FINAL PROJECT REPORT

Implementation of ECG Dynamical Model to Generate Three Scenarios of Synthetic ECG Signals Using Fourth Order Runge-Kutta Algorithm

Name	: Jeremia Christ Immanuel Manalu
NRP	: 5023231017
Course	: Biomodelling (A)
Class	: A
Lecturer	: Nada Fitrieyatul Hikmah, S.T., M.T.
Department	: Biomedical Engineering

FACULTY OF INTELLIGENT ELECTRICAL AND INFORMATICS TECHNOLOGY

INSTITUT TEKNOLOGI SEPULUH NOPEMBER

2025

CHAPTER I. FUNDAMENTAL THEORY

1.1. ECG Signal

The synthetic ECG program is built as an output of a two-part:

1. A physiological RR-interval or tachogram generator that models autonomic oscillations (the respiratory sinus arrhythmia or RSA and the slower Mayer waves).
2. A dynamical oscillator in a 3-D state space whose limit cycle motion produces an ECG-like waveform, that are the P, Q, R, S, T complexes by modulating a third coordinate when the oscillator phase approaches angular positions associated with each wave.

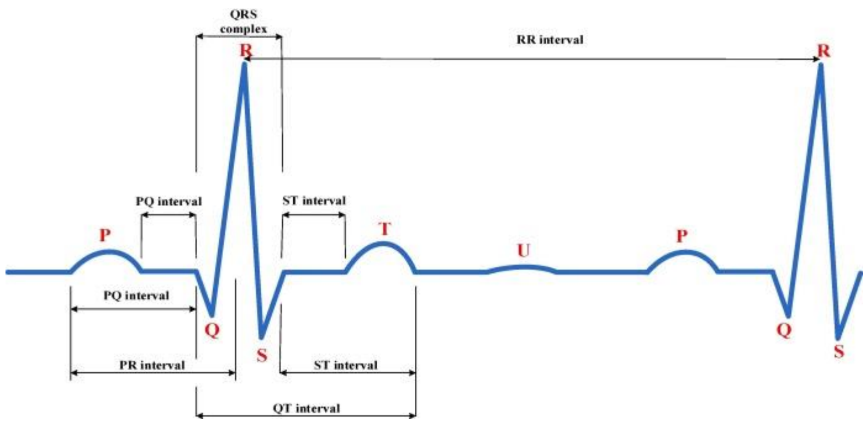


Figure 1.1. *An example of ECG Signal and Its Components*

RSA and Mayer waves are introduced in the RR-domain as spectral peaks at physiologically relevant frequencies (RSA in the high-frequency band related to breathing; Mayer waves in a lower frequency band). Those peaks are represented in the frequency domain and then converted to a time series (RR sequence). The RR sequence then drives the instantaneous angular velocity of the limit-cycle oscillator, which in turn determines the timing of morphological peaks.

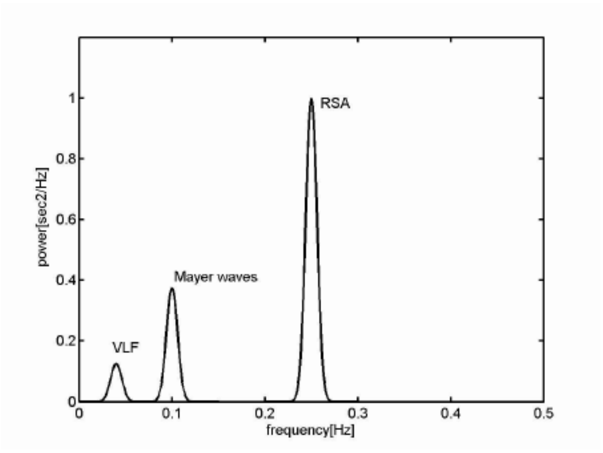


Figure 1.2. *The Mayer Waves and RSA Waves in Frequency Domain*

The mechanic description of the ECG signal morphology are as follow:

- The morphology is encoded by associating each morphological extrema (P, Q, R, S, two T components if used) with:
    - A phase angle on the oscillator's limit cycle (θ_i),

  - An amplitude coefficient (a_i),
  - A width (b_i) that controls how quickly that morphological contribution is localized around its phase,
  - And a Gaussian-like kernel centered at θ_i that produces a localized impulse in the third coordinate as the oscillator passes that phase.
- When the oscillator's phase $\theta(t)$ is near θ_i, the z-component receives a peaked forcing term shaped like $-a_i * \Delta\theta * \exp(-(\Delta\theta^2)/(2\ b_i^2))$ or an equivalent Gaussian-type shape, and the sign, the $\Delta\theta$ factor and the Gaussian control whether the resulting peak is upward (R/P/T) or downward (Q/S), and how sharp it is. The z-equation also contains a linear damping term that pulls z back to a baseline between impulses, producing distinct beats rather than a continuous drift.
- Respiratory modulation compresses or expands inter-beat intervals; the oscillator maps those inter-beat intervals into angular velocity, which faster HR means larger angular velocity. Morphology scales that are the peak widths/amplitudes and phase locations are adjusted based on mean heart rate so that wave durations and relative timing remain physiologically plausible at different rates.

1.2. Power spectrum and Frequency-Domain Design

The RR spectrum is constructed as a sum of Gaussian (normal) spectral components centered at chosen frequencies to represent RSA and Mayer waves with equation as follows:

$$S(f) \ = \ A_1 \cdot \mathcal{N}(f; f_1, \sigma_1^2) \ + \ A_2 \cdot \mathcal{N}(f; f_2, \sigma_2^2)$$

where: $\mathcal{N}(f; f_0, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(f-f_0)^2}{2\sigma^2}\right)$

To obtain the complex spectrum magnitude used for synthesis, the square root of the power spectral density is taken:

$$M(f) = \sqrt{S(f)}$$

To synthesize a real time signal from a one-sided magnitude spectrum, assign a random phase $\phi(f)$ uniformly in $[0,2\pi)$ to each (positive) frequency bin and form complex coefficients:

$$C(f_k) = M(f_k)\, e^{j\phi_k}$$

It is also must be noted to enforce conjugate symmetry for negative frequencies (mirror), so that the inverse discrete transform yields a real-valued time series:

$$C(-f_k) = C(\bar{f_k}) \Rightarrow \text{real time series.}$$

The discrete time RR-signal $r[n]$ is obtained via the inverse DFT of these complex coefficients:

$$r[n] \ = \ \frac{1}{N}\sum_{k=0}^{N-1} C(k)\, e^{j\frac{2\pi}{N}kn}, n = 0, \ldots, N-1$$

After the IDFT the resulting sequence is zero-mean (because only oscillatory components were synthesized); a mean RR (corresponding to the target mean heart rate) is added while also scaling the fluctuating component to match a target standard deviation.

1.3. Time domain (RR Tachogram, Baseline Wander, and Noise)

The time-domain RR series is the sequence of inter-beat intervals (seconds). After spectral design and inverse transform, the synthesized sequence is shifted by a desired mean RR (e.g., $60/\text{meanHR}$ in seconds) and scaled to match a target standard deviation (derived from target SD in beats per minute converted to seconds).

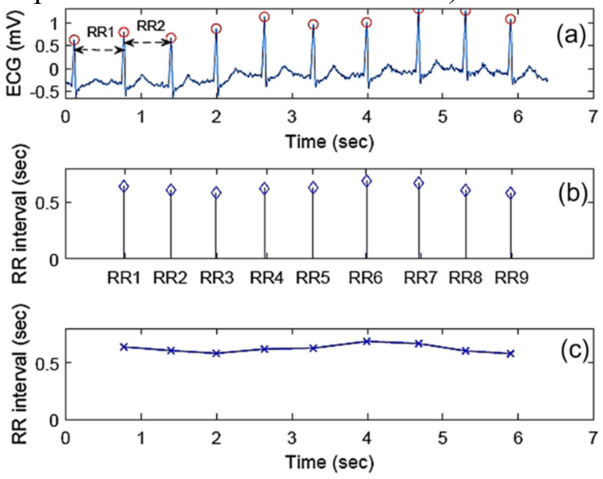


Figure 1.3. *Example of A RR Tachogram Graphs*

The oscillator uses the instantaneous RR to compute the instantaneous angular frequency (angular velocity):

$$\omega(t) = \frac{2\pi}{T(t)}$$

where $T(t)$is the current RR interval (in seconds). When $T(t)$varies beat to beat, ω(t) varies accordingly, compressing or expanding the oscillator's phase evolution and therefore the timing of P/Q/R/S/T events.

A slow baseline drift can be included (for example, a small sinusoid at a very low frequency) and higher-frequency additive noise can be added to the third coordinate to simulate measurement noise. The baseline term may appear in the z-equation as a slow sinusoidal offset toward which z relaxes. For practical notes, the synthesized RR time series should be of sufficient length for HRV metrics that require long windows (e.g., SDANN / 5-minute averages). Short recordings (e.g., <5 min) produce HRV estimates that must be interpreted with caution.

1.4. 3-D state space and modulation factor (α)

The core dynamical system has three variables (x, y, z). The (x,y) pair describe a self-sustained oscillator (limit cycle) and z is the morphological axis whose dynamics are driven by the oscillator's phase. A compact form of the system is in the equation below:

$$\begin{aligned}\dot{x} &= \alpha_r(x,y)\,x - \omega(t)\,y,\\ \dot{y} &= \alpha_r(x,y)\,y + \omega(t)\,x,\\ \dot{z} &= F_{\text{morph}}(\theta;\{\theta_i,a_i,b_i\}) - (z - z_{\text{base}}),\end{aligned}$$

with $\theta(t) = \arctan 2(y,x)$ and $\omega(t) = 2\pi/T(t)$.

The radial coefficient is chosen as a function of the oscillator radius $r = \sqrt{x^2 + y^2}$ often as $\alpha_r(x,y) = 1 - r$. In polar coordinates (r,θ), with $x = r\cos\,\theta, y = r\sin\,\theta$, this yields:

$$\dot{r} = \alpha_r \cdot r = r(1-r), \dot{\theta} = \omega(t).$$

The morphological term is a sum over wave contributions indexed by i:

$$F_{\text{morph}}(\theta) = \sum_i -a_i\ \Delta\theta_i\ \exp(-\frac{\Delta\theta_i^2}{2b_i^2}),$$

where $\Delta\theta_i$is the shortest angular difference between $\theta$and the i-th wave angle (handled with modular arithmetic to wrap across 2π). The negative sign and the Δθ factor tune the waveform shape; the exponential localizes the forcing near the target phase.

A modulation factor α is used to scale morphological parameters (angles, amplitudes, widths) according to mean heart rate. A commonly used simple mapping is:

$$\alpha = \sqrt{\frac{\text{mean HR}}{60\text{ bpm}}}$$

A linear damping term −(z − z_base) ensures that after the transient from the Gaussian-like forcing, z returns to baseline. The baseline z_base can itself be slowly time-varying (e.g., low-frequency wander)

1.5. Fourth-order Runge-Kutta (RK4) method

RK4 is used to integrate the ODE system forward in time with a fixed sampling interval (dt = 1 / sampling_frequency). RK4 balances accuracy and computational cost and is widely used for smooth ODEs like this oscillator. RK4 use the instantaneous RR to update ω(t) when the simulation crosses a beat boundary (like when integrated time reaches the next beat time). Small dt (high sampling rate) stabilizes the morphological details of the z output but costs computation. Typical ECG sampling frequencies (e.g., 250–1000 Hz) are used. Additive noise or baseline wander can be injected at each integration step into z if desired.

Standard RK4 step for a scalar ODE $\dot{u} = f(t,u)$ with step dt as follow:

$$\begin{aligned} k_1 &= f(t,u) \\ k_2 &= f\left(t + \frac{dt}{2}, \text{u}+\frac{dt}{2}k_1\right) \\ k_3 &= f\left(t + \frac{dt}{2}, \text{u}+\frac{dt}{2}k_2\right) \\ k_4 &= f(t + dt, u + dt\, k_3) \\ u(t+dt) &= u(t) + \frac{dt}{6}\,(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned}$$

Apply RK4 simultaneously to each of the three equations. Notationally for the Cartesian system:

$$k_{1x} = f_x(t,x,y,z), k_{1y} = f_y(t,x,y,z), k_{1z} = f_z(t,x,y,z)$$

$$k_{2x} = f_x\left(t + \frac{dt}{2}, x + \frac{k_{1x}dt}{2}, y + \frac{k_{1y}dt}{2}, z + \frac{k_{1z}dt}{2}\right)$$

etc. for $k_{2y}, k_{2z}, k_{3\cdot}, k_{4\cdot}$.

$$(x,y,z)_{t+dt} = (x,y,z)_t + \frac{dt}{6}\left((k_{1x},k_{1y},k_{1z}) + 2(k_{2x},k_{2y},k_{2z}) + 2(k_{3x},k_{3y},k_{3z}) + (k_{4x},k_{4y},k_{4z})\right)$$

1.6. HRV time-domain analysis

Let the RR sequence be $RR_1, \dots, RR_N$in seconds (or convert to ms as needed). Denote differences $\Delta_i = RR_{i+1} - RR_i$.

- **Mean RR**: $\bar{RR} = \frac{1}{N}\sum_{i=1}^{N} RR_i$. Convert to mean HR by mean HR (bpm) $= 60/\bar{RR}$.
- **SDNN (standard deviation of NN intervals)**:

$$\mathrm{SDNN} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(RR_i - \bar{RR})^2}.$$

- **RMSSD (root mean square of successive differences)**:

$$\mathrm{RMSSD} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N-1} \Delta_i^2}.$$

- **NN50 (count) and pNN50 (%)**:
  NN50 = number of $|\ \Delta_i\ | > 50$ ms.
  $p\mathrm{NN50} = 100 \times \frac{\mathrm{NN50}}{N-1}$.
- **SDANN (standard deviation of the average NN intervals for 5-minute segments)**: Partition the RR series into non-overlapping 5-minute windows (or other conventional windows), compute average RR within each window, then compute the standard deviation across those window averages. If the recording is shorter than the window length (e.g., <5 min), SDANN cannot be robustly estimated; it is common to mark results as “not enough data” or “for reference only” in that case.

1.7. ECG scenario modelling (Normal, Stress, Relax)

Model three physiological scenarios by setting (1) the target mean heart rate (HR) and (2) the target RR variability (standard deviation). Use the same synthesis pipeline from spectral design → IDFT → RR tachogram → phase-driven oscillator, but change the RR mean and the amplitude of the RR fluctuations.

- Converting mean HR (bpm) to mean RR (s):

$$\bar{RR} = \frac{60}{\bar{HR}}\text{(seconds)}.$$

- Represent the RR time series as a mean plus a normalized fluctuation:

$$RR(t) = \bar{RR} + A_{\mathrm{RR}} \cdot r_{\mathrm{norm}}(t),$$

where $r_{\mathrm{norm}}(t)$is a zero-mean unit-standard-deviation tachogram synthesized from the chosen power spectrum (RSA/Mayer components), and $A_{\mathrm{RR}}$is the amplitude that sets the RR standard deviation (i.e. $\mathrm{std}[RR(t)] = A_{\mathrm{RR}}$).

- Example mapping of scenario to parameters:

$$\begin{cases} Normal: HR \approx 60\ bpm, ARR \approx 0.04 - 0.06\ s \\ Stress\ (low\ HRV): HR \approx 80\ bpm, ARR \approx 0.01 - 0.03\ s \\ Relaxed\ (high\ HRV): HR \approx 55\ bpm, ARR \approx 0.06 - 0.10\ s \end{cases}$$

These numeric ranges are **examples** to make the differences clear, must be choose precise values based on validation targets or literature.

