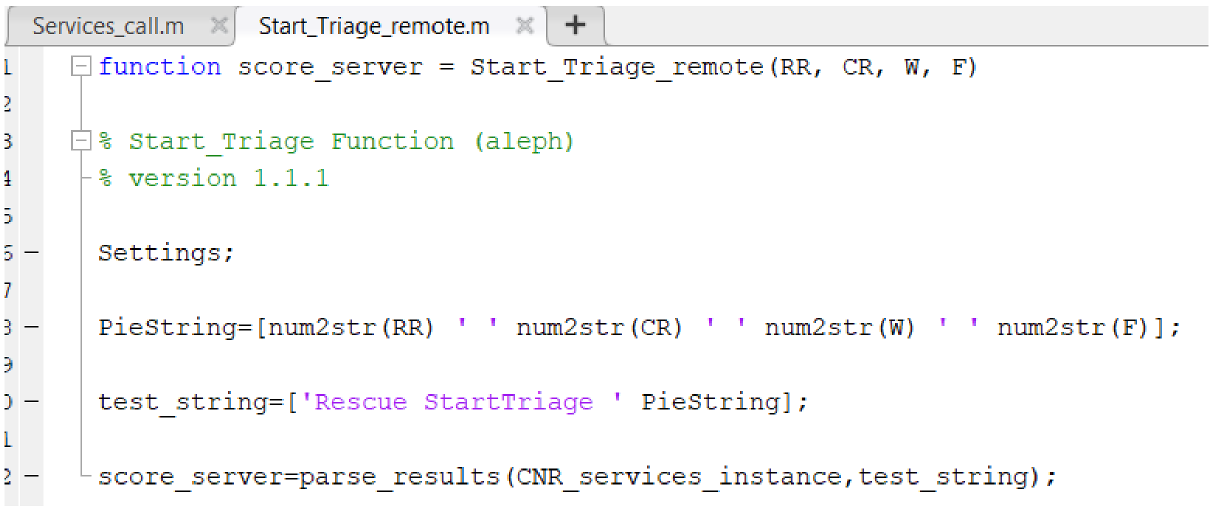
At present the Simple Triage And Rapid Treatment (START) algorithm [3] is the most commonly used mass casualty triage algorithm in the US. It takes in input the Respiratory Rate, the Capillary refill, the information about the ability to walk and to obey commands.



```
%% Triage - Start
fprintf('Start Triage \n')
RR = 50*rand; % Respiratory Rate - apnea (with open airways)/1-5/6-29/>=30/10-29
CR = (4-0.0001)*rand+0.0001; % Capillary Refill  - >2 sec/<2 sec
W = randi([0,1]); % Can Walk? [0;1] - NO/YES
F = randi([0,1]); % Can Follow simple commands [0;1] - NO/YES
%----------- Single call of Start_Triage ---------------%
start_color_code = Start_Triage_remote(RR, CR, W, F);
%start_color_code: 0=black; 1=red; 2=yellow; 3=green
```

Figure 2-20: Call to the "Start_Triage_remote.m" function

To call the service residing on the Server the string must contain the **Rescue StartTriage** information.



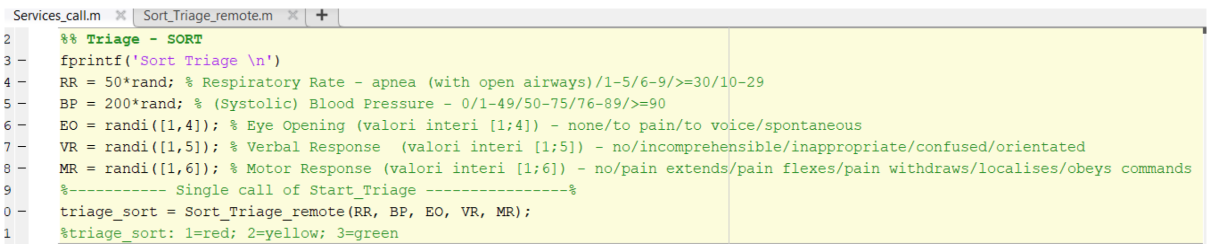
Services_call.m   Start_Triage_remote.m

```
function score_server = Start_Triage_remote(RR, CR, W, F)

% Start_Triage Function (aleph)
% version 1.1.1

Settings;

PieString=[num2str(RR) ' ' num2str(CR) ' ' num2str(W) ' ' num2str(F)];

test_string=['Rescue StartTriage ' PieString];

score_server=parse_results(CNR_services_instance,test_string);
```

