# User Manual:
# ECG Noise Extraction (ECG_NExT) Software Code

## Tool Reference

RST Reference Number: RST24IP01.01

Date of Publication: 08/08/2023

Recommended Citation: U.S. Food and Drug Administration. (2023). *ECG Noise Extraction Tool (ECGNExT)* (RST24IP01.01). https://cdrh-rst.fda.gov/ecg-noise-extraction-tool-ecgnext

## For more information

Catalog of Regulatory Science Tools to Help Assess New Medical Devices

# Disclaimer

## About the Catalog of Regulatory Science Tools

The enclosed tool is part of the Catalog of Regulatory Science Tools, which provides a peer-reviewed resource for stakeholders to use where standards and qualified Medical Device Development Tools (MDDTs) do not yet exist. These tools do not replace FDA-recognized standards or MDDTs. This catalog collates a variety of regulatory science tools that the FDA's Center for Devices and Radiological Health's (CDRH) Office of Science and Engineering Labs (OSEL) developed. These tools use the most innovative science to support medical device development and patient access to safe and effective medical devices. If you are considering using a tool from this catalog in your marketing submissions, note that these tools have not been qualified as Medical Device Development Tools and the FDA has not evaluated the suitability of these tools within any specific context of use. You may request feedback or meetings for medical device submissions as part of the Q-Submission Program.

For more information about the Catalog of Regulatory Science Tools, email OSEL_CDRH@fda.hhs.gov.

*Disclaimer*

The mention of commercial products, their sources, or their use in connection with material reported herein is not to be construed as either an actual or implied endorsement of such products by the Department of Health and Human Services.

# 1. General Information

This software code outputs an estimate of a noise signal collected as part of an electrocardiograph (ECG) recording. The objective is to obtain representative samples of noise and motion artifacts from ECG devices under test (DUT) that can then be used for testing the robustness of ECG analysis algorithms. The algorithm for noise estimation is described in [1]. The core operation of this software is based on removing the instances of the QRS complex from the ECG so that an estimate of the noise component of the recording remains. Required inputs are the ECG signal recorded with noise/motion artifacts and locations (indices) of the R-peaks.

# 2. System Requirements

The software code was written in a MATLAB® 2021b environment and consists of a core function and a shell script. The core function and its supporting functions are encapsulated in a class called “NoiseExtractClass.m” in the package. The shell script in the package is called “NoiseExtractShell.m” and is provided for demonstration purposes. While the core function can be used as a standalone code, the shell script can help with a quick start of a project. Software has been tested in MATLAB® 2021b environment without any additional toolboxes.

# 3. Data preparation:

The following steps are taken to prepare data as an input for processing and noise estimation by this software code:

1. Obtain an ECG signal using the device under test – *ECG_device* – at a location where noise/motion artifacts are captured in the ECG signal.
2. Obtain a simultaneously recorded clean ECG signal from a standard site of collection, e.g., the chest area for reference – *ECG_std*.
3. Apply a bandpass filter to the *ECG_device* signal to obtain *ECG_device_BP*. This is to clean the data from baseline wander and high-frequency noises as would normally be performed as part of data acquisition/processing for the device under test – typically in the range of 0.5 to 40 Hz. *ECG_std* can also be passed through a similar filtering scheme if needed for the purpose of better R peak detection.
4. Obtain the indices of the R peaks from, e. g., *ECG_std_BP* (or *ECG_device* if it provides sufficient fidelity for reliable peak detection).

A sample dataset as described in Section 6 is included in the software for demonstration purpose. The dataset includes variables *ECG_device*, *ECG_device_BP*, *ECG_std*, *fs* (sampling frequency), and *rPeaks* (R peak locations in samples).