- Phase / morphology scaling with HR (same α used elsewhere):

$$\alpha = \sqrt{\frac{H\bar{R}}{60\ \text{bpm}}}$$

Use α to scale morphological widths/durations so P–Q–R–S–T timing compresses at higher HR and expands at lower HR.

CHAPTER II. RESULT AND ANALYSIS

2.1. Problems Statement

Implement or modify the given ECG dynamical model to generate three scenarios of synthetic ECG signals:

- Normal HRV ➔ mean HR approximately 60 bpm, normal stdv.
- Stress (Low HRV) ➔ mean HR appr 80 bpm, lower stdv.
- Relaxed (High HRV) ➔ mean HR appr 55 bpm, higher sdtv.

2.2. Code Explanation

The following program is organized in three logical stages:

1. HRV/RR synthesis that will create a tachogram (sequence of RR intervals) from a user-specified frequency-domain design (two spectral peaks representing RSA and Mayer-like components), then normalize and scale it to a target mean and standard deviation.
2. RR post-processing/scenario selection will allow scenario selection (Normal, Stress, Relaxed) that sets the target mean HR and HRV (std dev) and optionally apply a step/block function or add baseline wander and measurement noise.
3. ECG morphology generation that drive a 3-D dynamical oscillator (x,y,z) using the instantaneous RR to make the phase evolve; produce ECG as the z-projection. Integration is done with a fourth-order Runge–Kutta (RK4) solver.

a. *procedure TForm1.FormCreate(Sender: TObject)*

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  GridECGParams.ColCount := 4;
  GridECGParams.RowCount := 7;
  GridECGParams.Cells[1, 0] := 'Teta (degrees)';
  GridECGParams.Cells[2, 0] := 'Amplitude (a)';
  GridECGParams.Cells[3, 0] := 'Width (b)';
  GridECGParams.Cells[0, 1] := 'P';
  GridECGParams.Cells[0, 2] := 'Q';
  GridECGParams.Cells[0, 3] := 'R';
  GridECGParams.Cells[0, 4] := 'S';
  GridECGParams.Cells[0, 5] := 'T-';
  GridECGParams.Cells[0, 6] := 'T+';
  GridECGParams.Cells[1, 1] := FloatToStr(RadToDeg(-Pi / 3));
  GridECGParams.Cells[2, 1] := '0.8';
  GridECGParams.Cells[3, 1] := '0.2';
  GridECGParams.Cells[1, 2] := FloatToStr(RadToDeg(-Pi / 12));
  GridECGParams.Cells[2, 2] := '-5.0';
  GridECGParams.Cells[3, 2] := '0.1';
  GridECGParams.Cells[1, 3] := '0';
  GridECGParams.Cells[2, 3] := '30.0';
  GridECGParams.Cells[3, 3] := '0.1';
  GridECGParams.Cells[1, 4] := FloatToStr(RadToDeg(Pi / 12));
  GridECGParams.Cells[2, 4] := '-7.5';
  GridECGParams.Cells[3, 4] := '0.1';
  GridECGParams.Cells[1, 5] := FloatToStr(RadToDeg((5 * Pi / 9) - (Pi / 60)));
  GridECGParams.Cells[2, 5] := '0.5';
  GridECGParams.Cells[3, 5] := '0.4';
  GridECGParams.Cells[1, 6] := FloatToStr(RadToDeg(5 * Pi / 9));
  GridECGParams.Cells[2, 6] := '0.75';
  GridECGParams.Cells[3, 6] := '0.2';
end;
```

The event handler (form initialization) that have the purpose to initialize the ECG parameter grid (morphology table) and default values for the P/Q/R/S/T waves. It will sets *GridECGParams.ColCount*, *RowCount*, and fills grid cells with initial angles (in degrees), amplitudes and widths. These are the *reference* morphological parameters used later and scaled by α in *btnGenerateECGClick*.

b. *procedure TForm1.BitBtn1Click(Sender: TObject)*

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  Close;
end;
```

This procedure implements the event handler close button that will closes the form/application on click. This procedure will just calls *Close;*. No variables or math.

c. *procedure TForm1.btnClearClick(Sender: TObject)*

```
procedure TForm1.btnClearClick(Sender: TObject);
begin
  Series1.Clear;
  Series2.Clear;
  Series3.Clear;
  Series4.Clear;
  Series5.Clear;
  Series6.Clear;
  Series7.Clear;
  ListBoxStep7.Clear;
  ListBoxStep8.Clear;
  ListBoxStep9.Clear;
  edtSDNN.Text := '';
  edtSDANN.Text := '';
  edtRMSSD.Text := '';
  edtNN50.Text := '';
  edtpNN50.Text := '';
  SetLength(sf, 0);
  SetLength(sw, 0);
  SetLength(reall, 0);
  SetLength(imag, 0);
  SetLength(idft_raw, 0);
  SetLength(rr_continuous, 0);
  SetLength(rr_final, 0);
  PageControl.ActivePageIndex := 0;
end;
```

This procedure is also an event handler (reset) that will clear all chart series, listboxes, HRV outputs, and free the dynamic arrays used in synthesis (so next generation starts fresh). It will clear plotted series (*SeriesX.Clear*) and UI listboxes/text, *SetLength(..., 0)* for all arrays holding spectral data and RR arrays and reset page control to the first page.

d. *procedure TForm1.btnGenerateHRVClick(Sender: TObject)*

```
procedure TForm1.btnGenerateHRVClick(Sender: TObject);
var
  i, k, n: Integer;
  f1, f2, c1, c2, ratio, hstd, magSf, f, rr_mean, rr_scaling: Double;
  tempReal, tempImag, angle: Double;
  sf1, sf2, random_phase: Double;
  target_std_dev_sec, raw_mean, raw_sum_sq_diff, raw_std_dev,
scale_factor: Double;
begin
  try
    f1    := StrToFloat(edtF1.Text);
    f2    := StrToFloat(edtF2.Text);
    c1    := StrToFloat(edtC1.Text);
    c2    := StrToFloat(edtC2.Text);
    ratio := StrToFloat(edtRatio.Text);
    Nrr   := StrToInt(edtNumPoints.Text);
    hmean := StrToFloat(edtMeanHR.Text);
    hstd  := StrToFloat(edtStdDevHR.Text);
  except
    on E: Exception do begin
      ShowMessage('Error: Invalid parameters in Step 1. ' + E.Message);
      Exit;
    end;
  end;
  btnClearClick(nil);
  SetLength(sf, Nrr);
  SetLength(sw, Nrr);
  SetLength(reall, Nrr);
  SetLength(imag, Nrr);
  SetLength(idft_raw, Nrr);
  SetLength(rr_continuous, Nrr);

  magSf := 1.7 / Nrr;
  for i := 0 to Nrr - 1 do
  begin
    f := i * 1.0 / Nrr;
    sf1 := ratio * magSf * Exp(-Sqr(f - f1) / (2 * Sqr(c1))) / Sqrt(2 * Pi * Sqr(c1));
    sf2 := magSf * Exp(-Sqr(f - f2) / (2 * Sqr(c2))) / Sqrt(2 * Pi * Sqr(c2));
    sf[i] := sf1 + sf2;
    Series1.AddXY(f, sf[i]);
  end;
  for i := 0 to Round(Nrr / 2) - 1 do sw[i] := Sqrt(sf[i]);
  for i := Round(Nrr / 2) to Nrr - 1 do sw[i] := Sqrt(sf[Nrr - i]);
  for i := 0 to Nrr - 1 do Series2.AddXY(i * 1.0 / Nrr, sw[i]);
  for i := 0 to Nrr - 1 do
  begin
    random_phase := Random * 2 * Pi;
    reall[i] := sw[i] * Cos(random_phase);
    imag[i]  := sw[i] * Sin(random_phase);
    Series3.AddXY(i * 1.0 / Nrr, reall[i]);
    Series4.AddXY(i * 1.0 / Nrr, imag[i]);
  end;
  for n := 0 to Nrr - 1 do
  begin
    tempReal := 0; tempImag := 0;
    for k := 0 to Nrr - 1 do
    begin
      angle := 2 * Pi * k * n / Nrr;
      tempReal := tempReal + (reall[k] * Cos(angle) - imag[k] * Sin(angle));
      tempImag := tempImag + (reall[k] * Sin(angle) + imag[k] * Cos(angle));
    end;
    idft_raw[n] := tempReal / Nrr;
    Series5.AddXY(n, idft_raw[n]);
  end;

  raw_mean := 0;
  for i := 0 to Nrr - 1 do raw_mean := raw_mean + idft_raw[i];
  raw_mean := raw_mean / Nrr;
  raw_sum_sq_diff := 0;
  for i := 0 to Nrr - 1 do raw_sum_sq_diff := raw_sum_sq_diff + Sqr(idft_raw[i] - raw_mean);
  if (Nrr > 1) then raw_std_dev := Sqrt(raw_sum_sq_diff / (Nrr - 1)) else raw_std_dev := 0;
  target_std_dev_sec := hstd / 60.0;
  if raw_std_dev > 1E-9 then scale_factor := target_std_dev_sec / raw_std_dev else scale_factor := 0;
  rr_mean := 60.0 / hmean;
  edtRRMean.Text := FloatToStrF(rr_mean, ffFixed, 8, 4);
  edtRRScaling.Text := FloatToStrF(scale_factor, ffFixed, 8, 4);

```

```
for i := 0 to Nrr - 1 do
begin
  rr_continuous[i] := ((idft_raw[i] - raw_mean) * scale_factor) + rr_mean;
end;

UpdateRRFinalPlot;

PageControl.ActivePageIndex := 5;
 end;
```

This procedure is the core of the HRV synthetis algorithm. It will build the RR tachogram from a frequency-domain design (two Gaussian peaks), convert magnitude to complex spectrum with random phases, perform inverse DFT, normalize & scale to target mean RR and standard deviation, and fill *rr_continuous*. Then call *UpdateRRFinalPlot*. It have some steps as follows:

1. Parameter read/validation that read *f1, f2, c1, c2, ratio, Nrr, hmean, hstd* from UI. (Error-handling with try/except.
2. Allocate arrays, *SetLength(sf, Nrr); SetLength(sw, Nrr);* etc.
3. Build power spectrum *sf[i]*, for each bin *i=0..Nrr-1* compute normalized frequency *f := i / Nrr*. Then compute the sum of two Gaussian spectral components where *magSf := 1.7 / Nrr*. The equation as follows:

$$S(f) = A_1\,\mathcal{N}(f; f_1, c_1^2) + A_2\,\mathcal{N}(f; f_2, c_2^2)$$

with normalization constant *1/sqrt(2π c^2)* and scaled by *magSf* and *ratio*.

4. Compute magnitude spectrum *sw* as sqrt of *sf*. The code does a half-spectrum mirroring, for indices *0..Nrr/2-1 sw[i] := sqrt(sf[i]); for Nrr/2 .. Nrr-1 sw[i] := sqrt(sf[Nrr-i])*. This creates a symmetric magnitude array appropriate for a real IDFT. Equation used: $M[k] = \sqrt{S[k]}$
5. Assign random phases and real/imag parts. For each index *i*, *draw random_phase := Random * 2 * Pi*, then compute *reall[i] := sw[i] * Cos(random_phase);* and *imag[i] := sw[i] * Sin(random_phase)*; So complex coefficient $\mathrm{C[k]} = \mathrm{M[k]e^{j\phi_k}}$ represented by (*reall,imag*).
6. The code explicitly computes IDFT by summation, with Equation: $r_{\text{raw}}[n] = \frac{1}{N}\sum_{k=0}^{N-1}\Re\{C[k]e^{j2\pi kn/N}\}$. The code computes only the real part, which is correct given conjugate symmetry. It is important to be noted that the IDFT is implemented by nested loops ($O(N^2)$) rather than FFT. It will fine for small *Nrr* but slow for large *N*.
7. Normalize and scale. First, compute *raw_mean* and *raw_std_dev* of *idft_raw*. Then, *target_std_dev_sec := hstd / 60.0* converts HR standard deviation (bpm) to seconds. The *scale_factor := target_std_dev_sec / raw_std_dev* will guard against tiny *raw_std_dev*. Next, *rr_mean := 60.0 / hmean* will convert mean HR to mean RR (sec). Equations used: $\bar{RR} = 60/\bar{HR}$, and $RR[n] = \bar{RR} + \frac{\sigma_{\text{target}}}{\sigma_{\text{raw}}}\,(r_{\text{raw}}[n] - \mu_{\text{raw}})$.
8. *Call UpdateRRFinalPlot* to make *rr_final* and plot it.

e. *procedure TForm1.UpdateRRFinalPlot*

```
procedure TForm1.UpdateRRFinalPlot;
var
  i, intervalDuration: Integer;
  // Variables to calculate the baseline
  sum_rr, mean_rr: Double;
begin
  if Length(rr_continuous) = 0 then Exit;

  SetLength(rr_final, Length(rr_continuous));

  // Read and validate the interval duration from the UI
  try
    intervalDuration := StrToInt(edtIntervalDuration.Text);
    if intervalDuration <= 0 then
      intervalDuration := High(Integer);
  except
    intervalDuration := High(Integer);
  end;

  // Pre-calculate the signal mean to use as baseline
  sum_rr := 0;
  for i := 0 to Length(rr_continuous) - 1 do
  begin
    sum_rr := sum_rr + rr_continuous[i];
  end;
  if Length(rr_continuous) > 0 then
    mean_rr := sum_rr / Length(rr_continuous)
  else
    mean_rr := 0;

  // Plotting Logic
  Series6.Clear;
  Series6.BeginUpdate;
  try
    for i := 0 to Length(rr_continuous) - 1 do
    begin
      var current_rr := rr_continuous[i];

      // If the checkbox is checked, apply the alternating block logic
      if cbApplyStepFunction.Checked then
      begin
        // Check if the current index falls into a "low" block (odd-numbered blocks)
        if (i div intervalDuration) mod 2 = 1 then
        begin
        // Set the value to the calculated mean (baseline)
        current_rr := mean_rr;
      end;
      // If it's an even block, 'current_rr' retains its original value.
    end;

    rr_final[i] := current_rr;
    Series6.AddXY(i, rr_final[i]);
  end;
finally
  Series6.EndUpdate;
end;
end;
```

This procedure is a private procedure that will build *rr_final* from *rr_continuous* and optional step/block transformation, plot *Series6* of the final RR values. First, it will validate *len(rr_continuous) > 0* and set *SetLength(rr_final, Length(rr_continuous))*. Second, it will read *intervalDuration := StrToInt(edtIntervalDuration.Text)* with fallback to *High(Integer)* if invalid or ≤ 0. Next, will compute the mean of *rr_continuous* (*mean_rr*) by summation. Then it will loop through indices *i = 0 .. Length(rr_continuous)-1*:

- *current_rr := rr_continuous[i].*
- If *cbApplyStepFunction.Checked* is true and *(i div intervalDuration) mod 2 = 1* then set *current_rr := mean_rr* (even blocks keep original RR, odd blocks set to baseline mean).
- *rr_final[i] := current_rr* and plot *Series6.AddXY(i, rr_final[i]).*

f. *procedure TForm1.cbApplyStepFunctionClick(Sender: TObject)*

```
procedure TForm1.cbApplyStepFunctionClick(Sender: TObject);
begin
  if Length(rr_continuous) > 0 then
  begin
    UpdateRRFinalPlot;
  end;
 end;
```

This procedure is a event handler (UI checkbox) that when user toggles the "apply step function" checkbox, call *UpdateRRFinalPlot* (if *rr_continuous* exists) to refresh *rr_final* and the plot.

