Appendix C

Matlab Codes

This appendix contains the main Matlab functions written during this thesis for test-design calculations, simulations and results representation.

C.1 Codes for the test design

OptimalTestSurfaces.m

```
function [Nopt, Kopt, Eopt] = OptimalTestSurfaces(Sn_1, N2, ppint)

Nopt = zeros(length(N2), length(Sn_1)); % Optimal sample size
Eopt = Nopt;                            % Expected delay-time
Kopt = Nopt;                            % Optimal K

for i = 1:length(N2)
    [Nopt(i,:),Kopt(i,:),Eopt(i,:)] = test_optimo_ConfInterval(...
     Sn_1,N2(i), ppint);
    disp([num2str(i), ' de ', num2str(length(N2))])
end
```

test_optimo_ConfInterval.m

```
function [Nopt, Kopt, Eopt] = test_optimo_ConfInterval(S,N2,ppint)

% Study of the optimal values for N & K for the test as a function
% of S for the given distribution. Confidence interval of sigma is
% taken into account.
% N2 must be a scalar.
diary off

S = S(:)./rstd(N2); % vector columna
N = [2:50];    % vector fila
Ic = ConfidenceInterval(N2, .025);
K = 3.3418./sqrt(N) .* (1 + Ic);
K = K' * S' ; % Diada: columna por fila.
```

```
Eopt = zeros(1, length(Sn_1));
Kopt = Eopt;
Nopt = Kopt;

% First only for the 1st value of S, so we can reduce the size of
% vectors N and K in the loop when calling expectedtimeII.m. This
% is MUCH MUCH faster.

Snmin = Sn_1 .* (1-Ic);
E = expectedtimeII(0, Snmin(1), 147, Snmin(1), K(:, 1), N, ppint);
[v, a] = min(E);
Eopt(1) = v;
Nopt(1) = N(a); % Nopt(i) = N(a); would also work.
Kopt(1) = K(a, 1);
% disp([num2str(1), ' de ', num2str(length(Sn_1))])

for i = 2:length(Snmin)
    ap1 = min(a+1, length(N));
    E = expectedtimeII(0, Snmin(i), 147, Snmin(i), K(a:ap1, i), ...
     N(a:ap1), ppint);
    [v, arelativ] = min(E);
    a = a + arelativ - 1;
    Eopt(i) = v;
    Nopt(i) = N(a);
    Kopt(i) = K(a, i);
%     disp([num2str(i), ' de ', num2str(length(S))])
end
```

test_optimo.m

```
function [Nopt, Kopt, Eopt] = test_optimo(sigma, ppint)

% Study of the optimal values for N & K for the test as a
% function of S for the given distribution.

diary off

sigma = sigma(:); % column vector
N = [2:22];      % row vector
% K* = 3.3418 ==> alpha = 0.01
K = repmat((3.3418 .* sigma), 1, length(N));

Eopt = zeros(1, length(sigma));
Kopt = Eopt;
Nopt = Kopt;
```

```
for j = 1:length(N)
    sqrtj = sqrt(N(j));
    K(:,j) = K(:,j) / sqrtj;
end

E = expectedtimeII(0, sigma(1), 147, sigma(1), K(1,:), N, ppint);
[v, a] = min(E);
Eopt(1) = v;
Nopt(1) = N(a); % Tbn sirve: Nopt(i) = N(a);
Kopt(1) = K(1,a);
disp([num2str(1), ' de ', num2str(length(sigma))])

for i = 2:length(sigma)
    E = expectedtimeII(0, sigma(i), 147, sigma(i), K(i,a:a+1),...
     N(a:a+1), ppint);
    [v, arelativ] = min(E);
    a = a + arelativ - 1;
    Eopt(i) = v;
    Nopt(i) = N(a);
    Kopt(i) = K(i,a);
    disp([num2str(i), ' de ', num2str(length(sigma))])
end
```