Figure 2-21: Content of the “Start_Triage_remote.m” function

2.9 Call to the “SORT_Triage” PHYSIO service

The SORT triage is a revision of the Trauma Score (TS). It is based on Glasgow Coma Scale (GCS), systolic blood pressure (SBP), and respiratory rate (RR) while capillary refill and respiratory expansion, which are in general difficult to assess in the field, are not included [2].

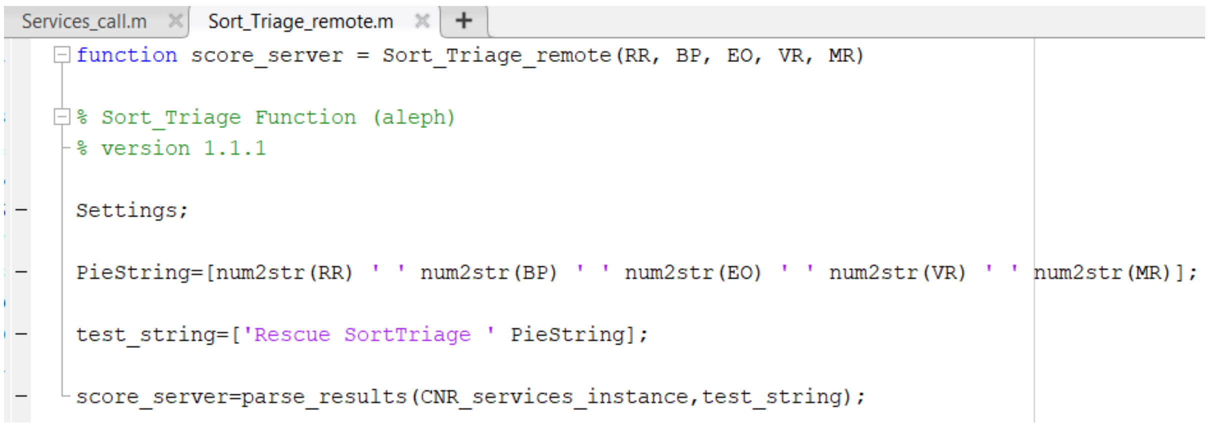


Services_call.m   Sort_Triage_remote.m

```
%% Triage - SORT
fprintf('Sort Triage \n')
RR = 50*rand; % Respiratory Rate - apnea (with open airways)/1-5/6-9/>=30/10-29
BP = 200*rand; % (Systolic) Blood Pressure - 0/1-49/50-75/76-89/>=90
EO = randi([1,4]); % Eye Opening (valori interi [1;4]) - none/to pain/to voice/spontaneous
VR = randi([1,5]); % Verbal Response  (valori interi [1;5]) - no/incomprehensible/inappropriate/confused/orientated
MR = randi([1,6]); % Motor Response (valori interi [1;6]) - no/pain extends/pain flexes/pain withdraws/localises/obeys commands
%----------- Single call of Start_Triage ----------------%
triage_sort = Sort_Triage_remote(RR, BP, EO, VR, MR);
%triage_sort: 1=red; 2=yellow; 3=green
```

Figure 2-22: Call to the “SORT_Triage_remote.m” function

The service is called with the test string: test_string=[**'Rescue SortTriage** ' PieString], where the PieString includes the inputs for the computation of the score.