g. *procedure TForm1.btnApplyScenarioClick(Sender: TObject)*

```
procedure TForm1.btnApplyScenarioClick(Sender: TObject);
begin
  case pcScenarios.ActivePageIndex of
    0: begin // Normal HRV
      edtMeanHR.Text := edtNormalHR.Text;
      edtStdDevHR.Text := edtNormalSTD.Text;
    end;
    1: begin // Stress (Low HRV)
      edtMeanHR.Text := edtStressHR.Text;
      edtStdDevHR.Text := edtStressSTD.Text;
    end;
    2: begin // Relaxed (High HRV)
      edtMeanHR.Text := edtRelaxedHR.Text;
      edtStdDevHR.Text := edtRelaxedSTD.Text;
    end;
  end;
  btnGenerateHRVClick(nil);
  if Length(rr_final) > 0 then
  begin
    btnGenerateECGClick(nil);
    CalculateAndDisplayHRV;
  end;
end;
```

The event handler that will apply scenario preset. This procedure will copy a scenario preset (Normal/Stress/Relaxed) from the scenario tab controls into the main mean/std fields, then regenerate HRV (*btnGenerateHRVClick*) and ECG (*btnGenerateECGClick*) and compute HRV metrics.

h. *procedure TForm1.btnGenerateECGClick(Sender: TObject)*

```
procedure TForm1.btnGenerateECGClick(Sender: TObject);
var
  i: TWaveIndex;
  alpha: Double;
  base_theta, base_a, base_b: array[TWaveIndex] of Double;
const
  waveNames: array[TWaveIndex] of string = ('P', 'Q', 'R', 'S', 'T-', 'T+');
begin
  if Length(rr_final) = 0 then
  begin
    ShowMessage('Please generate Heart Rate Variability (Step 1-5) first.');
    Exit;
  end;
  Series7.Clear;
  ListBoxStep7.Clear;
  ListBoxStep8.Clear;
  ListBoxStep9.Clear;

  try
    fecg := StrToFloat(edtSamplingFreq.Text);
    if fecg <= 0 then raise Exception.Create('Sampling Frequency must be positive.');

    for i := Low(TWaveIndex) to High(TWaveIndex) do
    begin
      base_theta[i] := StrToFloat(GridECGParams.Cells[1, i + 1]);
      base_a[i]     := StrToFloat(GridECGParams.Cells[2, i + 1]);
      base_b[i]     := StrToFloat(GridECGParams.Cells[3, i + 1]);
    end;
  except
    on E: Exception do begin
      ShowMessage('Error: Invalid ECG parameters in Step 6. ' + E.Message);
      Exit;
    end;
  end;

  ListBoxStep7.Items.Add('### Modulation Factor α (alpha) Calculation ###');
  alpha := Sqrt(hmean / 60.0);
  ListBoxStep7.Items.Add(Format('Mean HR (hmean) = %.2f bpm', [hmean]));
  ListBoxStep7.Items.Add(Format('Modulation Factor α = sqrt(%.2f / 60) = %.4f', [hmean, alpha]));
  ListBoxStep7.Items.Add('');
  ListBoxStep7.Items.Add('\\\ Morphological Parameter Adjustment (using Grid values) \\\');
  ListBoxStep7.Items.Add(Format('%-5s | %-15s | %-15s | %-15s', ['Wave', 'Final Teta (rad)', 'Final Amplitude (a)', 'Final Width (b)']));
  ListBoxStep7.Items.Add(StringOfChar('-', 60));

  theta_i[0] := DegToRad(base_theta[0]) * Sqrt(alpha);
  a_i[0]     := base_a[0];
  b_i[0]     := base_b[0] * alpha;
  theta_i[1] := DegToRad(base_theta[1]) * alpha;
  a_i[1]     := base_a[1];
  b_i[1]     := base_b[1] * alpha;
  theta_i[2] := 0;
  a_i[2]     := base_a[2];
  b_i[2]     := base_b[2] * alpha;
  theta_i[3] := DegToRad(base_theta[3]) * alpha;
  a_i[3]     := base_a[3];
  b_i[3]     := base_b[3] * alpha;
  theta_i[4] := (DegToRad(base_theta[5]) * Sqrt(alpha)) - (DegToRad(base_theta[0])/20 * Sqrt(alpha));
  a_i[4]     := base_a[4] * Power(alpha, 2.5);
  b_i[4]     := base_b[4] / alpha;
  theta_i[5] := DegToRad(base_theta[5]) * Sqrt(alpha);
  a_i[5]     := base_a[5] * Power(alpha, 2.5);
  b_i[5]     := base_b[5] * alpha;

  for i := Low(TWaveIndex) to High(TWaveIndex) do
  begin
```

```
    ListBoxStep7.Items.Add(Format('%-5s | %-15.4f | %-15.4f | %-15.4f',
      [waveNames[i], theta_i[i], a_i[i], b_i[i]]));
  end;

  ListBoxStep8.Items.Add('### Angular Velocity Concept (ω(t)) ###');
  ListBoxStep8.Items.Add('The model''s circular motion speed is determined by the RR interval.');
  ListBoxStep8.Items.Add('Shorter RR intervals (faster heart rate) result in higher ω(t).');
  ListBoxStep8.Items.Add('Formula: ω(t) = 2 * π / T(t), where T(t) is the RR interval at time t.');
  ListBoxStep8.Items.Add('The logic now correctly updates this value for each beat.');

  ListBoxStep9.Items.Add('### Ordinary Differential Equation (ODE) System ###');
  ListBoxStep9.Items.Add('This model uses 3 equations to describe the trajectory (x,y,z) in 3D space.');
  ListBoxStep9.Items.Add('The ECG output is the projection of this trajectory onto the z-axis.');
  ListBoxStep9.Items.Add('Equations: ẋ = α*x - ω*y, ẏ = α*y + ω*x, ż = ...');
  ListBoxStep9.Items.Add('The ż equation creates P,Q,R,S,T-,T+ peaks as the angle (θ) nears each wave''s angle (θi).');

  RungeKutta4_ECG;
  PageControl.ActivePageIndex := 9;
end;
```

This procedure is a event handler that will prepare ECG morphology and call integrator. It will read sampling frequency and base morphology grid, compute modulation factor alpha from *hmean*, compute final morphology parameters *theta_i*, *a_i*, *b_i* (scaling by α in different ways per wave), show textual summary (ListBoxes), then call *RungeKutta4_ECG*. The modulation factor is *alpha := Sqrt(hmean / 60.0);* and the procedure will reads base parameters from *GridECGParams*. Then, final parameters are computed using a mixture of alpha, *Sqrt(alpha)*, *Power(alpha, 2.5)*, and divisions by *alpha*. So different waves use different scaling laws (some multiply width by *alpha*, some divide, some change *theta* by *alpha* or *sqrt(alpha)*, and amplitudes of T waves scaled by *alpha^2.5*).

i. procedure TForm1.RungeKutta4_ECG

```
procedure TForm1.RungeKutta4_ECG;
var
  i: Integer;
  t, dt: Double;
  x, y, z: Double;
  k1x, k1y, k1z, k2x, k2y, k2z, k3x, k3y, k3z, k4x, k4y, k4z: Double;
  noise, noise_amp: Double;
  beat_index: Integer;
  time_of_next_beat: Double;
begin
  dt := 1.0 / fecg;
  x := 1.0; y := 0.0; z := 0.04;
  t := 0.0;
  beat_index := 0;
  if Length(rr_final) > 0 then
  begin
    time_of_next_beat := rr_final[0];
    current_rr_interval := rr_final[0];
  end else begin
    time_of_next_beat := 60.0 / hmean;
    current_rr_interval := 60.0 / hmean;
  end;

  if cbAddNoise.Checked then try
      noise_amp := StrToFloat(edtNoiseAmp.Text);
    except
      noise_amp := 0.0;
    end
  else
    noise_amp := 0.0;

  Series7.BeginUpdate;
  try
    for i := 0 to Round(Nrr) do
    begin
      if t >= time_of_next_beat then
      begin
        Inc(beat_index);
        if beat_index < Length(rr_final) then
        begin
          time_of_next_beat := time_of_next_beat + rr_final[beat_index];
          current_rr_interval := rr_final[beat_index];
        end
        else
        begin
          if Length(rr_final) > 0 then
            current_rr_interval := rr_final[Length(rr_final) - 1];
        end;
      end;

      k1x := ddt(t, x, y, z, 1);
      k1y := ddt(t, x, y, z, 2);
      k1z := ddt(t, x, y, z, 3);
      k2x := ddt(t + 0.5*dt, x + 0.5*k1x*dt, y + 0.5*k1y*dt, z + 0.5*k1z*dt, 1);
      k2y := ddt(t + 0.5*dt, x + 0.5*k1x*dt, y + 0.5*k1y*dt, z + 0.5*k1z*dt, 2);
      k2z := ddt(t + 0.5*dt, x + 0.5*k1x*dt, y + 0.5*k1y*dt, z + 0.5*k1z*dt, 3);
      k3x := ddt(t + 0.5*dt, x + 0.5*k2x*dt, y + 0.5*k2y*dt, z + 0.5*k2z*dt, 1);
      k3y := ddt(t + 0.5*dt, x + 0.5*k2x*dt, y + 0.5*k2y*dt, z + 0.5*k2z*dt, 2);
      k3z := ddt(t + 0.5*dt, x + 0.5*k2x*dt, y + 0.5*k2y*dt, z + 0.5*k2z*dt, 3);
      k4x := ddt(t + dt, x + k3x*dt, y + k3y*dt, z + k3z*dt, 1);
      k4y := ddt(t + dt, x + k3x*dt, y + k3y*dt, z + k3z*dt, 2);
      k4z := ddt(t + dt, x + k3x*dt, y + k3y*dt, z + k3z*dt, 3);
      x := x + (dt / 6.0) * (k1x + 2*k2x + 2*k3x + k4x);
      y := y + (dt / 6.0) * (k1y + 2*k2y + 2*k3y + k4y);
      z := z + (dt / 6.0) * (k1z + 2*k2z + 2*k3z + k4z);

      if cbAddNoise.Checked then
      begin
        noise := (Random * 2 - 1) * noise_amp;
        z := z + noise;
      end;
      Series7.AddXY(t, z);
      t := t + dt;
    end;
  finally
    Series7.EndUpdate;
  end;
end;
```

This procedure is a private procedure (numerical integrator) that will numerically integrate the 3-D dynamical system using classical 4th-order Runge–Kutta (RK4) at sampling frequency *fecg*, update *current_rr_interval* at beat boundaries using *rr_final*, optionally add measurement noise, and output ECG samples in *Series7*. It also have a noise amplitude reading, where if checkbox *cbAddNoise.Checked* then *noise_amp := StrToFloat(edtNoiseAmp.Text)* or else *noise_amp := 0*. And if *cbAddNoise.Checked* then *noise := (Random * 2 - 1) * noise_amp; z := z + noise;*

The equations used RK4 as standard:

$$k_1 = f(t,u),\ k_2 = f(t + dt/2, u + dtk_1/2),\ k_3 = \cdots,\ u(t + dt) = u(t) + \frac{dt}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where f is provided by ddt. angfreq(t) is referenced within ddt to form $\omega(t)$.

j. *function TForm1.ddt(t, x, y, z: Double; trig: Integer): Double*

```
function TForm1.ddt(t, x, y, z: Double; trig: Integer): Double;
var
  alpha, teta, delta_teta, z_sum, z_base: Double;
  i: TWaveIndex;
  wander_amp, wander_freq: Double;
begin
  alpha := 1.0 - Sqrt(Sqr(x) + Sqr(y));
  case trig of
    1: Result := alpha * x - angfreq(t) * y;
    2: Result := alpha * y + angfreq(t) * x;
    3: begin
      teta := ArcTan2(y, x);
      z_sum := 0;
      for i := Low(TWaveIndex) to High(TWaveIndex) do
      begin
        delta_teta := modulus(teta - theta_i[i], 2 * Pi);
        if delta_teta > Pi then
          delta_teta := delta_teta - 2 * Pi;
        z_sum := z_sum - a_i[i] * delta_teta * Exp(-0.5 * Sqr(delta_teta) /
    Sqr(b_i[i]));
      end;

      if cbBaselineWander.Checked then
      begin
        try
          wander_amp := StrToFloat(edtWanderAmp.Text);
          wander_freq := StrToFloat(edtWanderFreq.Text);
          z_base := wander_amp * Sin(2 * Pi * wander_freq * t);
        except
          z_base := 0.0;
        end;
      end else
        z_base := 0.0;
      Result := z_sum - (z - z_base);
    end
  else
    Result := 0;
  end;
end;
```

This is a function returning derivative component (three-mode dispatch) that will computes derivatives for the 3-D system. When *trig=1* returns $\dot{x}$; *trig=2* returns $\dot{y}$; *trig=3* returns $\dot{z}$. This single function supplies the derivative values needed by RK4. If trig=1: result $\dot{x} = \alpha \cdot x - \omega(t) \cdot y$. If trig=2: result $\dot{y} = \alpha \cdot y + \omega(t) \cdot x$. If trig=3: compute θ, then it will sum over morphological kernels. Then *z_base* optionally set to *wander_amp * Sin(2 * Pi * wander_freq * t)* if baseline wander is enabled, final *Result := z_sum - (z - z_base);*. The mathematical form:

$$\dot{z} = -\sum_i a_i \Delta\theta_i \exp(-\frac{\Delta\theta_i^2}{2b_i^2}) \ - \ (z - z_{\text{base}}(t)).$$

where *Deltaθ* is wrapped to $(-\pi, \pi]$ using modulus. The *-(z - z_base)* term is a first-order relaxation pulling z back to baseline. *z_base* may be a slow sinusoid.

k. function TForm1.angfreq(t: Double): Double

```
function TForm1.angfreq(t: Double): Double;
```

```
begin
  if (current_rr_interval > 1E-9) then
    Result := 2.0 * Pi / current_rr_interval
  else
    Result := 2.0 * Pi / (60.0 / hmean);
end;
```

This function will return instantaneous angular frequency $\omega(t) = \frac{2\pi}{T(t)}$ using global *current_rr_interval*; fallback to *60/hmean* if *current_rr_interval* is invalid. Equation used: $\omega(t) = 2\pi/$current_rr_interval.

l. *function TForm1.modulus(in1, in2: Double): Double*

```
function TForm1.modulus(in1, in2: Double): Double;
begin
  Result := fmod(in1, in2);
  if Result < 0 then
    Result := Result + in2;
end;
```

This function will compute *in1 mod in2* as floating-point remainder, ensuring a result in *[0, in2)*. It uses *fmod* and corrects negative remainder by adding *in2*.