expectedtimeII.m

```
function t = expectedtimeII(m1, s1, m2, s2, lim, N, ppint)

% Computes the expected number of experiments necessary for
% the mean of a N-size sample to be higher than LIM when the
% population parameters change from {m1, s1} to {m2, s2}
%
% It is prepared for a single value of s1 and s2 and vec-
% torial values of lim and N in order to make calculations
% for many N values with a constant value of alpha.

N = round(N);
var1 = s1 ^ 2; % var "uno"
var2 = s2 ^ 2;
if all(size(lim) == [1 1]) & any(size(N) ~= [1 1])
    lim = repmat(lim, size(N, 1), size(N, 2));
end

deltaMu = (m1-m2);
deltaMu2 = deltaMu * deltaMu;
t = zeros(1, length(N));
```

```
for j = 1:length(N)
    p = zeros (1, N(j));
    m1i = m1 * N(j);
    m2i = 0;
    var1i = var1 * N(j);
    var2i = 0;
    for i = 1:N(j)
        Nj_i = N(j)-i;
        m1i = m1i - m1;
        m2i = m2i + m2;
        var1i = var1i - var1;
        var2i = var2i + var2;
%
        meanvaluei = (m1i + m2i) / N(j);
        stdi = sqrt( (var1i + var2i) ) / (N(j));


        %p(i) = 1 - cdf('Normal', lim(j), meanvaluei, stdi);
        % We change this line to use the empirical distribution
        p(i) = 1 - mycdf(ppint, lim(j), meanvaluei, stdi);
    end

    % Mean of a geometric distribution with non-constant "p":
    t(j) = pascalemean(p);

%disp(['exptimeII N = ', num2str(N(j)), ' => p = ', num2str(p)])
end
```

pascalemean.m

```
function m = pascalemean(p)

% Mean of a geometric distribution with a non-constant "p".

p=[p, p(end)]; % p must be a row vector;
q= 1-p;
n = length(p);
product = zeros(1, n);

delta = 1;
s1 = 0; s2 = 0;
oldvalue = 0;
% oldvalue and newvalue will contain the partial current
% value of the sum at each iteration.

product(1) = 1;
for j = 2:n
```

```
    product(j) = product(j-1) * q(j-1);
end


for i=1:n
    pi_X_producti = p(i) * product(i);
    s1 = s1 + (i * pi_X_producti);
    s2 = s2 + (pi_X_producti);
end
newvalue = s1 / s2;

lastproduct = product(n);
i=n + 1;
while ((delta > eps) & ((i-n) < 3000)) | ((i-n) < 100)
    % Minimum n + 100 iterations, maximum n + 3000

%     s1 = s1 + (i * q(index).^(i-1));
%     s2 = s2 + q(index).^(i-1);

    % The loop here is for restricting the comprobation of
    % the convergence to once every 20 times. lines with
    % "for" and "end" can actually be deleted.
    for w = 1:20
    lastproduct = lastproduct * product(n);
    pn_X_lastproduct = p(n) * lastproduct;
    s1 = s1 + (i * pn_X_lastproduct);
    s2 = s2 + (pn_X_lastproduct);
    i = i + 1;
    end

    newvalue = s1 / s2;
    delta = (newvalue - oldvalue) / newvalue;
    oldvalue = newvalue;
end

% disp(['i - n = ', num2str(i - n), ', delta = ', num2str(delta)]);
% if (i-n) >= 3000
%     wstr = [':Warning: Too many iterations. The function ...
% "pascalemean" might not have converged']
%     disp(['N = ', num2str(length(p)), wstr])
% end

m = newvalue;
```

ConfidenceInterval.m

```
function relativeDelta = ConfidenceInterval(N, alpha)
```

```
% Tchebicheff:
% relativeDelta= sqrt(1-rstd(N).^2.*N./(N-1))./sqrt(alpha);
% relativeDelta=sqrt(((N-1)./N-rstd(N).^2))./rstd(N)./sqrt(alpha);

c2a = zeros(1,length(N));
for i = 1 : length(N)
    c2a(i) = chi2alpha(N(i)-1, alpha);
end

relativeHighLimit = (c2a ./ (N-1)); % para Sn-1 ^ 2
relativeDelta = sqrt(relativeHighLimit) - 1; % para Sn-1
```