# 4. Input and Output of ECG_NExT

Inputs:

1. *ECG_device:* 1-dimensional array of ECG samples recorded with noise at least 30 seconds length (the test data provides 60 seconds of data)
2. *ECG_device_BP*: 1-dimensional array Bandpass-filtered version of *ECG_device*,.
3. *rPeaks*: 1-dimensional array of R peak indices in samples (not in time units).
4. *fs*: sampling rate set to 1000 Hz. This sampling rate is hard coded in the software and will terminate with an error message if a different sampling rate is provided.
5. *crossFadePar* : This optional input is not shown in the sample shell code and can be used to specify cross fade parameters. This is a two-element array; the first element, the Blank parameter, specifies the length of data (in seconds) to be removed from both sides of each R peak. The second element, the Blend parameter, specifies the length of data (in seconds) to be copied from neighboring segments of the removed data, crossfaded and inserted in the gap created by the process of removal of the area around the R peaks. The default values for Blank and Blend parameters are 40 and 60 milliseconds, respectively. This optional input allows the user to adjust for the differences in R-Peak widths within a data set. For more details please refer to [1].

Outputs:

1. *noiseEstimated*: 1-dimensional array of estimated noise samples.
2. *tVec*: 1-dimensional array corresponding time vector for demonstration purposes, e. g., plotting original signal and the estimated noise versus time on a shared horizontal axis. The removed segments from the original signals are accounted for in this time vector.

# 5. Algorithm Development Test Conditions:

The test signals used during the software code development had the following characteristics:

1. Signals equivalent to *ECG_device* and *ECG_std* were 60-second-long signals sampled at 1000 Hz.

The signals representing *ECG_device_BP* and *ECG_std_BP* were obtained by applying a bandpass filter on the signals *ECG_device* and *ECG_std*, respectively. The *ECG_std_BP* was obtained for R peak detection purpose. The bandpass filter was a $5^{th}$ order Butterworth design with 0.5 and 40 Hz corner frequencies corresponding to low and high cutoff frequencies, respectively.

# 6. Software Package Content:

The software package includes the class containing the core and supporting functions for ECG noise estimation (NoiseExtractClass.m). In addition, a sample wrapper/shell code is provided to run the function with a test set, TestData.mat. This data set consists of a clean ECG, i.e., *ECG_std*, a noisy ECG representing an ECG from DUT, i.e., *ECG_device*, the band passed version of *ECG_device*, *ECG_device_BP*, the corresponding R peak locations, *rPeaks* and the value for sampling rate, *fs*. In the provided test set, ECG_std is a simulated ECG generated with the ECGSYN waveform generator available from the PhysioNet ECG repositories [2, 3]. In obtaining ECG_std using this model the following model parameters were used: ECG sampling frequency (sfecg = 1000 Hz), internal sampling frequency (sfint = 1000 Hz) and approximate number of heartbeats (N = 60). Remaining parameters used the default values. The first 60000 samples (60 seconds) of the simulated ECG were used for the test data *ECG_std*. *ECG_device* in this case is obtained by adding random Gaussian noise (zero mean, 500 mV std) to the *ECG_std*. The signals are 60-second long and sampled at 1000 Hz.

# References

[1]. Galeotti, L. and C.G. Scully, *A method to extract realistic artifacts from electrocardiogram recordings for robust algorithm testing.* Journal of Electrocardiology, 2018. **51**(6, Supplement): p. S56-S60.
[2]. Goldberger, A., Amaral, L., Glass, L., Hausdorff, J., Ivanov, P. C., Mark, R., ... & Stanley, H. E., *PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals.*, in *Circulation [Online]*. 2000. p. e215 - e220.
[3]. McSharry, P.E., G.D. Clifford, L. Tarassenko and L.A. Smith, *A dynamical model for generating synthetic electrocardiogram signals.* IEEE Transactions on Biomedical Engineering, 2003. p. 289-294.

# Appendix I

## Software Code

The shell code:

This is the primary module that runs the core software and initiates various function calls.

```
close all
clear
clc
% =+=+=+=+=+=+=+=+=+=+=+=+=+ User data +=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
% User data can be added here and should replace the test data below:
% =+=+=+=+=+=+=+=+=+=+=+ End of user data =+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

% =+=+=+=+=+=+=+=+=+=+=+=+=+ Test data +=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
testDat = load('TestData.mat');
ECG_device = testDat.ECG_device;
ECG_device_BP = testDat.ECG_device_BP;
rPeaks = testDat.rPeaks;
fs = testDat.fs;

% =+=+=+=+=+=+=+=+=+=+=+ End of test data =+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
NEWC = NoiseExtractClass;
[noiseEstimated, tVec] = NEWC.ecgNoiseExtractor(ECG_device, ECG_device_BP,
rPeaks, fs);

% Reduce noise record to same length as calculated noise record
NEWC.TestPlot(noiseEstimated, tVec, fs, ECG_device, testDat.ECG_std)
```