m. *procedure TForm1.CalculateAndDisplayHRV*

```
procedure TForm1.CalculateAndDisplayHRV;
var
  i, nn50_count, end_j: Integer;
  sum, mean, sum_sq_diff, sdnn, rmssd, diff, sdann: Double;
  segment_averages: array of Double;
  segment_sum: Double;
  segment_count, segment_length, j: Integer;
  total_duration: Double;
  warning_msg: string;
begin
  if Length(rr_final) < 2 then
  begin
    edtSDNN.Text  := 'N/A';
    edtSDANN.Text := 'N/A';
    edtRMSSD.Text := 'N/A';
    edtNN50.Text  := 'N/A';
    edtpNN50.Text := 'N/A';
    Exit;
  end;

  total_duration := 0;
  for i := 0 to Length(rr_final) - 1 do
    total_duration := total_duration + rr_final[i];

  if total_duration < 300 then
    warning_msg := ' (< 5 min, for reference only)'
  else
```

```
    warning_msg := '';

  sum := 0;
  for i := 0 to Length(rr_final) - 1 do
    sum := sum + rr_final[i];
  mean := sum / Length(rr_final);

  sum_sq_diff := 0;
  for i := 0 to Length(rr_final) - 1 do
    sum_sq_diff := sum_sq_diff + Sqr(rr_final[i] - mean);
  sdnn := Sqrt(sum_sq_diff / (Length(rr_final) - 1));

  sum_sq_diff := 0;
  nn50_count := 0;
  for i := 1 to Length(rr_final) - 1 do
  begin
    diff := rr_final[i] - rr_final[i-1];
    sum_sq_diff := sum_sq_diff + Sqr(diff);
    if Abs(diff) * 1000 > 50 then
      Inc(nn50_count);
  end;
  rmssd := Sqrt(sum_sq_diff / (Length(rr_final) - 1));

  if hmean > 0 then
    segment_length := Round(60 / (60 / hmean))
  else
    segment_length := 60;

  if segment_length > 0 then
    segment_count := Length(rr_final) div segment_length
  else
    segment_count := 0;

  if (segment_count > 1) then
  begin
    SetLength(segment_averages, segment_count);
    for i := 0 to segment_count - 1 do
    begin
      segment_sum := 0;
      end_j := Min(segment_length - 1, Length(rr_final) - (i * segment_length) - 1);
      for j := 0 to end_j do
        segment_sum := segment_sum + rr_final[i * segment_length + j];
      if end_j > -1 then
        segment_averages[i] := segment_sum / (end_j + 1);
    end;

    sum := 0;
    for i := 0 to segment_count - 1 do sum := sum + segment_averages[i];
    mean := sum / segment_count;
    sum_sq_diff := 0;
    for i := 0 to segment_count - 1 do sum_sq_diff := sum_sq_diff + Sqr(segment_averages[i] - mean);
    sdann := Sqrt(sum_sq_diff / (segment_count - 1));
    edtSDANN.Text := FloatToStrF(sdann * 1000, ffFixed, 8, 2) + warning_msg;
  end
  else
  begin
    edtSDANN.Text := 'N/A (not enough data)';
  end;

  edtSDNN.Text := FloatToStrF(sdnn * 1000, ffFixed, 8, 2) + warning_msg;
  edtRMSSD.Text := FloatToStrF(rmssd * 1000, ffFixed, 8, 2);
  edtNN50.Text := IntToStr(nn50_count);
  if Length(rr_final) > 1 then
    edtpNN50.Text := FloatToStrF((nn50_count / (Length(rr_final) - 1)) * 100, ffFixed, 8, 2)
  else
    edtpNN50.Text := 'N/A';
end;
```

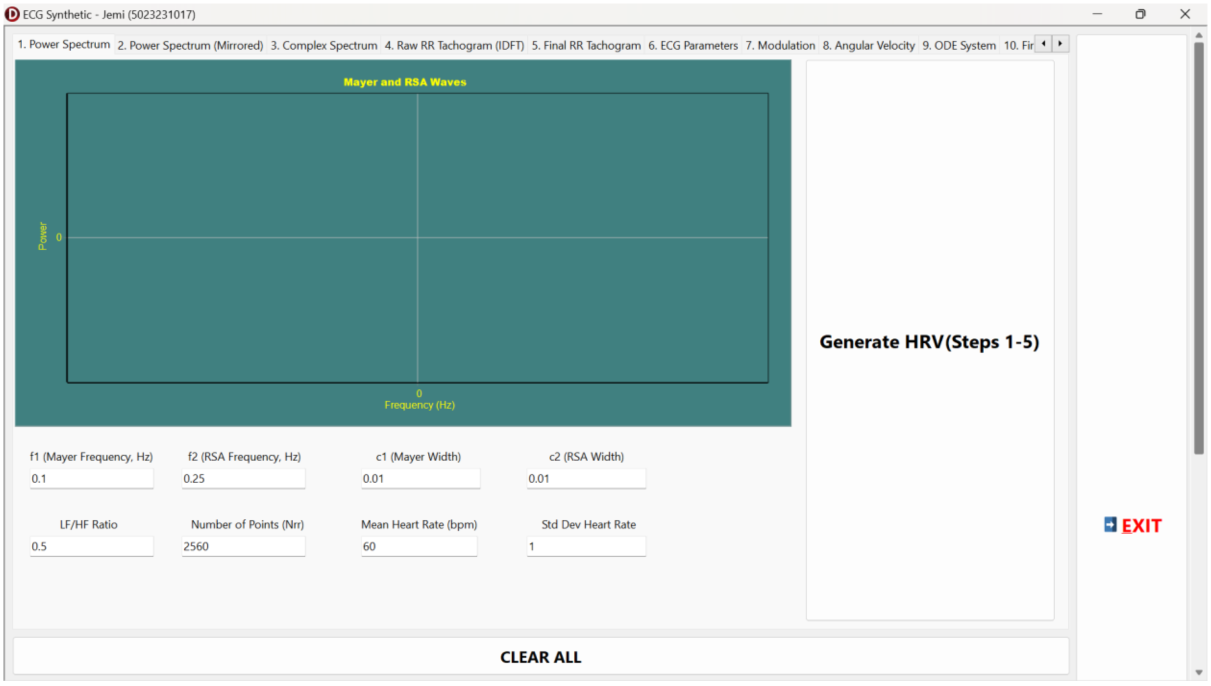
This procedure is private procedure (HRV metrics) that will compute time-domain HRV metrics from *rr_final[]* (per-beat RR in seconds): Mean RR, SDNN, RMSSD, NN50, pNN50, SDANN; display results in UI edit boxes and add a warning if total duration < 5 minutes. The algorithm is as follows:

1. Guard clause, if *Length(rr_final) < 2* then set displays to 'N/A' and exit.
2. Total duration, sum RR to get *total_duration* and if *total_duration < 300* then set *warning_msg := ' (< 5 min, for reference only)'.*
3. Mean RR that is *mean := sum(rr_final)/N.*
4. SDNN (standard deviation of NN intervals): SDNN = $1000\sqrt{\frac{1}{N-1}\sum(RR_i - \bar{RR})^2}$
5. RMSSD that compute successive differences *diff := rr_final[i] - rr_final[i-1]*, accumulate *sum_sq_diff += diff^2*, count *nn50_count* for *|diff|*1000 > 50.* Equation: RMSSD = $1000\sqrt{\frac{1}{N-1}\sum(\Delta_i)^2}$ (ms).
6. SDANN, the code computes *segment_length := Round(60 / (60 / hmean))*, this expression simplifies to 1 (and looks redundant) but intent seems to set segment length in beats equal to ~60. Then *segment_count := Length(rr_final) div segment_length*, compute segment averages and sd of those averages.

There are several issues however regarding the HRV Time-Domain analysis, that is the *segment_length* computation appears to be a bug/oddity and may not yield conventional 5-minute segments. And for real SDANN use, the segment length in *beats per 5 minutes ≈ 5*60 / mean_RR*. Or better, use partition by time not by fixed beat counts. It will be considered to fixing to time-based windows and compute number of beats whose cumulative RR fills 5 minutes, then average those windows.

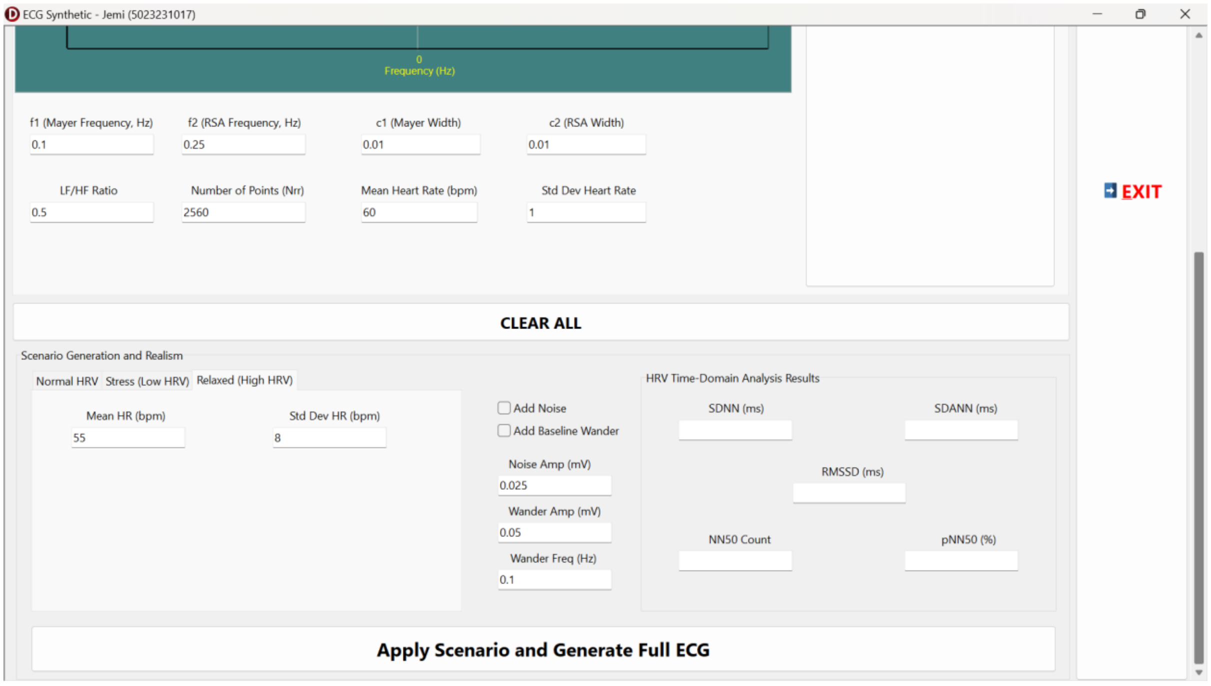
2.3. Result of the Program and Analysis

*Initial GUI



f1 (Mayer Frequency, Hz): 0.1
f2 (RSA Frequency, Hz): 0.25
c1 (Mayer Width): 0.01
c2 (RSA Width): 0.01
LF/HF Ratio: 0.5
Number of Points (Nrr): 2560
Mean Heart Rate (bpm): 60
Std Dev Heart Rate: 1

Figure 2.1. *Initial GUI Components for ECG Synthetic Program (1)*


ECG Synthetic - Jemi (5023231017)
0
Frequency (Hz)
EXIT
CLEAR ALL
Scenario Generation and Realism
Normal HRV
Stress (Low HRV)
Relaxed (High HRV)
HRV Time-Domain Analysis Results
Apply Scenario and Generate Full ECG

f1 (Mayer Frequency, Hz): 0.1
f2 (RSA Frequency, Hz): 0.25
c1 (Mayer Width): 0.01
c2 (RSA Width): 0.01
LF/HF Ratio: 0.5
Number of Points (Nrr): 2560
Mean Heart Rate (bpm): 60
Std Dev Heart Rate: 1

Mean HR (bpm): 55
Std Dev HR (bpm): 8

☐ Add Noise
☐ Add Baseline Wander
Noise Amp (mV): 0.025
Wander Amp (mV): 0.05
Wander Freq (Hz): 0.1

SDNN (ms):
SDANN (ms):
RMSSD (ms):
NN50 Count:
pNN50 (%):

Figure 2.2. *Initial GUI Components for ECG Synthetic Program (2)*

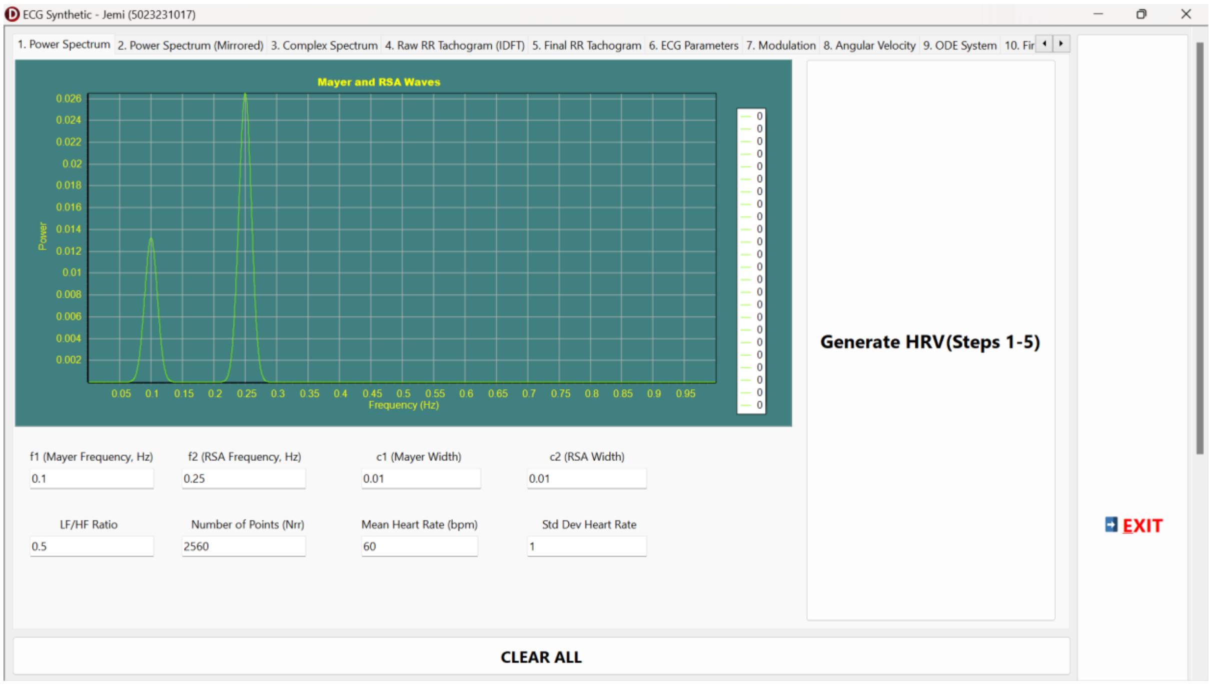
As seen in the images above, it is the initial GUI components where the ECG Signal generation and scenario generation are taking place. It have 2 distinct TGroupBox components, the components in Figure 2.1 as the ECG Signal generation frame and the other below as the ECG scenario generation frame.

So, the next section will explain the results obtained from the ECG Synthetic and Scenario Generation program.

2.3.1. ECG Signal Generation

1. Power Spectrum of Mayer and RSA Waves

The section begins by summarizing the spectral-to-time pipeline that produces the RR tachogram and then the ECG waveform. The power spectrum shown in Figure 2.3 illustrates the deliberate construction of two Gaussian-shaped spectral components intended to represent respiratory sinus arrhythmia (RSA, high-frequency) and Mayer-like oscillations (low-frequency). Mathematically the design is the sum of two Gaussians, $S(f) = A_1\mathcal{N}(f; f_1, \sigma_1^2) + A_2\mathcal{N}(f; f_2, \sigma_2^2)$, where $\mathcal{N}$ denotes the normalized Gaussian kernel (see Chapter I). In the program this operation is implemented in the HRV synthesis routine (the power-spectrum loop in *btnGenerateHRVClick*), which evaluates the two component values at each discrete frequency bin and stores them in the array used to form the magnitude spectrum. The visual peak separation and relative amplitudes in the plotted spectrum confirm the code correctly builds the prescribed spectral energy distribution.



f1 (Mayer Frequency, Hz): 0.1
f2 (RSA Frequency, Hz): 0.25
c1 (Mayer Width): 0.01
c2 (RSA Width): 0.01
LF/HF Ratio: 0.5
Number of Points (Nrr): 2560
Mean Heart Rate (bpm): 60
Std Dev Heart Rate: 1

Figure 2.3. *The Power Spectrum of Mayer and RSA Waves Generation*

2. Mirrored Power Spectrum of Mayer and RSA Waves

Figure 2.4 shows the mirrored (two-sided) magnitude pattern used to guarantee conjugate symmetry for a real-valued inverse transformation. The program takes the square root of the one-sided power $M(f) = \sqrt{S(f)}$ and assembles a symmetric magnitude array (the *sw* array) before attaching random phases. This mirroring step is visible in the code block that fills *sw[i]* for indices above and below the Nyquist half-point. The mirroring preserves the desired spectral envelope while ensuring the IDFT returns a real tachogram.