chi2alpha.m

```
function c2a = chi2alpha(N, alpha)

% Calculates c2a so that: P(\chi^2 > c2a) = \alpha
% (see eq. 7.7)

% Following limits have been established by looking
% the tables to limit the search field (being sure
% that the solution remains inside) in order to increase
% the speed of the algorithm:
if (N < 10)
    X2_N = 0:25;
elseif (N < 27)
    X2_N = 2:50;
elseif (N < 41)
    X2_N = 11:70;
elseif (N < 71)
    X2_N = 20:105;
else
    X2_N = 43:141;
end


P = cdf('chi2', X2_N, N);

[v, a] = min(abs((1-P)-alpha));

c2a = X2_N(a);

X2_N = linspace(c2a-1, c2a+1, 201);

P = cdf('chi2', X2_N, N);
```

```
[v, a] = min(abs((1-P)-alpha));

c2a = X2_N(a);
```

Coefficient c_2: rstd.m

```
function c2 = rstd(n)
% Coefitient used by Fisher's theorem:
% \sigma^2 = c2(n)*(1/n)*sum_{i=1..n}((s_i)^2)

c2 = exp(gammaln(n./2) - gammaln((n-1)./2))./sqrt(n./2);
```

chi2alpha.m

mycdf.m

```
function F = mycdf(ppint, x, mu, sigma)

if nargin==2
    X=x;
elseif nargin == 4
    X = (x-mu)./sigma;
else
    error('unexpected number of parameters')
end

F = zeros (size (X));
F(:) = ppval(ppint, X(:));
```

gain.m

```
function A = gain( wT, N, delta )

% Calculates the gain effect of calculating the mean
% of a sample comming from a sine-wave (2nd figure on
% appendix B. Delta was set to zero)
L = length (N);
A = zeros( length(delta), L, length(wT) );

while delta > 2*pi
    delta = delta - 2 *pi;
end

for d = 1:length(delta)
for i = 1:L
    j = 0:N(i)-1;
    for k = 1:length(wT)
        A(d, i, k) = 1./N(i) .* sqrt( sum(cos(...
```

```
        delta(d)-wT(k).*j)).^2+ sum(sin(delta(d)-wT(k).*j)).^2);
    end
end
end
```

C.2 Codes for test simulations and results representation

This section contains the codes used for the simulation of the three different tests developed in this work (basic algorithm, direct self-adapting test and self-adapting test with use of the double queue algorithm.). Besides, functions written for an easy representation of the results are provided as well.

simulationSampleMean.m

```
function [NP, mu, xmed] = simulationSampleMean(data, K, N, muinit)

% The function receives the measurements DATA, the width of the
% control area K and the sample size N and the initial value of
% the process mean
%
% It returns vectors containing the evolution over time of: number
% of pieces "NP", asumed estimation of the process mean "MU" and
% the sample mean value "XMED".
%
% All vectors have the same size as vector "DATA"

i = 1; % Variable for the data vector. Plays the role of time
L = length(data);
NP = zeros(size(data))';
mu = zeros(size(data))';
xmed = zeros(size(data))';
queue = zeros(1, N);    % Queue of N data

NPi = 0;        % current number of pieces
mui = muinit;   % current sample mean

for j = 1 : N
    queue(j) = data(i);
    mu(i) = mui;
    xmed(i) = NaN;
    i = i+1;
end

HL = mui + K;   % high limit
LL = mui - K;   % low limit
```

```
while i <= L
% Read data.
    queue = insertnewvalue(queue, data(i));

    % Calculation of the Mean value.
    xmed(i) = mean(queue);

    if xmed(i) > HL
        NPi = NPi + 1;
        mui = mui + 147;
        HL = mui + K;
        LL = mui - K;
    elseif xmed(i) < LL
        NPi = NPi - 1;
        mui = mui - 147;
        HL = mui + K;
        LL = mui - K;
    end

    NP(i) = NPi;
    mu(i) = mui;
    i = i + 1;
end

%-----------------------------------
function x = insertnewvalue(x, v)
for i = 1:length(x)-1
    x(i) = x(i + 1);
end

 x(end) = v;
```