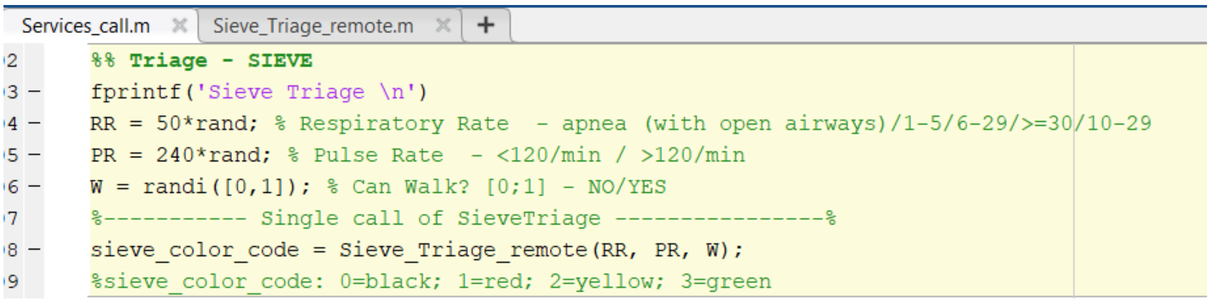


```
function score_server = Sort_Triage_remote(RR, BP, EO, VR, MR)

% Sort_Triage Function (aleph)
% version 1.1.1

Settings;

PieString=[num2str(RR) ' ' num2str(BP) ' ' num2str(EO) ' ' num2str(VR) ' ' num2str(MR)];

test_string=['Rescue SortTriage ' PieString];

score_server=parse_results(CNR_services_instance,test_string);
```

Figure 2-23: Content of the "SORT_Triage_remote.m" function

2.10 Call to the "Sieve_Triage" PHYSIO service

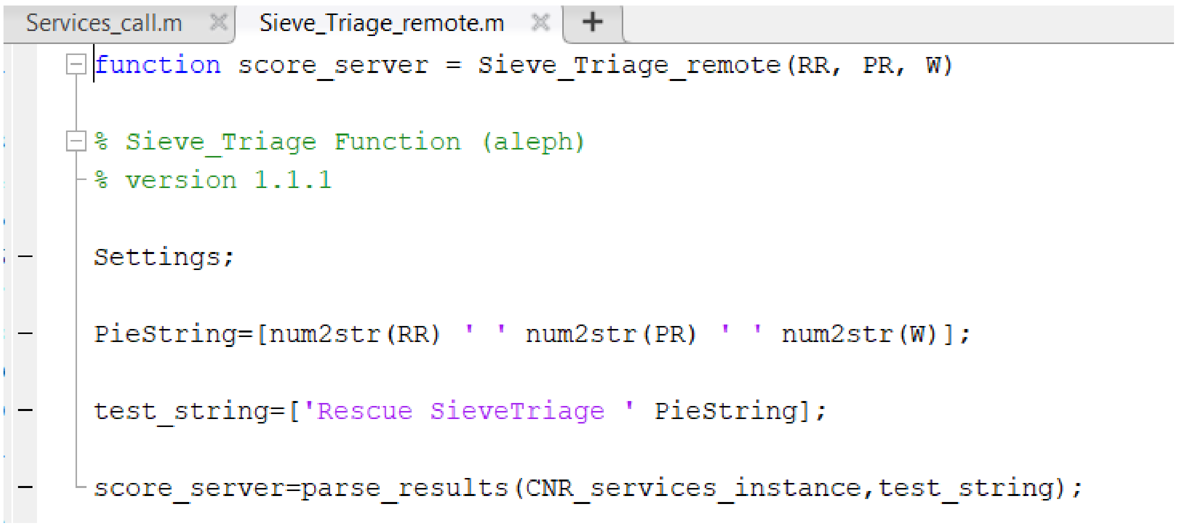
The Sieve algorithm [4] first uses the walking filter to examine the injured individual, and uses four codes (red, yellow, green, and black) to classify the injured patients, according to respiratory rate and pulse rate.



```
%% Triage - SIEVE
fprintf('Sieve Triage \n')
RR = 50*rand; % Respiratory Rate  - apnea (with open airways)/1-5/6-29/>=30/10-29
PR = 240*rand; % Pulse Rate  - <120/min / >120/min
W = randi([0,1]); % Can Walk? [0;1] - NO/YES
%----------- Single call of SieveTriage ----------------%
sieve_color_code = Sieve_Triage_remote(RR, PR, W);
%sieve_color_code: 0=black; 1=red; 2=yellow; 3=green
```

Figure 2-24: Call to the "SIEVE_Triage_remote.m" function

The call to the webservice occurs by means of the test string '**Rescue SieveTriage** '.