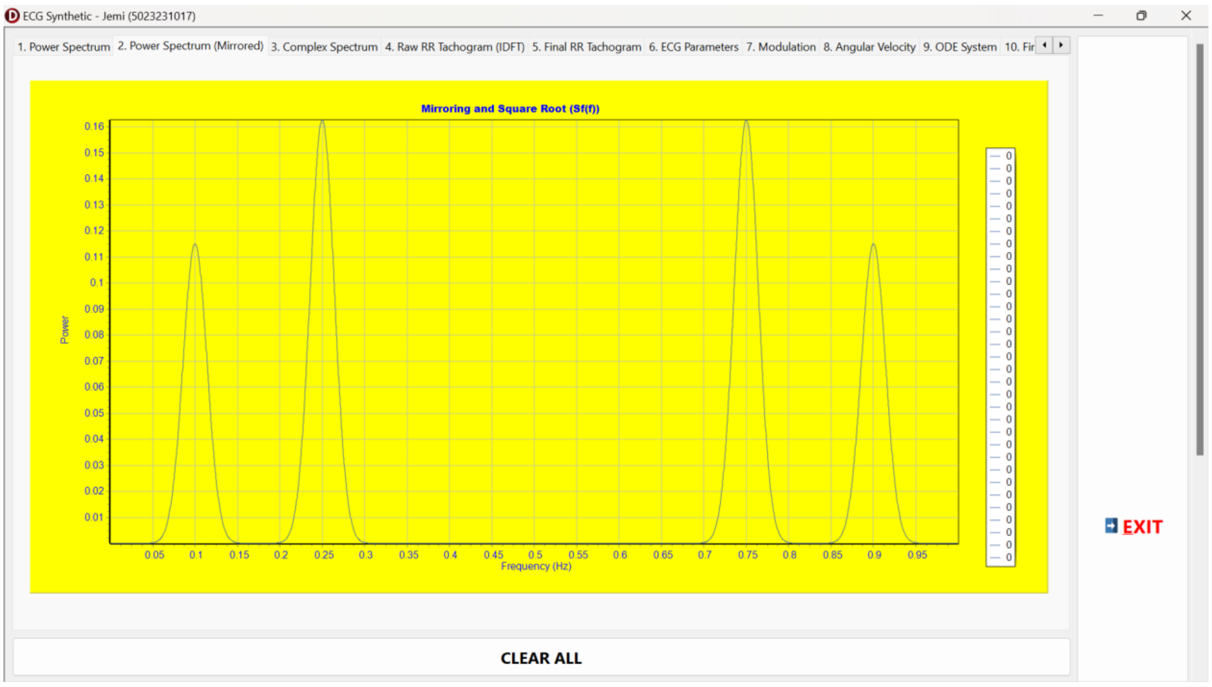


Figure 2.4. *The Mirrored Version of Power Spectrum of Mayer and RSA Waves*

3. Complex Spectrum (Generating Random Phase)

Figure 2.5 presents the complex-spectrum view after random-phase assignment: each positive-frequency magnitude $M(k)$receives an independent random phase ϕ_k, producing complex coefficients $C(k) = M(k)e^{j\phi_k}$. The code implements this by computing *reall[i] := sw[i]*Cos(phase) and imag[i] := sw[i]*Sin(phase)*. Assigning random phases yields realistic beat to beat jitter in the time-domain realization while preserving the designed power distribution. The figure therefore verifies the intermediate complex coefficients before inverse transform.

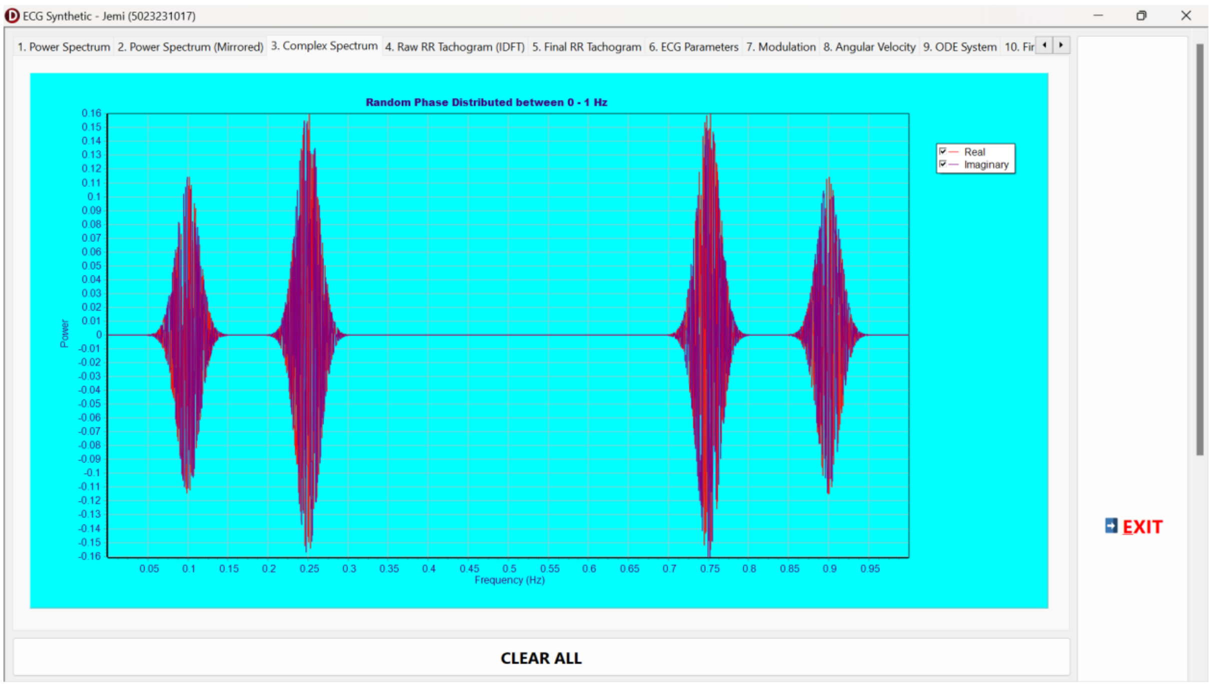
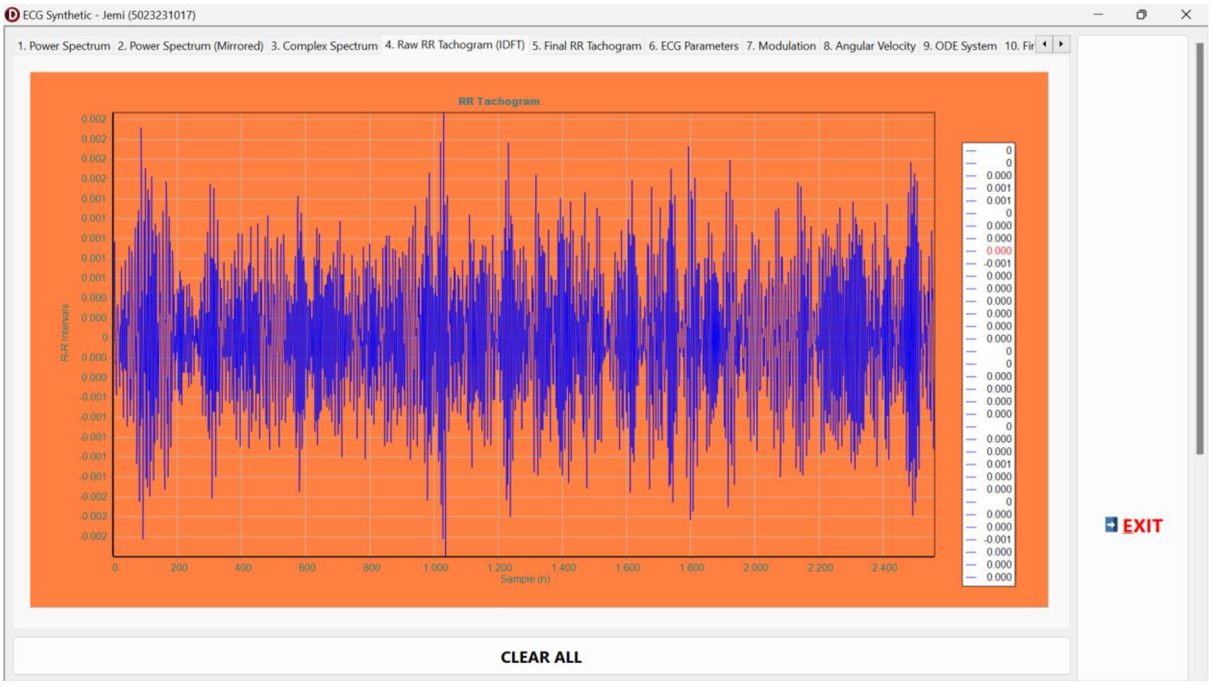


Figure 2.5. *The Mirrored Version of Power Spectrum of Mayer and RSA Waves*

4. Raw RR Tachogram (from IDFT)

The raw RR tachogram in Figure 2.6 is the direct inverse-DFT (IDFT) result of those complex coefficients. The IDFT used by the program follows the formula $r_{\text{raw}}[n] = (1/N)\sum_{k=0}^{N-1} C[k]e^{j2\pi kn/N}$ and the code computes the real part explicitly with nested summations (the *idft_raw* loop in *btnGenerateHRVClick*). The plot shows zero-mean oscillations whose envelope reflects the two spectral peaks; this is expected because the IDFT of the designed spectrum reconstructs the targeted LF/HF beats fluctuations. In practice the program next computes raw mean and raw standard deviation and rescales the oscillation to the requested HR mean and HRV magnitude, following the scaling formula $RR[n] = \bar{RR} + (\sigma_{\text{target}}/\sigma_{\text{raw}})\,(r_{\text{raw}}[n] - \mu_{\text{raw}})$, so Figure 2.6 is the intermediate output prior to that shift and scaling.

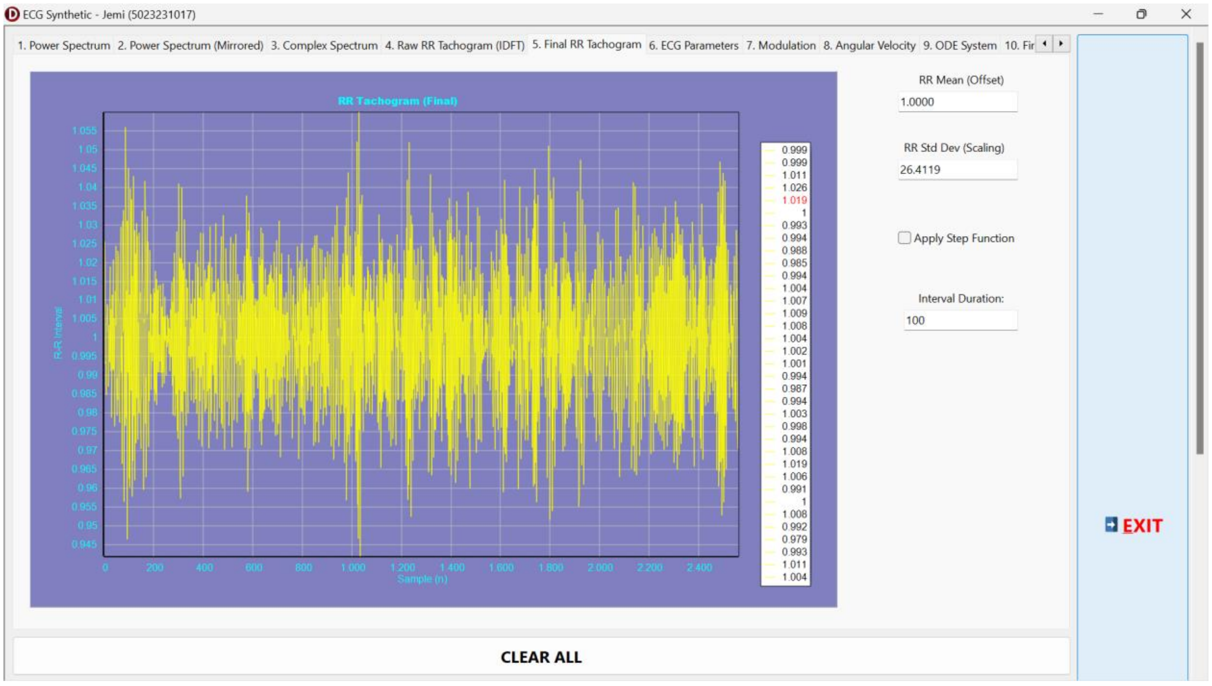


```
0
0
0.000
0.001
0.001
0
0.000
0.000
0.000
-0.001
0.000
0.000
0.000
0.000
0.000
0.000
0
0
0.000
0.000
0.000
0.000
0
0.000
0.000
0.001
0.000
0.000
0
0.000
0.000
-0.001
0.000
0.000
0.000
```

Figure 2.6. *The Raw RR Tachogram from IDFT Conversion*

5. Final RR Tachogram

Figure 2.7 shows the final RR tachogram after mean offset and amplitude scaling have been applied and, optionally, after a random-phase seed variation. The final series that is the *rr_continuous* therefore represents the per-beat intervals in seconds, centered at $\bar{RR} = 60/\bar{HR}$ and with standard deviation equal to the scenario's target. The visible amplitude (beat to beat spread) and baseline confirm that the conversion from spectral design to a physiologically interpretable tachogram works as intended; the code stores this array and then passes it to *UpdateRRFinalPlot*.