**The main code:**

```
classdef NoiseExtractClass
%{
This classdef construct includes the functions and sub modules necessary
to implement the ECG_NExT algorithm that obtains representative samples
of noise and motion artifacts from ECG devices under test.

Authors:
Ahmad Suliman
Christopher Scully
Loriano Galeotti

The ECG_NExT algorithm is described and published in:
Galeotti, L. and C.G. Scully, "A method to extract realistic artifacts from
electrocardiogram recordings from robust algorithm testing," Journal of
Electrocardiology, 2018. 51(6, Supplement): p. S56-S60.

If you have found this software useful, please consider citing our
publication.

Disclaimer:
This software and documentation (the "Software") were developed at the Food
and Drug Administration (FDA) by employees of the Federal Government in the
course of their official duties. Pursuant to Title 17, Section 105 of the
United States Code, this work is not subject to copyright protection and is
in the public domain. Permission is hereby granted, free of charge, to any
person obtaining a copy of the Software, to deal in the Software without
restriction, including without limitation the rights to use, copy, modify,
merge, publish, distribute, sublicense, or sell copies of the Software or
derivatives, and to permit persons to whom the Software is furnished to do
so. FDA assumes no responsibility whatsoever for use by other parties of
the Software, its source code, documentation or compiled executables, and
makes no guarantees, expressed or implied, about its quality, reliability,
or any other characteristic. Further, use of this code in no way implies
endorsement by the FDA or confers any advantage in regulatory decisions.
Although this software can be redistributed and/or modified freely, we ask
that any derivative works bear some notice that they are derived from it,
and any modified versions bear some notice that they have been modified.
%}
    properties
        Property1
    end

    methods (Static)
        function [NOISE_Calc, tVec] = ecgNoiseExtractor(ECG_test, ...
                ECGtest_BP, rPeaksRef, fs, varargin)
            %     [NOISE_Calc, tVec] = ecgNoiseExtractor(ECG_test, ...
            % ECGtest_BP, rPeaksRef, fs)
            %
            % This function extracts noise from ECG recording as described
            % in Galeotti and Scully, JECG 2018.
            %
```

```
% Inputs:
% ECG_test: The ECG signal obtained from the device under test.
% ECGtest_BP: The bandpass-filtered version of ECG_test.
% rPeaksRef: Accurately and reliably detected R peak indices of
%            the ECG obtained.
% fs: Sampling frequency at which the ECG signals are sampled.
%
% Outputs:
% NOISE_Calc: The calculated noise using this algorithm/code.
% tVec: Contains the time references corresponding to the noise
% instances remained after removing areas surrounding the R
% peaks. This is only needed for demo purposes when plotting the
% noise and ECG under consideration against time is desired.
%
% [...] = ecgNoiseExtractor(..., CROSSFADE_PAR) accepts a
% two-element array of Blank and Blend parameters ([Blank Blend])
% specifying the amount of data be removed from around the R
% peaks (Blank in seconds) and the amount of data to be copied
% from the neighboring sides of the removed part (Blend in
% seconds) to be cross faded and used to fill the gap
% created after removing the data from around the R peaks.
% The default values for the Blank and Blend parameters are set
% to 40 ms and 60 ms, respectively.

% Check for sampling frequency:
% Although the code should work with any sampling frequency, the
% sampling frequency used during test and development of this