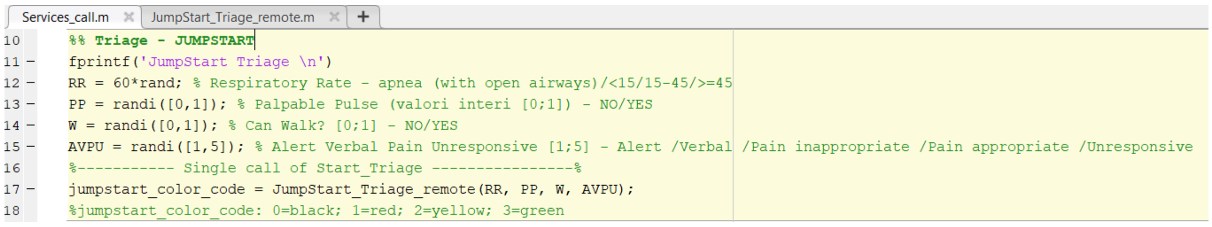


```
function score_server = Sieve_Triage_remote(RR, PR, W)

% Sieve_Triage Function (aleph)
% version 1.1.1

Settings;

PieString=[num2str(RR) ' ' num2str(PR) ' ' num2str(W)];

test_string=['Rescue SieveTriage ' PieString];

score_server=parse_results(CNR_services_instance,test_string);
```

Figure 2-25: Content of the "SIEVE_Triage_remote.m" function

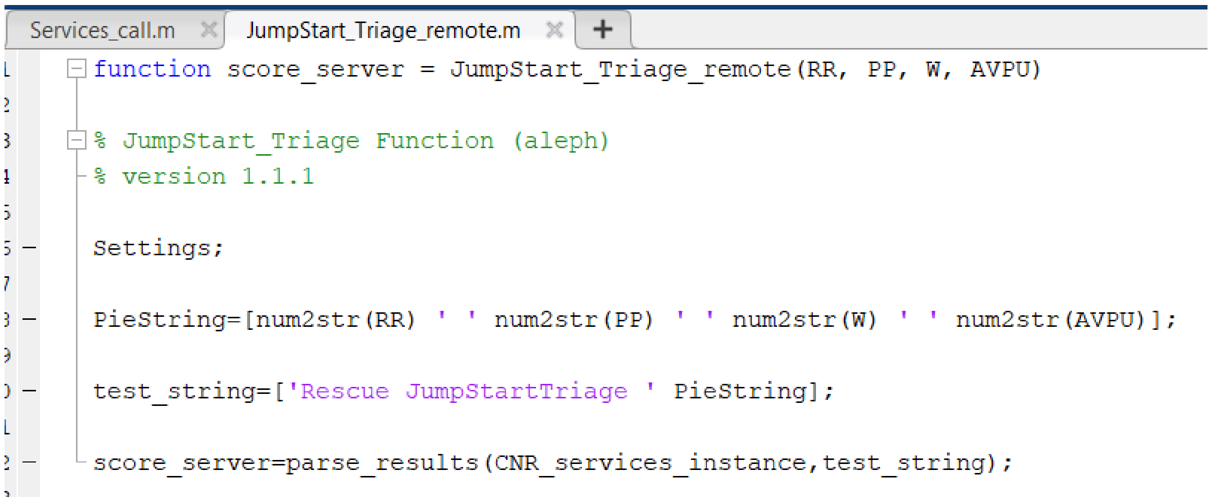
2.11 Call to the "JumpStart_Triage" PHYSIO service