RR Mean (Offset): 1.0000

RR Std Dev (Scaling): 26.4119

☐ Apply Step Function

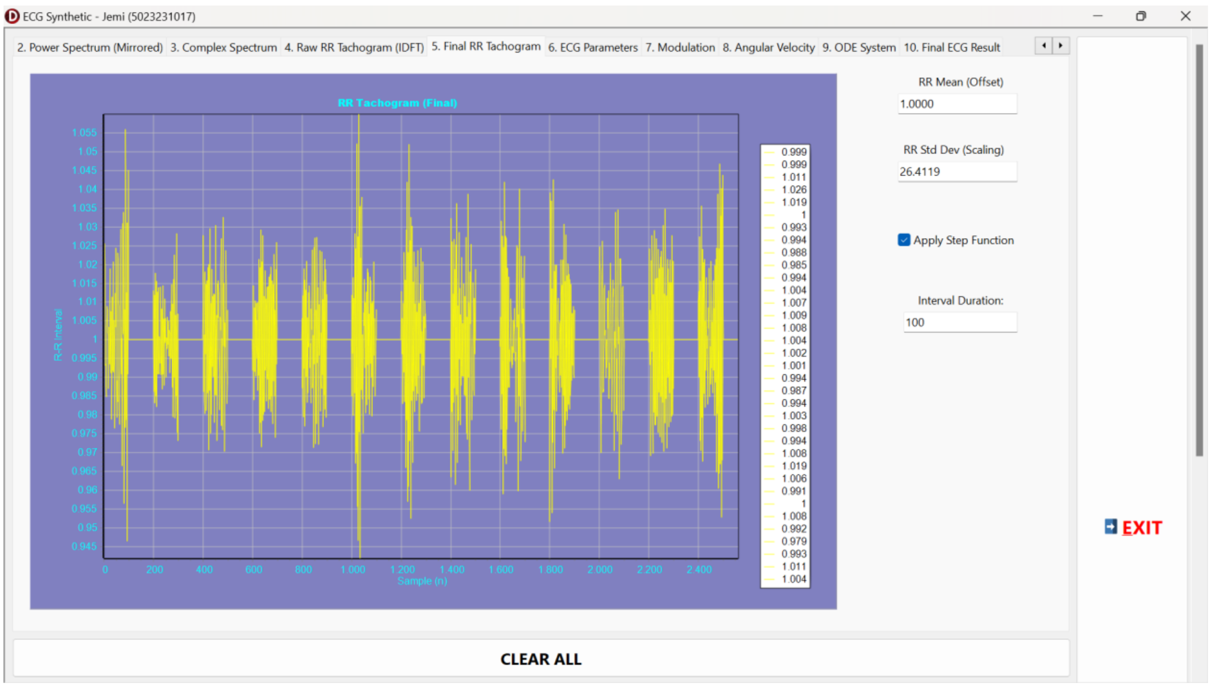
Interval Duration: 100

```
0.999
0.999
1.011
1.026
1.019
1
0.993
0.994
0.988
0.985
0.994
1.004
1.007
1.009
1.008
1.004
1.002
1.001
0.994
0.987
0.994
1.003
0.998
0.994
1.008
1.019
1.006
0.991
1
1.008
0.992
0.979
0.993
1.011
1.004
```

Figure 2.7. The Final RR Tachogram after using Random Noise Seed

6. Applying Step Function into Final RR Tachogram (Additional Final Project Demo)

Figure 2.8 documents an optional post-processing mode where a step/block function is applied to the RR series, that is the alternating blocks of beats are replaced by the mean RR (the *cbApplyStepFunction* logic in *UpdateRRFinalPlot*). The paragraph of code and the plot show the practical effect of toggling the block logic, so we can alternate between “variable” and “baseline” segments. This demo highlights how simple post-processing of the RR list changes the beat scheduling that ultimately drives $\omega(t) = 2\pi/T(t)$ and therefore the temporal structure of the synthetic ECG without modifying the underlying spectral construction.



RR Mean (Offset): 1.0000

RR Std Dev (Scaling): 26.4119

[✓] Apply Step Function

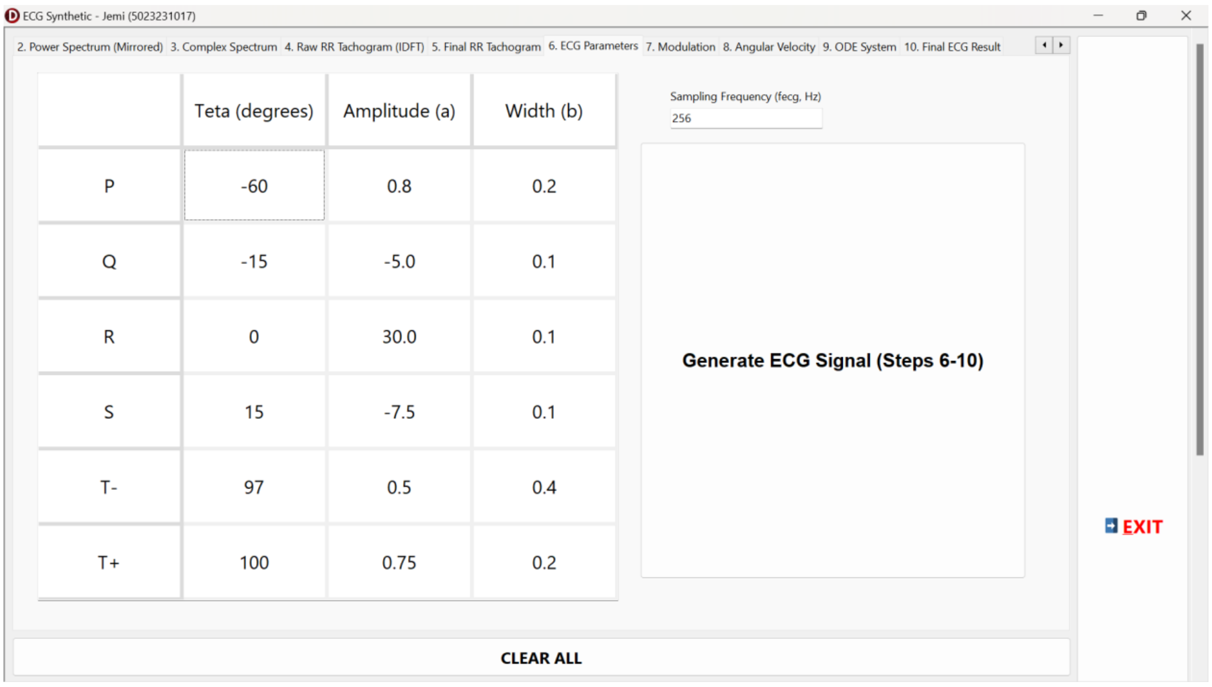
Interval Duration: 100

```
0.999
0.999
1.011
1.026
1.019
1
0.993
0.994
0.988
0.985
0.994
1.004
1.007
1.009
1.008
1.004
1.002
1.001
0.994
0.987
0.994
1.003
0.998
0.994
1.008
1.019
1.006
0.991
1
1.008
0.992
0.979
0.993
1.011
1.004
```

Figure 2.8. *The Final RR Tachogram after using Random Noise Seed and Multiplied by Step Function*

7. ECG Parameters Input Initialization

Figure 2.9 displays the ECG-parameter grid used to define base morphology for P, Q, R, S, T− and T+ waves (base phase angles, amplitudes and widths). These grid values are read at runtime by *btnGenerateECGClick*, which computes the modulation factor $\alpha = \sqrt{\bar{HR}/60}$ and then produces the final per-wave parameters using heuristics, that is a mixture of linear, square-root and power-law scalings shown in the code. The figure demonstrates the interface that allows the user to set the reference $\theta_{0,i}, a_{0,i}, b_{0,i}$and thereby tune the synthetic ECG morphology before scaling occurs.



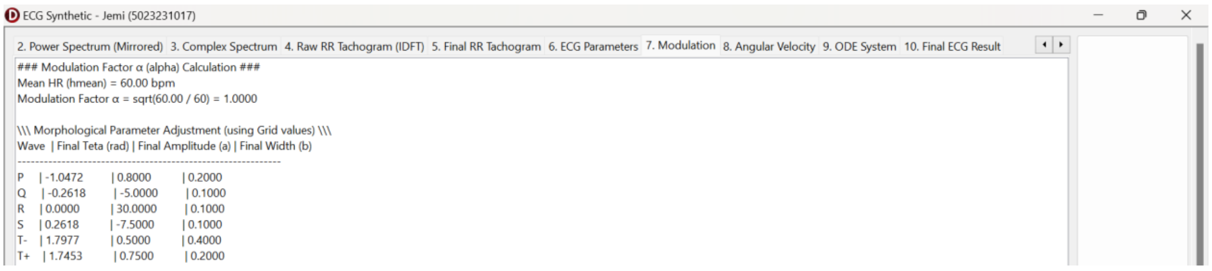
	Teta (degrees)	Amplitude (a)	Width (b)
P	-60	0.8	0.2
Q	-15	-5.0	0.1
R	0	30.0	0.1
S	15	-7.5	0.1
T-	97	0.5	0.4
T+	100	0.75	0.2

Sampling Frequency (fecg, Hz): 256

Figure 2.9. The ECG Parameters Input

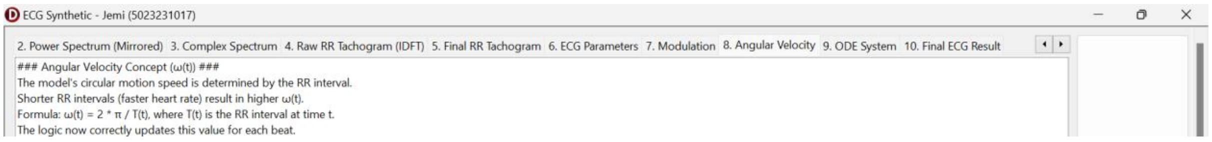
8. Modulation, Angular Velocity, and ODE System

Figures 2.10, 2.11, and 2.12 illustrate the modulation concept, the angular-velocity mapping, and the ordinary differential equation (ODE) system explanation. These images are the graphical counterpart of the equations in Chapter I: the oscillator's phase $\theta(t) = \operatorname{atan2}(y, x)$ evolves at the instantaneous angular speed $\omega(t) = 2\pi/T(t)$ (that computed in *angfreq*), and the z-equation applies localized Gaussian-like forcing to produce morphological peaks. Concretely the morphological forcing implemented in *ddt* is $F_{\text{morph}}(\theta) = -\sum_i a_i\, \Delta\theta_i \exp(-\Delta\theta_i^2/(2b_i^2))$, and the z-dynamics include a linear relaxation $-(z - z_{\text{base}})$.



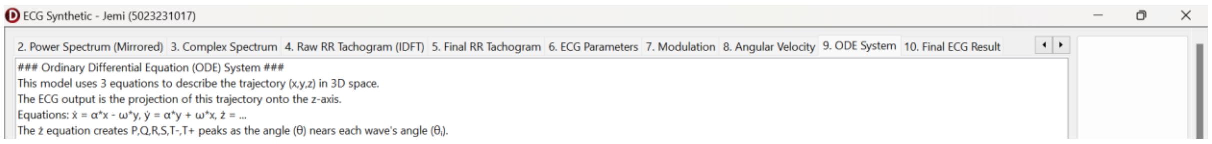
```
### Modulation Factor α (alpha) Calculation ###
Mean HR (hmean) = 60.00 bpm
Modulation Factor α = sqrt(60.00 / 60) = 1.0000

\\\ Morphological Parameter Adjustment (using Grid values) \\\
Wave  | Final Teta (rad) | Final Amplitude (a) | Final Width (b)
------------------------------------------------------------
P    | -1.0472        | 0.8000        | 0.2000
Q    | -0.2618        | -5.0000       | 0.1000
R    | 0.0000         | 30.0000       | 0.1000
S    | 0.2618         | -7.5000       | 0.1000
T-   | 1.7977         | 0.5000        | 0.4000
T+   | 1.7453         | 0.7500        | 0.2000
```

Figure 2.10. The Modulation Steps Description



```
### Angular Velocity Concept (ω(t)) ###
The model's circular motion speed is determined by the RR interval.
Shorter RR intervals (faster heart rate) result in higher ω(t).
Formula: ω(t) = 2 * π / T(t), where T(t) is the RR interval at time t.
The logic now correctly updates this value for each beat.
```

Figure 2.11. The Angular Velocity Concept Explanation



```
### Ordinary Differential Equation (ODE) System ###
This model uses 3 equations to describe the trajectory (x,y,z) in 3D space.
The ECG output is the projection of this trajectory onto the z-axis.
Equations: ẋ = α*x - ω*y, ẏ = α*y + ω*x, ż = ...
The ż equation creates P,Q,R,S,T-,T+ peaks as the angle (θ) nears each wave's angle (θᵢ).
```

Figure 2.12. *The ODE System Explanation*

9. Final ECG Result

In figure 2.13 is the program's final ECG output (z-projection) in the absence of added noise or baseline drift. The waveform displays clearly resolved P, Q, R, S and T feature whose relative timing and widths are set by the scaled $\theta_i, a_i, b_i$and whose beat-to-beat timing follows the synthesized RR sequence. The R-peak prominence and inter-beat spacing visually validate that the RK4 integrator (*RungeKutta4_ECG*) and the derivative function (*ddt*) correctly implement the ODE dynamics and the phase-triggered morphology kernels described in Chapter I. Because the integrator samples at the user-supplied ECG sampling frequency (*fecg*) and updates *current_rr_interval* on beat boundaries, the final ECG achieves physiological time scaling and morphological consistency across HR changes.

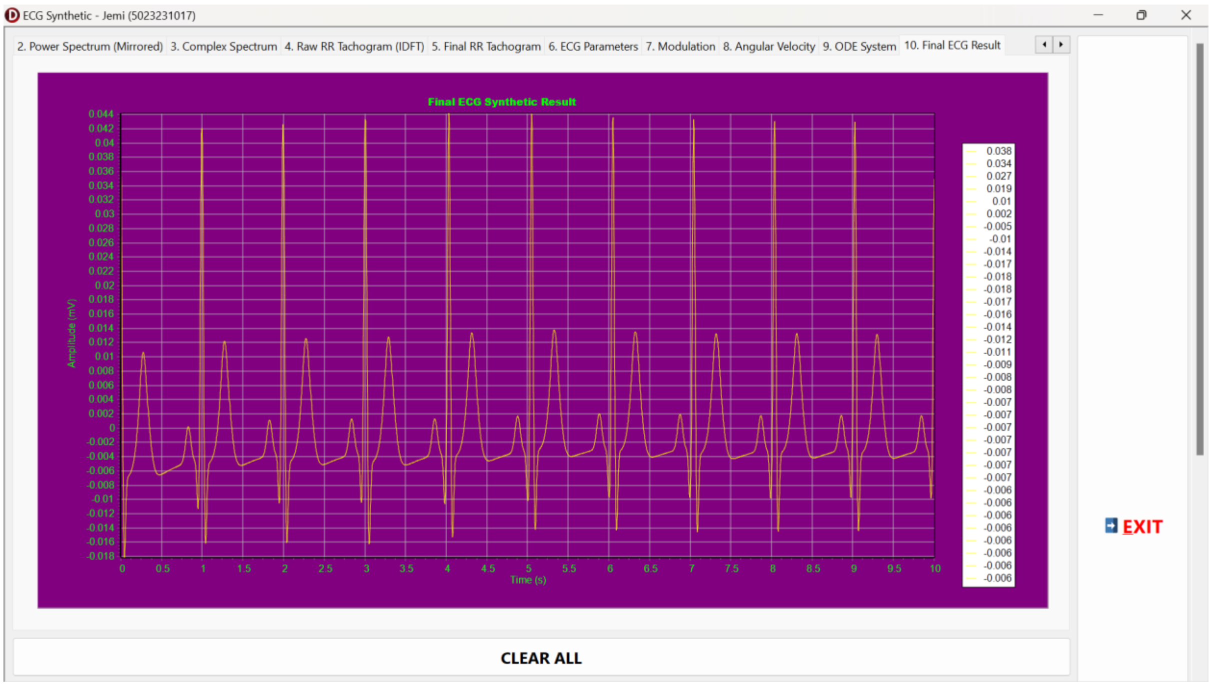


Figure 2.13. *The Final ECG Signal Result*

2.3.2. Scenario Generation of ECG Signal (Normal, Stress, Relax)

The scenario-generation subsection begins with the preset logic and intent, that are the three presets (Normal, Stress, Relax) change only the mean HR and the RR standard deviation, and optionally the LF/HF spectral ratio. The program's event *btnApplyScenarioClick* copies scenario values into the main parameters and re-runs the HRV to ECG pipeline so each scenario is produced identically except for the scenario parameters; this controlled experimental design makes comparisons valid. The HRV interpretation relies on standard metrics, that are SDNN, RMSSD, NN50 and pNN50 (equations defined in Chapter I and computed in *CalculateAndDisplayHRV*). For additional information, the Number of Points/samples used is Nrr = 76800 data and the sampling frequency of the ECG signal is 256 Hz. The motive behind this value input is to make sure that the minimal HRV Time-Domain interval is achieved, that is t = 300 second or 5 minutes.