simgraphic.m

```
function simgraphic(data, xmean, mu, K, N, adjet, A)

% Function written for representing results from
% "simulationSampleMean.m"

plot(data,'.')
hold on
plot(xmean, 'g')
plot(mu, 'r')
plot(mu+K, 'k')
plot(mu-K,'k')

titulo = [adjet, ' test '];
```

```
if nargin == 7
    titulo = [titulo,'with \mu filter.',', A = ',num2str(A)];
end
titulo = [titulo, ', N = ', num2str(N), ', K = ',num2str(K)];

title(titulo);
legend('data', 'sample mean', 'inferred process mean',...
 'test limits', 0);
xlabel('Samples')
ylabel('Weight [g]')
v=axis;
axis([0, length(data), v(3:4)]);
```

simSM_AdaptiveTest.m

```
function [NP, MeanFilt_Out, Mean_Out, StdFilt_out, Std_Out,...
Nout, Kout] = simSM_AdaptiveTest(data, A, muinit, sigmainit)

% this function performs a simulation of the adaptive test.
%
% It RECIEVES:
% % a vector DATA with the measurements, a 2-elements vector A
% (typically [0.9998, 0.998]), the initial mean value of the weight
% MUINIT and the initial standard deviation SIGMAINIT (they are not
% calculated automatically).
%
% It RETURNS:
% a vector NP with the number of pieces in each moment (starting
% with zero),
% MEANFILT_OUT containing the filtered estimation of the real
% weight in the scale in each moment,
% MEAN_OUT containing the SAMPLE MEAN values in each moment,
% STDFILT_OUT containing the filtered estimation of the real
% standard deviation of the process,
% STD_OUT  containing the non-filtered estimations of the std.,
% NOUT With the size of the sample each moment,
% KOUT with values of K (size of the control area.)
%
% You can use SIMGRAPHICADAPTIVE.M to easily see the results

B = A(2);
A = A(1);
B_1 = 1-B;
A_1 = 1-A;
i = 1;      % Variable para el vector de datos
L = length(data);
NP = zeros(size(data))'; % Numero de piezas
```

```
MeanFilt_Out= zeros(size(data))';
StdFilt_out = MeanFilt_Out;
Mean_Out = zeros(size(data))';
Std_Out = Mean_Out;
Nout = Mean_Out;
Kout = Mean_Out;

N = 51;
N2 = 52;

if nargin < 4
   % Calc. of the first value of the std if SIGMAINIT is not
   % received as input.
   % Not yet implemented.
end
if nargin < 3
    % Calculation of the first value of the mean if MUINIT is not
    % received.
    % Not yet implemented.
end

[N, K] = optimalTestValues(sigmainit, N2); % ELECCION DE {N, K}.
queue = zeros(1, max(N2, N));    % Cola de N datos para la muestra

NPi = 0;        % Current number of pieces
MeanFilt = muinit;   % Current asumed mean value of the process
StdFilt = sigmainit; % Current asumed sigma of the process

HL = MeanFilt + K;   % high limit
LL = MeanFilt - K;   % low limit

for j = 1 : max(N2, N)   % Fill the queue out
    queue(j) = data(i);
    MeanFilt_Out(i) = MeanFilt;

    Mean_Out(i) = NaN;
    Std_Out(i) = NaN;
    i = i+1;
end

while i <= L
% Read data:
    queue = insertnewvalue(queue, data(i));

    % Calculation of the Sample Mean value.
    Mean_Out(i) = mean(queue(end-N+1:end));
```

```
    % Calculation of the Standard Deviation.
    Std_Out(i) = std(queue(end-N2+1:end));

    if Mean_Out(i) > HL
        NPi = NPi + 1;
        MeanFilt = MeanFilt + 147;
    elseif Mean_Out(i) < LL
        NPi = NPi - 1;
        MeanFilt = MeanFilt - 147;
    else
     % Filter for MeanFilt
        MeanFilt = A * MeanFilt + A_1 * Mean_Out(i);

        % Filter for sigmai
        StdFilt = B * StdFilt + B_1 * Std_Out(i);

        % Election of N and K {N, K}
        [N, K] = optimalTestValues(StdFilt, N2);
    end

    HL = MeanFilt + K;
    LL = MeanFilt - K;

    NP(i) = NPi;
    MeanFilt_Out(i) = MeanFilt;
    StdFilt_out(i) = StdFilt;
    Nout(i) = N;
    Kout(i) = K;
    i = i + 1;
end

%-------------------------------------------------------
function y = insertnewvalue(x, v)

   y = zeros(size(x));
   for i = 1:length(x)-1
       y(i) = x(i + 1);
   end

y(end) = v;
```