This algorithm was designed by Dr Romig in 1995 for triaging children under the age of 8. In 2001, based on the principles of the START algorithm, some modifications have been made [5] to differentiate between adults and children, taking into account the higher probability of the respiratory failure in children than adults, the different breath rate in children, and the inability of the young children to follow verbal commands. The relative function name on the Server is **JumpStartTriage** and is called by means of the string **Rescue JumpStartTriage**.



```
%% Triage - JUMPSTART
fprintf('JumpStart Triage \n')
RR = 60*rand; % Respiratory Rate - apnea (with open airways)/<15/15-45/>=45
PP = randi([0,1]); % Palpable Pulse (valori interi [0;1]) - NO/YES
W = randi([0,1]); % Can Walk? [0;1] - NO/YES
AVPU = randi([1,5]); % Alert Verbal Pain Unresponsive [1;5] - Alert /Verbal /Pain inappropriate /Pain appropriate /Unresponsive
%----------- Single call of Start_Triage ----------------%
jumpstart_color_code = JumpStart_Triage_remote(RR, PP, W, AVPU);
%jumpstart_color_code: 0=black; 1=red; 2=yellow; 3=green
```

Figure 2-26: Call to the "JumpStart_Triage_remote.m" function

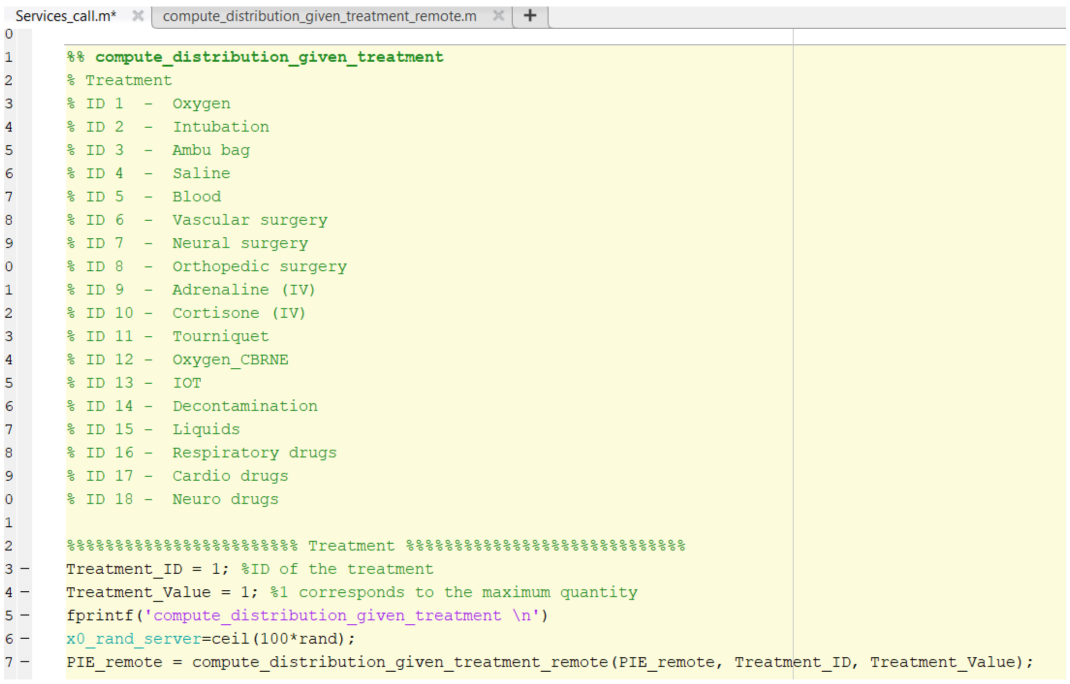


```
function score_server = JumpStart_Triage_remote(RR, PP, W, AVPU)

% JumpStart_Triage Function (aleph)
% version 1.1.1

Settings;

PieString=[num2str(RR) ' ' num2str(PP) ' ' num2str(W) ' ' num2str(AVPU)];

test_string=['Rescue JumpStartTriage ' PieString];

score_server=parse_results(CNR_services_instance,test_string);
```