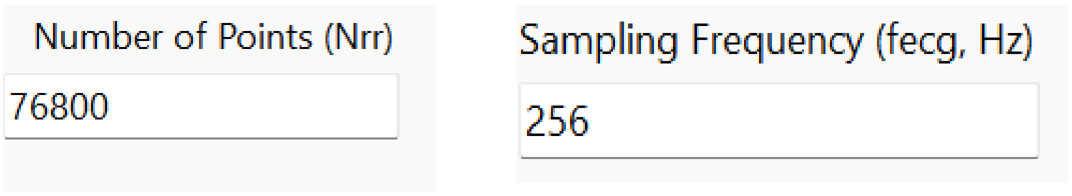
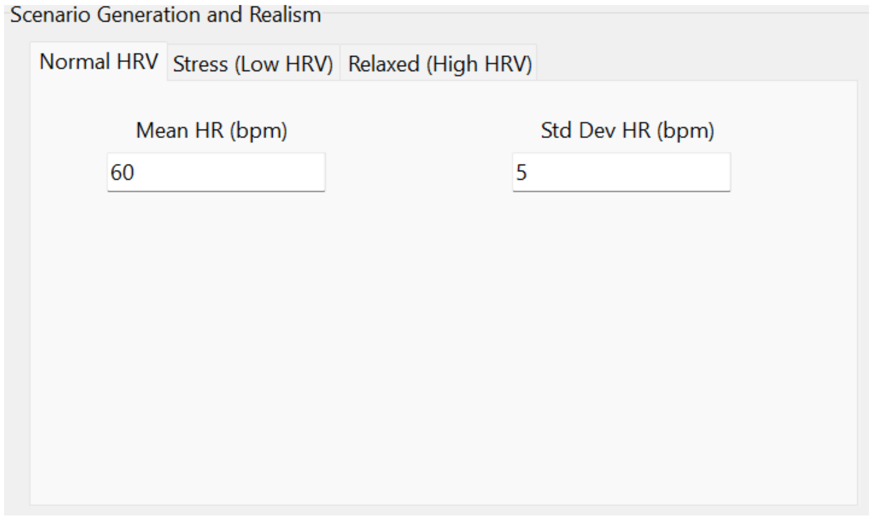

Number of Points (Nrr)
76800
Sampling Frequency (fecg, Hz)
256

Figure 2.14. *The Nrr Value used on the ECG Scenario Generation with Fs=256*

1. Normal Condition

In the Normal case (Figures 2.15) the initial parameter panel shows a mean HR near 60 bpm and a moderate RR standard deviation.


Scenario Generation and Realism

Mean HR (bpm)
60
Std Dev HR (bpm)
5

Figure 2.15. *The Normal Condition Scenario Initial Values*

The ECG output (Figure 2.16) displays regular beats with moderate beat-to-beat variability and physiologically plausible P–Q–R–S–T morphology.

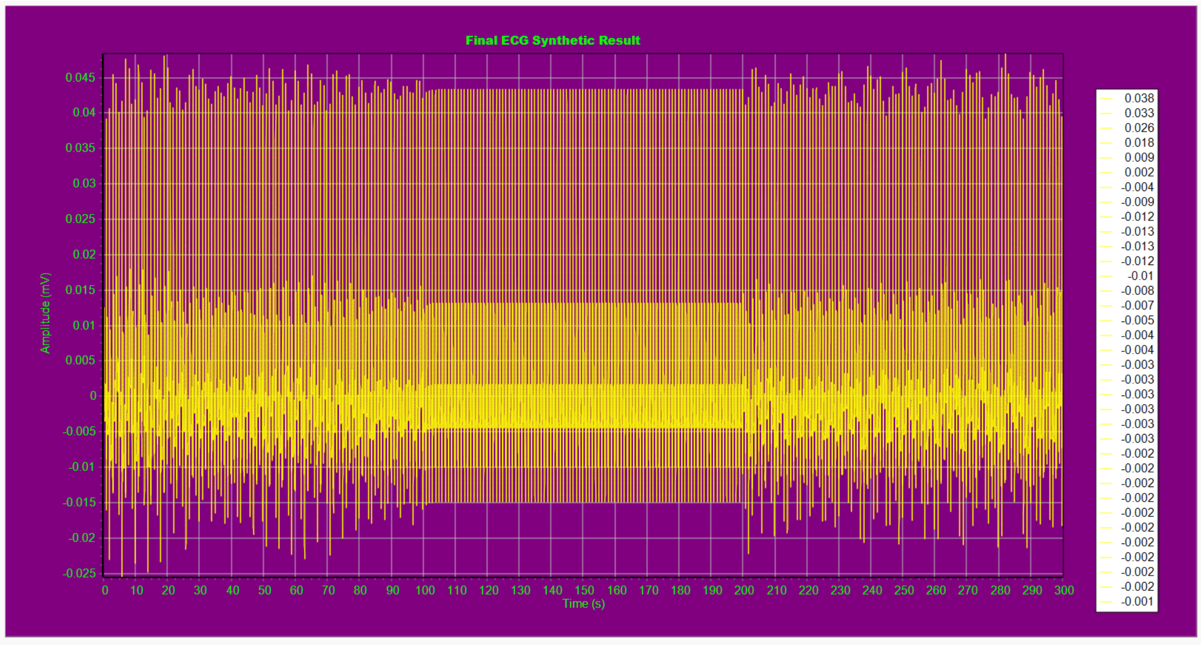


Figure 2.16. *The Normal Condition Scenario ECG Signal Output*

The zoomed view (Figure 2.17) emphasizes waveform shapes (QRS width, T-wave decay) that result from the chosen a_i, b_iafter α-scaling.

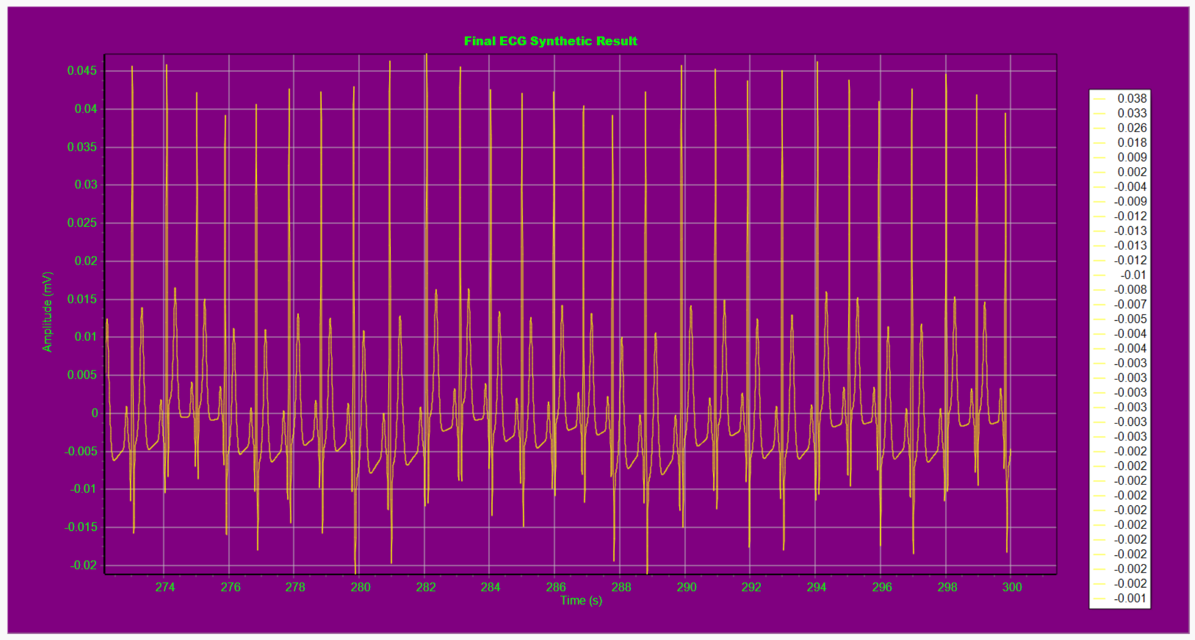
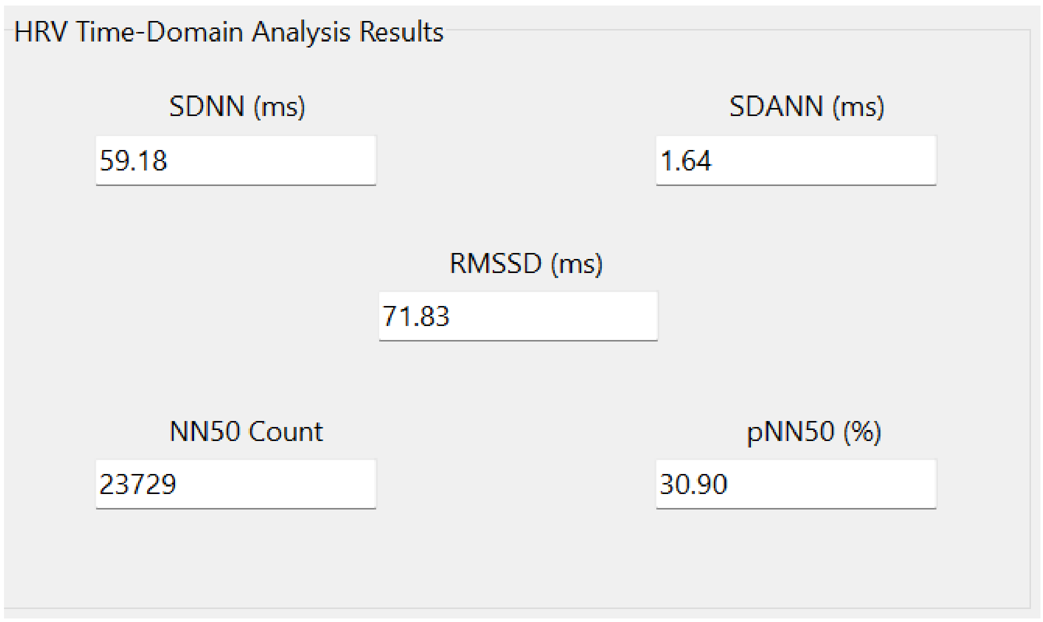


Figure 2.17. *The Normal Condition Scenario on Zoom*

The HRV time-domain plot (Figure 2.18) exhibits intermediate SDNN and RMSSD values consistent with a healthy resting pattern; these metrics are computed from the per-beat rr_final array by the formulae SDNN $= \sqrt{\frac{1}{N-1}\sum(RR_i - \bar{RR})^2}$ and RMSSD $= \sqrt{\frac{1}{N-1}\sum(\Delta_i)^2}$. Because the dataset length of the example may be shorter than clinical recommendations, the program appends a “<5 min, for reference only” note when appropriate. The Normal scenario therefore validates the pipeline under nominal conditions.



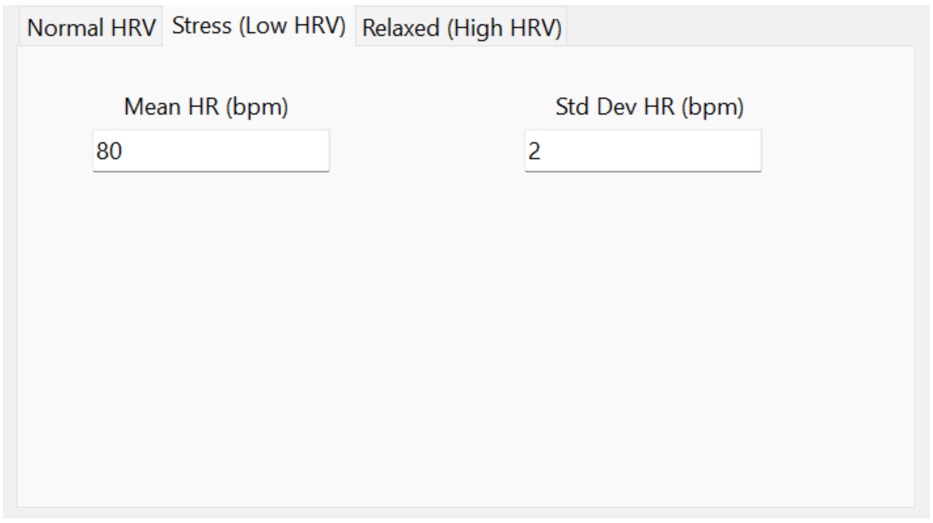
HRV Time-Domain Analysis Results

SDNN (ms)	59.18
SDANN (ms)	1.64
RMSSD (ms)	71.83
NN50 Count	23729
pNN50 (%)	30.90

Figure 2.18. *The Normal Condition Scenario HRV Time-Domain Output*

2. Stress Condition

In the Stress (low-HRV) scenario (Figures 2.19) the initial values indicate a higher mean HR (≈80 bpm) and a smaller RR standard deviation.



Mean HR (bpm)	80
Std Dev HR (bpm)	2

Figure 2.19. *The Stress Condition Scenario Initial Values*

The ECG trace (Figure 2.20) accordingly shows compressed beat spacing and, after $\alpha = \sqrt{\bar{HR}/60}$ scaling, slightly narrowed morphological widths. This is the direct effect of mapping mean HR into morphology via the rules coded in *btnGenerateECGClick.*

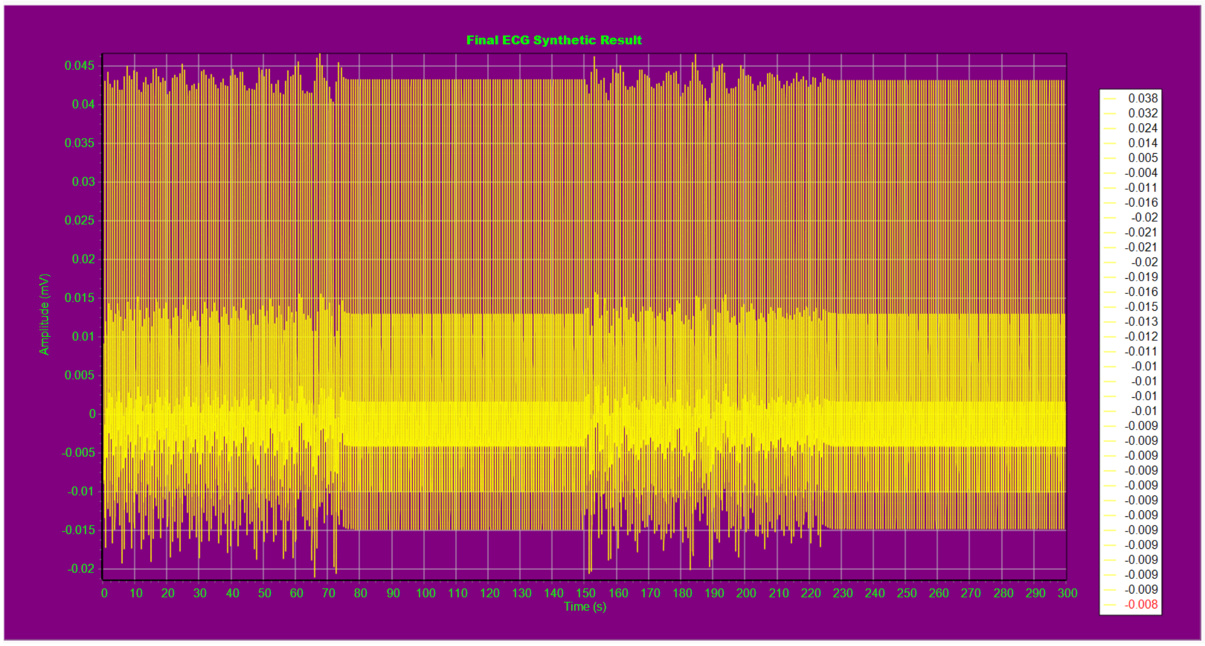


Figure 2.20. The Stress Condition Scenario ECG Signal Output

The zoom (Figure 2.21) highlights the compressed P–Q–R–S–T spacing and sometimes slightly increased R-rate, while the HRV output (Figure 2.22) shows reduced SDNN and RMSSD consistent with low vagal modulation; if one also reduces the HF spectral component in the spectral design step, the synthesized RR series shows less HF power, which matches physiological patterns of stress (reduced RSA). This scenario demonstrates how the pipeline can produce a controlled low-HRV condition by changing only a few parameters.