% software was 1000 Hz. We, therefore, make certain that the
% users provide signals with 1000 Hz sampling frequency.
if fs ~= 1000
    error('Please change the sampling frequency to 1 kHz.')
end

sigLen = length(ECGtest_BP);

% Track time:
tTemp = (0:sigLen - 1)/fs;

% Ensure consistent dimensionality of 1xn -- row vectors:
ECG_test = ECG_test(:)';
ECGtest_BP = ECGtest_BP(:)';
rPeaksRef = rPeaksRef(:)';

% Get residuals from test ECG after filtering (noise components
% outside bandwidth of bandpass filtered signal)
ECGtest_residuals = ECG_test - ECGtest_BP;

% Obtain median beat from filtered ECG signal
[MedBeat] = ComputeMedianBeat_long(ECGtest_BP, rPeaksRef, fs);

% Generate synthetic signal using the median beat:
[SynECG] = GenSynECG_blend(MedBeat, rPeaksRef, sigLen, fs);

% Obtain the noise component of the signal:
```

```
    noiseRaw = ECGtest_BP - SynECG;

    % Remove areas around the R-peaks in the obtained noise as well
    % as in the residuals obtained above and the time vector:

    % Handle optional inputs:
    if isempty(varargin)
        NOISE_Calc = RemRpkEffect(noiseRaw, rPeaksRef, fs);
        residualLessRpeaks = RemRpkEffect(ECGtest_residuals, …
          rPeaksRef, fs);
        tVec = RemRpkEffect(tTemp, rPeaksRef, fs);
    else
        NOISE_Calc = RemRpkEffect(noiseRaw, rPeaksRef, fs, …
          varargin[2]);
        residualLessRpeaks = RemRpkEffect(ECGtest_residuals, …
          rPeaksRef, fs, varargin[2]);
        tVec = RemRpkEffect(tTemp, rPeaksRef, fs, varargin{1});
    end

    % Add noise to residual from bandpass filtered signal:
    NOISE_Calc = NOISE_Calc + residualLessRpeaks;

    % Remove NaN instances:
    tVec = tVec(~isnan(NOISE_Calc));
    NOISE_Calc = NOISE_Calc(~isnan(NOISE_Calc));

% End of ecgNoiseExtractor function
end

function TestPlot(noiseEstimated, tVec, fs, ECG_raw, ECG_ref)
    % This is a supplemental function for demo purposes in case the
    % user wants to plot the obtained noise with the ECG under
    % consideration.
    %
    % Inputs:
    % noiseEstimated: The estimated noise by the ECG_NExT algorithm.
    % tVec: the vector of time references corresponding to the
    %       estimated noise samples in noiseEstimated.
    % fs: sampling frequency
    % ECG_raw: The ECG collected from the device under test.
    % ECG_ref: The clean reference ECG (if collected). If not
    %       provided, the ECG_raw will be used.

    % Handle optional input(s):
    if nargin < 5
        ECG_ref = ECG_raw;
    end
    figure
    tECG = (0:length(ECG_raw)-1)/fs;
    ax(1) = subplot(3,1,1);
    plot(tECG, ECG_ref)
    title('Reference ECG')
    ax(2) = subplot(3,1,2);
    plot(tECG, ECG_raw)
```

```
            title('Unfiltered Device ECG')
            ax(3) = subplot(3,1,3);
            plot(tVec, noiseEstimated)
            title('Estimated Noise')
            linkaxes(ax, 'x')
            xlabel('Time (sec)')
            set(gcf, 'units', 'normal')
            set(gcf, 'position', [0.1753    0.0977    0.5064    0.7831])

        % End of function
        end

    % End of method
    end
% End of classdef
end

function [sigOut, stitch] = RemRpkEffect(sigIn, Rpeaks, fs, varargin)
    % This function removes an area around R peaks and merges the segments
    % surrounding the removed area with crossfade of neighboring signal
    % segments around the removed area.
    %
    % Inputs:
    % sigIn: This input can represent an ECG, the estimated noise or the time
    % vector in this context as a row of samples.
    % Rpeaks: R peak indices
    % fs: sampling frequency
    %
    % Outputs:
    % sigOut: blended sigIn