Figure 2-27: Content of the "JumpStart_Triage_remote.m" function

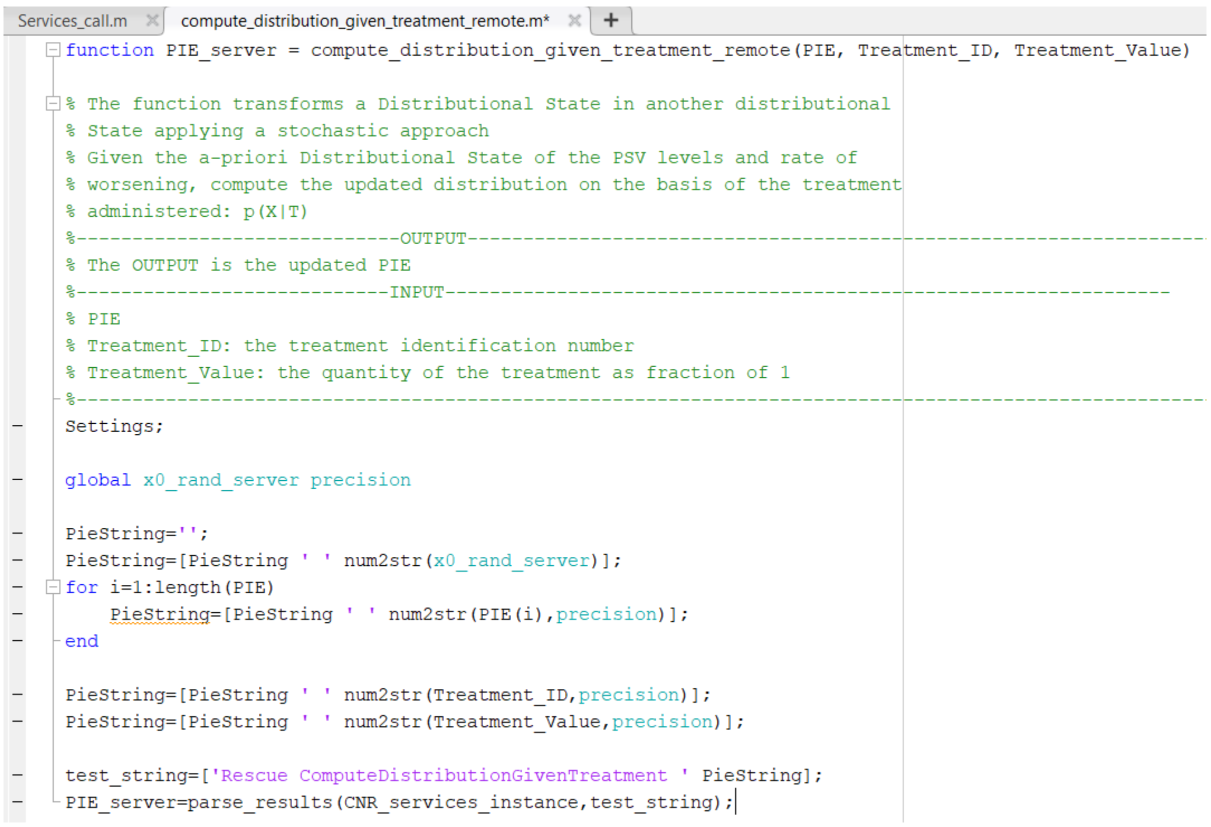
2.12 Call to the "compute_distribution_given_treatment" PHYSIO service

The script below calls the remote version of the function **compute_distribution_given_treatment** that computes an update of the current PIE on the basis of administered treatments. In the current version of the PHYSIO DSS, the considered treatments/therapies are those reported as comment in the script. The function takes in input the current PIE, the identification number of the treatment and the quantity of treatment administered as a proportion of the maximum quantity administrable quantity. The service is called by means of the string **Rescue ComputeDistributionGivenTreatment**.



```
%% compute_distribution_given_treatment
% Treatment
% ID 1  -  Oxygen
% ID 2  -  Intubation
% ID 3  -  Ambu bag
% ID 4  -  Saline
% ID 5  -  Blood
% ID 6  -  Vascular surgery
% ID 7  -  Neural surgery
% ID 8  -  Orthopedic surgery
% ID 9  -  Adrenaline (IV)
% ID 10 -  Cortisone (IV)
% ID 11 -  Tourniquet
% ID 12 -  Oxygen_CBRNE
% ID 13 -  IOT
% ID 14 -  Decontamination
% ID 15 -  Liquids
% ID 16 -  Respiratory drugs
% ID 17 -  Cardio drugs
% ID 18 -  Neuro drugs

%%%%%%%%%%%%%%%%%%%%%%%% Treatment %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Treatment_ID = 1; %ID of the treatment
Treatment_Value = 1; %1 corresponds to the maximum quantity
fprintf('compute_distribution_given_treatment \n')
x0_rand_server=ceil(100*rand);
PIE_remote = compute_distribution_given_treatment_remote(PIE_remote, Treatment_ID, Treatment_Value);
```