optimalTestValues.m

```
function [N, K] = optimalTestValues(sampleStd, N2)

% Diciembre: Con N2 = 52
```

```
limits = [100   113   127   140   154   169   188   205  221 ...
236   251   266   284];
Nopt = 6:18;

segments = length(limits);

% ConfidenceInterval(N2, .025); when N2 = 52 :
ConfInterval = 0.19328184794182;
DeltasampleStd = ConfInterval * sampleStd ./ rstd(N2);
sampleStdhigh = (sampleStd + DeltasampleStd);

N = zeros(size(sampleStd));
K = N;
for j=1:length(sampleStd)
    ind = 1;
    while ind < segments
        if  sampleStdhigh(j) > limits(ind)
            ind = ind + 1;
        else
            break
        end
    end
    if ind == segments
        warning('Sigma might be higher than the highest value...
        considered during the study!!')
    end
    N(j) = Nopt(ind);
end

K = max(3.3418 * sampleStd ./ sqrt(N), 73.5);   % alpha = 1e-2
% 73.5 is the minimal value of K for hysteresis not to happen
```

simSM_AdaptiveTest_StdOfMean.m

```
function [NP, MeanFilt_Out, Mean_Out, StdMFilt_out, StdM_Out,...
Nout, Kout] = simSM_AdaptiveTest_StdOfMean(data, A, muinit,...
sigmainit)

% This function performs the simulation of the self-adapting test
% by using the DOUBLE-QUEUE ALGORITHM. It is almost the same as
% simSM_AdaptiveTest.m
%
% this function RECIEVES:
% a vector DATA with the measurements, a 2-elements vector A
% (typically [0.9998, 0.998]), the initial mean value of the weight
% MUINIT and the initial standard deviation SIGMAINIT (they are not
% calculated automatically).
```

```
%
% It RETURNS:
% a vector NP with the number of pieces in each moment (starting
% with zero),
% MEANFILT_OUT containing the filtered estimation of the real
% weight in the scale in each moment,
% MEAN_OUT containing the SAMPLE MEAN values in each moment,
% STDMFILT_OUT containing the filtered estimation of the real
% standard deviation of the sample mean,
% STDM_OUT  containing the non-filtered estimations of the std.,
% NOUT With the size of the sample each moment,
% KOUT with values of K (size of the control area.)
%
% You can use SIMGRAPHICADAPTIVE.M to easily see the results

B = A(2);
A = A(1);
B_1 = 1-B;
A_1 = 1-A;
i = 1;      % Index for the data vector. (Plays the role of time)
L = length(data);
NP = zeros(size(data))'; % Number of units

% Vectores de salida de la simulacion,
MeanFilt_Out= zeros(size(data))';
StdMFilt_out = MeanFilt_Out;
Mean_Out = zeros(size(data))';
StdM_Out = Mean_Out;
Nout = Mean_Out;
Kout = Mean_Out;

N = 51;
N2 = 52;

if nargin < 4
   % Calc. of the first value of the std if SIGMAINIT is not
   % received as input.
   % Not yet implemented.
end
if nargin < 3
    % Calculation of the first value of the mean if MUINIT is not
    % received.
    % Not yet implemented.
end

% Selection of the first values of N and K:
[N, K] = optimalTestValues_StdOfMean(sigmainit/sqrt(N), N, N);
```

```
% queue with N data for the sample (its size will change, so in C
% it should take directly the highest value it can reach)
queue = zeros(1, N);    % cola de N datos para la muestra

% queue with N2 data for the sample mean values
MeanQueue = zeros(1, N2);

NPi = 0;        % Initial number of pieces on the scale.
MeanFilt = muinit;   % MeanFilt, media aceptada del proceso
StdMFilt = sigmainit/sqrt(N);   % desv aceptada del proceso

HL = MeanFilt + K;   % high limit
LL = MeanFilt - K;   % low limit

% STEP 1: Fill the queue out
for j = 1 : N   % Fill the queue out
    queue(j) = data(i);
    MeanQueue(j) = sum(queue(1:j)) / j;
    Mean_Out(i) = MeanQueue(j);
    MeanFilt = A * MeanFilt + A_1 * Mean_Out(i);
    MeanFilt_Out(i) = MeanFilt;

    StdM_Out(i) = NaN;
    i = i+1;
end

StdM_Out(i) = std(MeanQueue);

while i <= L
    queue = insertnewvalue(queue, data(i), N); % Read data.

    % Calculation of the Sample Mean value:
    Mean_Out(i) = mean(queue);
    MeanQueue = insertnewvalue(MeanQueue, Mean_Out(i), N2);
%     N = length(MeanQueue);
    Nout(i) = N;

    % Calculation of the Standard Deviation:
    StdM_Out(i) = std(MeanQueue);

% THE TEST ITSELFS: ----------------------------------------
    if Mean_Out(i) > HL
        NPi = NPi + 1;
        MeanFilt = MeanFilt + 147;
    elseif Mean_Out(i) < LL
        NPi = NPi - 1;
```

```
        MeanFilt = MeanFilt - 147;
    else
     % Filter for MeanFilt
        MeanFilt = A * MeanFilt + A_1 * Mean_Out(i);

        % Filter for sigmai
        StdMFilt = B*StdMFilt + B_1 * min(StdM_Out(i),300/sqrt(6));

        % ELECCION OF {N, K}
        [N, K] = optimalTestValues_StdOfMean(StdMFilt, N2, N);
    end

    HL = MeanFilt + K;
    LL = MeanFilt - K;

    NP(i) = NPi;
    MeanFilt_Out(i) = MeanFilt;
    StdMFilt_out(i) = StdMFilt;
%     Nout(1, i) = N;
    Kout(i) = K;
    i = i + 1;
end

%-----------------------------------------------------
function y = insertnewvalue(x, v, N)

if N == length(x)
    y = zeros(size(x));
    for i = 1:length(x)-1
        y(i) = x(i + 1);
    end
else
% N can not grow faster than 1 per sample!!
    y = zeros(1, min(N, length(x)+1));
    for i = 1:length(y)-1
        y(i) = x(i+1+(length(x)-length(y)));
    end
end

y(end) = v;
```