    % stitch: midpoint of crossfade region (samples of sigOut).

    % Definition of blanklen and blendlen parameters:
    % The blanklen parameter is the length of signal (in milliseconds) being
    % removed from around of each R peak in final ECG noise; this is the part
    % really canceled before and after each Rpeak.
    % The blendlen parameter is the length of the neighboring signal segment
    % (in milliseconds) around the canceled part to be crossfaded when
    % stitching.

    % An "if" construct is included to make blanklen and blendlen parameters
    % optional and assign them default values.
    if isempty(varargin)
        blanklen = 40e-3;
        blendlen = 60e-3;
    else
        blanklen = varargin{1}(1);
        blendlen = varargin{1}(2);
    end

    % Convert to samples:
    blanklen = blanklen * fs;
    blendlen = blendlen * fs;
```

```
    % Create linear crossfade function
    W = linspace(1, 0, blendlen + 1);

    out_locs = [Rpeaks(:)-blanklen Rpeaks(:)+blanklen];
    blend_locs = [out_locs(:,1)-blendlen out_locs(:,2)+blendlen];

    % Adjust for Rpeaks too close to end of signal
    if blend_locs(end,2) >= length(sigIn)
        blend_locs(end,:) = [];
        out_locs(end,:) = [];
        Rpeaks(end) = [];
    end

    % Initialize the stitch vector
    rpkLen = length(Rpeaks);
    stitch = zeros(1, rpkLen);

    % Start after 1st beat to remove errors related to location of 1st beat
    sigOut(1:blend_locs(1,2)) = NaN;
    for n = 2:rpkLen
        % Copy from end of last blend region to start of new blend region
        segadd = sigIn(blend_locs(n-1,2)+1:blend_locs(n,1)-1);
        sigOut(length(sigOut)+1:length(sigOut)+length(segadd)) = segadd;

        % Blend cross fade region
        segadd = sigIn(blend_locs(n,1):out_locs(n,1)).*W + ...
            sigIn(out_locs(n,2):blend_locs(n,2)).*(1-W);

        % Keep track of stitch points:
        stitch(n) = length(sigOut) + floor(length(segadd)/2);
        sigOut(length(sigOut)+1:length(sigOut)+length(segadd)) = segadd;
    end
    stitch(1) = [];

    % Add data after last Rpeak
    segadd = sigIn(blend_locs(n,2)+1:end);
    sigOut(length(sigOut)+1:length(sigOut)+length(segadd)) = segadd;

% End of function
end

function [MedBeat] = ComputeMedianBeat_long(ECG, Rpeaks, fs)
    % Compute median beat
    %
    % Inputs: ECG, R-peak locations
    % Output: Median beat, correction

    % Get short median, it is used to adjust X and Y offset
    MedShort = ComputeMedianBeat_short(ECG, Rpeaks);

    shortl = floor(length(MedShort)/3);

    % Arbitrary area for the QRS complex to do time sync. For normal patients
    % should be <120ms, in this case let's do 50 ms each side.
```

```
arbLen = 50e-3;
arbLenSamp = arbLen*fs;
qrsa = min(arbLenSamp, shortl-1) ;

% Arbitrary max lag for crosscorr search in seconds.
maxcorrlag = 20e-3;
maxcorrlagSamp = maxcorrlag*fs;

% Number of extra samples in each median beat (on the left double on the
% right) should be at least as much as the cross-fade half duration
extrabeat = 60;

% Since sampling frequency at development time was 1000 samples/s:
extrabeat = ceil(extrabeat*fs/1000);

% Determine max RR interval to set segment lengths
RRint = diff(Rpeaks);
maxRR = max(RRint);

% Get left and right segments, chop out a bit from 1/3>
segl = floor(maxRR*0.35) + extrabeat + 1;
segr = 2*segl;

% Initialize
ECGmatxy = NaN(length(Rpeaks),segl + segr + 1);
yoffset = NaN(length(Rpeaks),1);
xoffset = yoffset ;

m = 1;

% Catch cases where first Rpeak is too close to start of record
while (m < length(Rpeaks)) && (Rpeaks(m) - segl - maxcorrlagSamp < 1)
    m = m + 1;
end
% Main loop on each qrs peak to fill ECG beat array