Figure 2-28: Call to the "compute_distribution_given_treatment_remote.m" function



```
function PIE_server = compute_distribution_given_treatment_remote(PIE, Treatment_ID, Treatment_Value)

% The function transforms a Distributional State in another distributional
% State applying a stochastic approach
% Given the a-priori Distributional State of the PSV levels and rate of
% worsening, compute the updated distribution on the basis of the treatment
% administered: p(X|T)
%----------------------------OUTPUT-------------------------------------------------------------------
% The OUTPUT is the updated PIE
%---------------------------INPUT-------------------------------------------------------------------
% PIE
% Treatment_ID: the treatment identification number
% Treatment_Value: the quantity of the treatment as fraction of 1
%-----------------------------------------------------------------------------------------------------
Settings;

global x0_rand_server precision

PieString='';
PieString=[PieString ' ' num2str(x0_rand_server)];
for i=1:length(PIE)
    PieString=[PieString ' ' num2str(PIE(i),precision)];
end

PieString=[PieString ' ' num2str(Treatment_ID,precision)];
PieString=[PieString ' ' num2str(Treatment_Value,precision)];

test_string=['Rescue ComputeDistributionGivenTreatment ' PieString];
PIE_server=parse_results(CNR_services_instance,test_string);
```

Figure 2-29: Content of the "compute_distribution_given_treatment_remote.m" function

3 Conclusions

This deliverable describes in detail how the services offered by the PHYSIO DSS component can be called by a client. The one described in this document is a MATLAB client, but any programming language can be employed once the correct specifications are provided. This deliverable therefore aims to accompany the development of the PHYSIO DSS, that is the set of webservices residing on the CNR server and which represents the real result of part of the work carried out in the context of task T4.5.

After a brief description of the component and its implementation aspects, the MATLAB code is presented to allow calls to all the external services.

This final version of the PHYSIO DSS will be tested during the next S&R Use Case 1 "Victim trapped under the rubble" which will take place on April 28 in Poggioreale, Sicily, Italy. Since Use Case 1 is the first opportunity to test the PHYSIO DSS in a real setting, any shortcomings that arise from the demo will be resolved and a new version of the component will be released to be then tested in Use Case 5.

4 References

[1] G. L. Sternbach, «The Glasgow Coma Scale,» *The Journal of Emergency Medicine,* vol. 19, n. 1, pp. 67-71, 2000.

[2] H. Champion, W. Sacco, W. Copes, D. Gann, T. Gennarelli e M. Flanagan, «A revision of the Trauma Score,» *J Trauma,* vol. 29, n. 5, pp. 623-629, 1989.

[3] M. Benson, K. Koenig e C. Schultz, «Disaster triage: START, then SAVE--a new method of dynamic triage for victims of a catastrophic earthquake,» *Prehosp Disaster Med,* vol. 11, n. 2, pp. 112-124, 1996.

[4] J. Bazyar, M. Farrokhi e H. Khankeh, «Triage Systems in Mass Casualty Incidents and Disasters: A Review Study with A Worldwide Approach,» *J Med Sci.,* vol. 7, n. 3, pp. 482-494, 2019.

[5] L. Roming, «Pediatric triage. A system to JumpSTART your triage of young patients at MCIs,» *JEMS,* vol. 27, n. 7, pp. 60-63, 2002.

[6] W3C, «Simple Object Access Protocol (SOAP) 1.1,» [Online]. Available: https://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[7] R. Richard, Pro PHP XML and Web Services (Books for Professionals by Professionals), APRESS, 2006.