optimalTestValues_StdOfMean.m

```
function [N, K] = optimalTestValues_StdOfMean(sampleStdOfMean,...
N2,lastN)

limits = [100   113   127   140   154   169   188   205  221 ...
```

```
236   251   266   284];
Nopt = 6:18;
sampleStd = sampleStdOfMean * sqrt(lastN);
segments = length(limits);

% ConfidenceInterval(N2, .025); when N2 = 52 :
ConfInterval = 0.19328184794182;
DeltasampleStd = ConfInterval * sampleStd ./ rstd(N2);
sampleStdhigh = (sampleStd + DeltasampleStd);

% sampleStdlow = sampleStd - DeltasampleStd;
% sampleStdlow is not used here. It is already taken into account
%since it was used to calculate vector 'limits'

N = zeros(size(sampleStd));
K = N;
for j=1:length(sampleStd)
    ind = 1;
    while ind <= segments
        if  sampleStdhigh(j) > limits(ind)
            ind = ind + 1;
        else
            break
        end
    end

    if ind == segments
        warning('Sigma might be higher than the highest value...
        considered during the study!!')
    end
    N(j) = Nopt(ind);
end

K = max(3.3418 * sampleStd./sqrt(lastN), 73.5);   % alpha = 1e-2
% 73.5 is the minimal value of K for hysteresis not to happen
```

simgraphicAdaptive.m

```
function simgraphicAdaptive(data, xmean, mu, K, NP, A)

% Function written for representing results from
% "simSM_AdaptiveTest.m" or "simSM_AdaptiveTest_StdOfMean.m"

plot(data,'.')
hold on
plot(xmean, 'g')
plot(mu, 'r')
```