while (m < length(Rpeaks)) && ...
        (Rpeaks(m) + segr + maxcorrlagSamp < length(ECG))

    % Calculate difference from short beat on Y
    shortdiff = MedShort - ECG(Rpeaks(m)-shortl:Rpeaks(m)+2*shortl);
    yoffset(m) = mean(shortdiff);
    [crosscorr, xcorrlag] = xcorr(ECG(Rpeaks(m)-qrsa:Rpeaks(m)+qrsa), ...
        MedShort(shortl-qrsa:shortl+qrsa), maxcorrlagSamp);
    [~, maxxcorri] = max(crosscorr);
    xoffset(m) = xcorrlag(maxxcorri);
    xoff = -xoffset(m);

    % Correct on X and Y.
    ECGmatxy(m,:) = (ECG(Rpeaks(m)-segl+xoff:Rpeaks(m)+2*segl+xoff))' + …
        yoffset(m);
    % Increase loop counter
    m = m + 1;

% End while
```

```
    end

    % Compute median from ECG beat array
    MedBeat = median(ECGmatxy, 1, 'omitnan');
% End of function
end

function [MedBeat, ECGmat] = ComputeMedianBeat_short(ECG, Rpeaks)
    % Compute median beat
    %
    % Inputs: ECG, R-peak locations
    % Output: Median beat

    % Get RR intervals
    RRint = diff(Rpeaks);
    minRR = min(RRint);

    % Get left and right segments, chop out a bit from 1/3
    segl = floor(minRR*.29);
    segr = 2*segl;

    % Remove peaks too close to the edge
    if Rpeaks(1) < segl
        Rpeaks = Rpeaks(2:end);
    end
    if length(ECG) < Rpeaks(end) + segr
        Rpeaks = Rpeaks(1:end-1);
    end

    % Initialize and fill ECG beat array
    ECGmat = zeros(length(Rpeaks), segl + segr + 1);
    for m = 1:length(Rpeaks)
        ECGmat(m,:) = (ECG(Rpeaks(m)-segl:Rpeaks(m)+segr))';
    end

    % Compute median from ECG beat array
    MedBeat = median(ECGmat,1, "omitnan");

    % Ensure it is a row
    MedBeat = MedBeat(:)';
end

function [SynECG] = GenSynECG_blend(MedBeat, Rpeaks, ecgRefLen, fs)
    % Compute synthetic ECG by blending median beats.
    %
    % Inputs: MedBeat, Rpeak location, ecgRefLen (Length of reference ECG)
    % Output: synthetic ECG

    % Cross-fade over last xx elements
    % Half of the crossfade
    crosshalfn = 50;

    % Since sampling rate at development time was 1000 sample/s:
    crosshalfn = floor(crosshalfn*fs/1000);
```

```
medl = floor(length(MedBeat)/3);

% Add a dummy beat at the end to fill the full space
Rpeaks(end+1) = Rpeaks(end)+floor(mean(diff(Rpeaks)));

% Set total out recording duration to avoid overflow
ECGlen = max(ecgRefLen, Rpeaks(end)+2*medl+2);

% Create linear crossfade function
W = linspace(1, 0, 2*crosshalfn + 1);

% Create empty output.
SynECG = NaN(1,ECGlen);
m = 2 ;

% If initial Rpeak is <= medl, delete and start with second Rpeak
if Rpeaks(1) < medl
    Rpeaks(1) = [];
end

while m < length(Rpeaks)
    % Get segment duration for current beat (a beat is 1 segment to the
    % left and 2 segments to the right of QRS)
    segl = floor((Rpeaks(m) - Rpeaks(m-1))*1/3);

    % Merge point is 1/3 beat back.
    mergepoint = Rpeaks(m) - segl;

    % Getting pointers for extremes of crossfade region of output
    sl = mergepoint - crosshalfn;
    sr = mergepoint + crosshalfn;

    % Pointers for xfade region of median beat
    ml = medl + 1 - segl - crosshalfn;
    mr = ml + 2*crosshalfn;
    mx = length(MedBeat) - mr;

    % Check for errors:
    if ml < 1
        disp (['ERROR! median beat too short to crossfade! add some' ...
            ' margin to median beats!!! (there is a setting in ' ...
            'compute median beat function!)'])
    end

    % Crossfade on the left
    SynECG(sl:sr) = SynECG(sl:sr).*W + MedBeat(ml:mr).*(1 - W);

    % Stamp median beat outside the crossfade region
    SynECG(sr+1: sr+mx) = MedBeat(mr+1:end);

    % Increment counter
    m = m + 1;
% End while
end
```

```
    SynECG = SynECG(1:ecgRefLen);
% End function
end
```