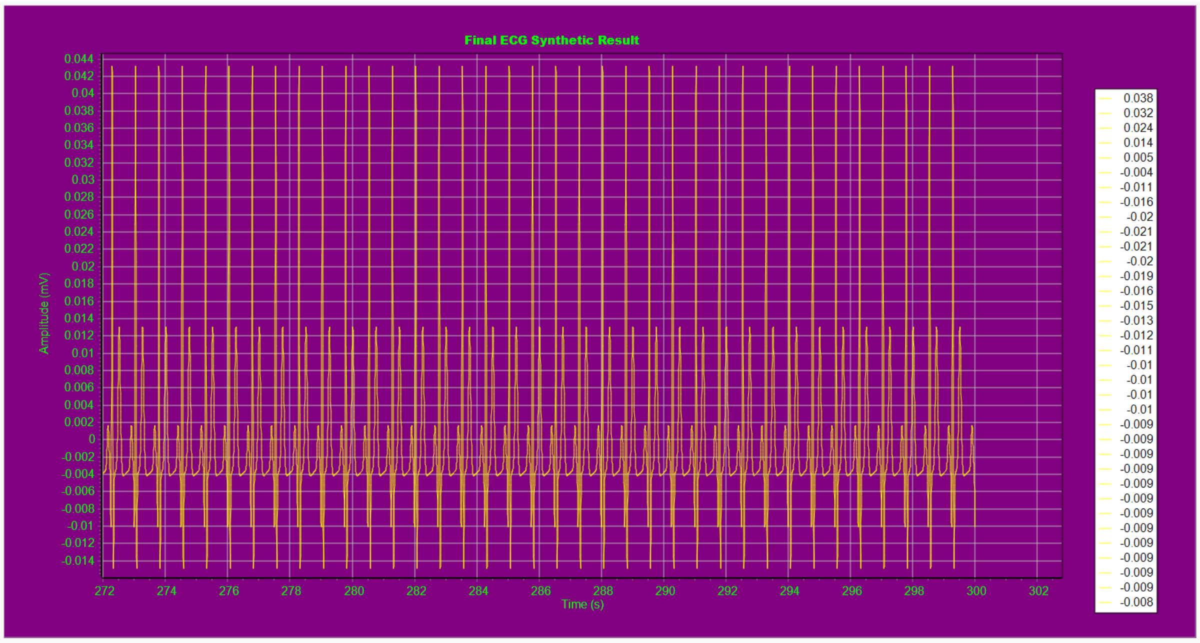
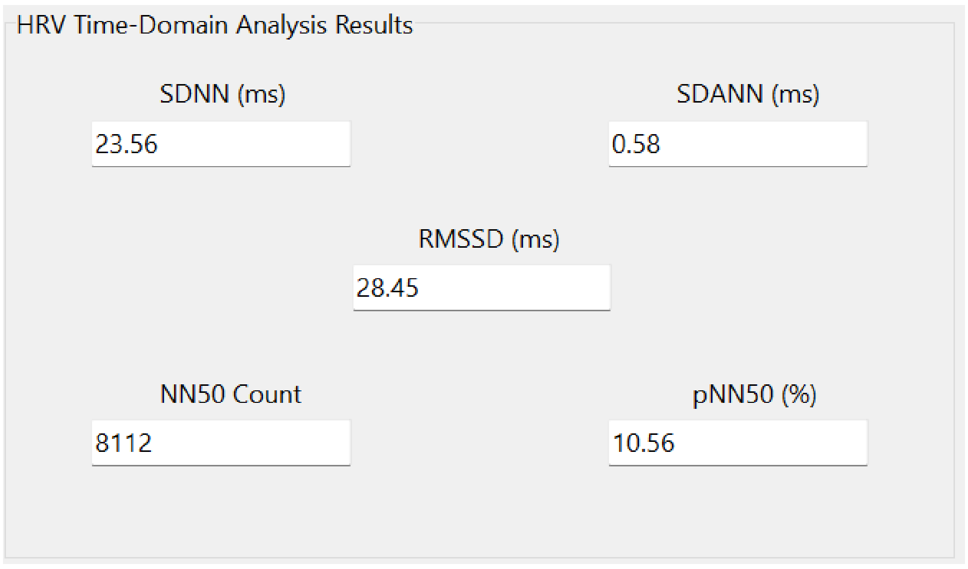


Figure 2.21. The Stress Condition Scenario on Zoom



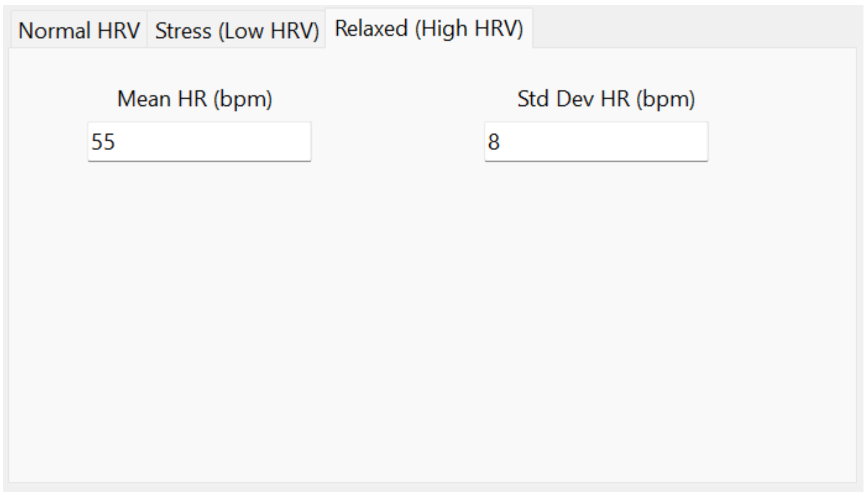
HRV Time-Domain Analysis Results

SDNN (ms)	SDANN (ms)
23.56	0.58
RMSSD (ms)	
28.45	
NN50 Count	pNN50 (%)
8112	10.56

Figure 2.22. *The Stress Condition Scenario HRV Time-Domain Output*

3. Relax Condition

In the Relaxed (high-HRV) scenario (Figures 2.23) the initial panel sets a lower mean HR (≈55 bpm) and a larger RR standard deviation.



Mean HR (bpm)	Std Dev HR (bpm)
55	8

Figure 2.23. *The Relax Condition Scenario Initial Values*

The synthesized ECG (Figure 2.24) shows wider inter-beat intervals and, after α-scaling, broadened P/Q/T components that reflect slower cardiac cycles.

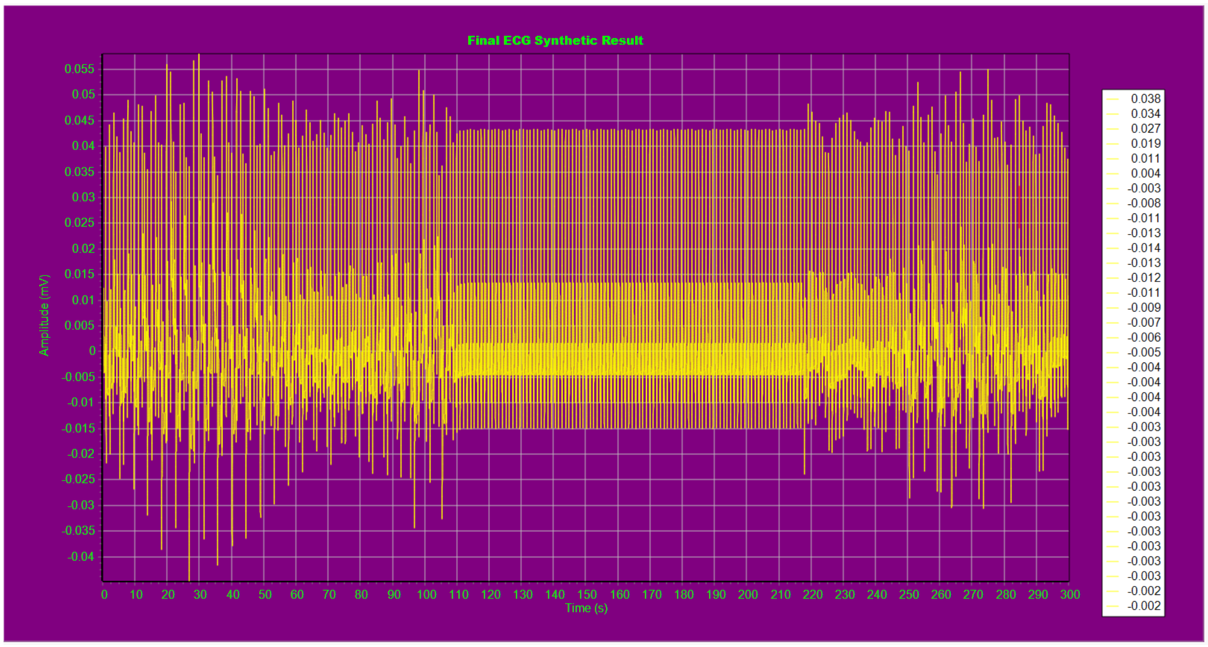


Figure 2.24. *The Relax Condition Scenario ECG Signal Output*

The zoom (Figure 2.25) highlights longer PR and QT-like intervals (relative to the other scenarios) and the HRV plot (Figure 2.26) displays increased SDNN and RMSSD values, for example, the per-beat RR series has larger amplitude fluctuations. This scenario confirms that raising the RR standard deviation and shifting mean RR produces the expected morphological and HRV changes when fed through the exact same dynamical oscillator.

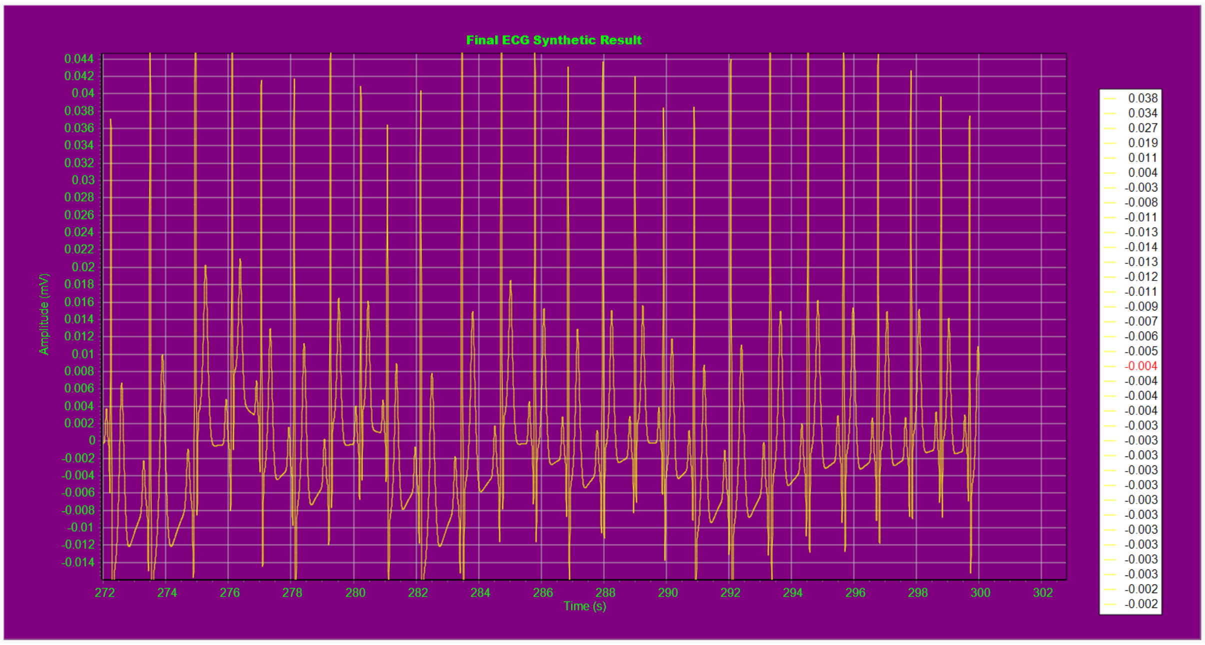
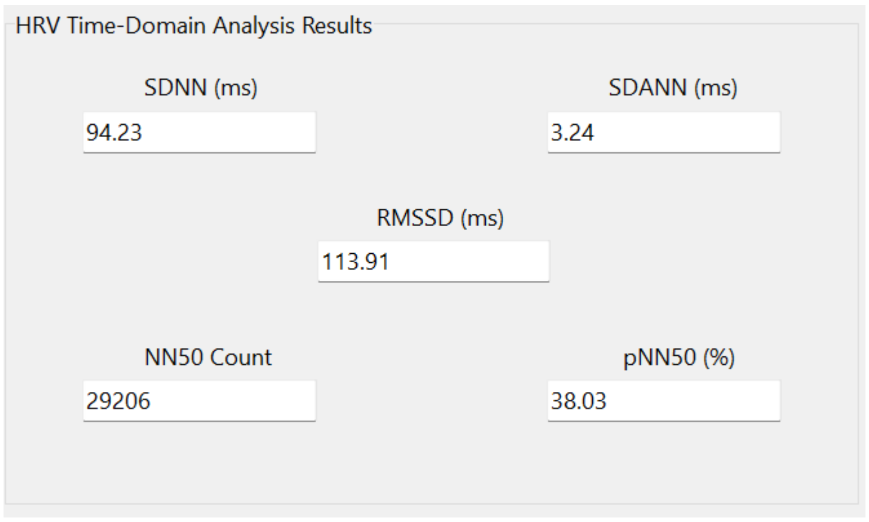


Figure 2.25. *The Relax Condition Scenario on Zoom*



HRV Time-Domain Analysis Results

SDNN (ms)
94.23

SDANN (ms)
3.24

RMSSD (ms)
113.91

NN50 Count
29206

pNN50 (%)
38.03

Figure 2.26. *The Relax Condition Scenario HRV Time-Domain Output*

2.3.3. The Noise and Baseline Wander Addition

The noise and baseline-wander experiments (Figures 2.27–2.29) show the controlled addition of measurement-like perturbations and their effect on waveform quality. The program implements baseline wander as a slow sinusoid $z_{\text{base}}(t) = A_{\text{wander}}\sin(2\pi f_{\text{wander}} t)$ that enters the z-relaxation term in *ddt*, and additive measurement noise $\eta(t)$that the integrator sometimes injects sample-wise into z when *cbAddNoise* is checked. Figure 2.27 (noise only) demonstrates small amplitude jitter of the baseline and minor R-peak amplitude variability.

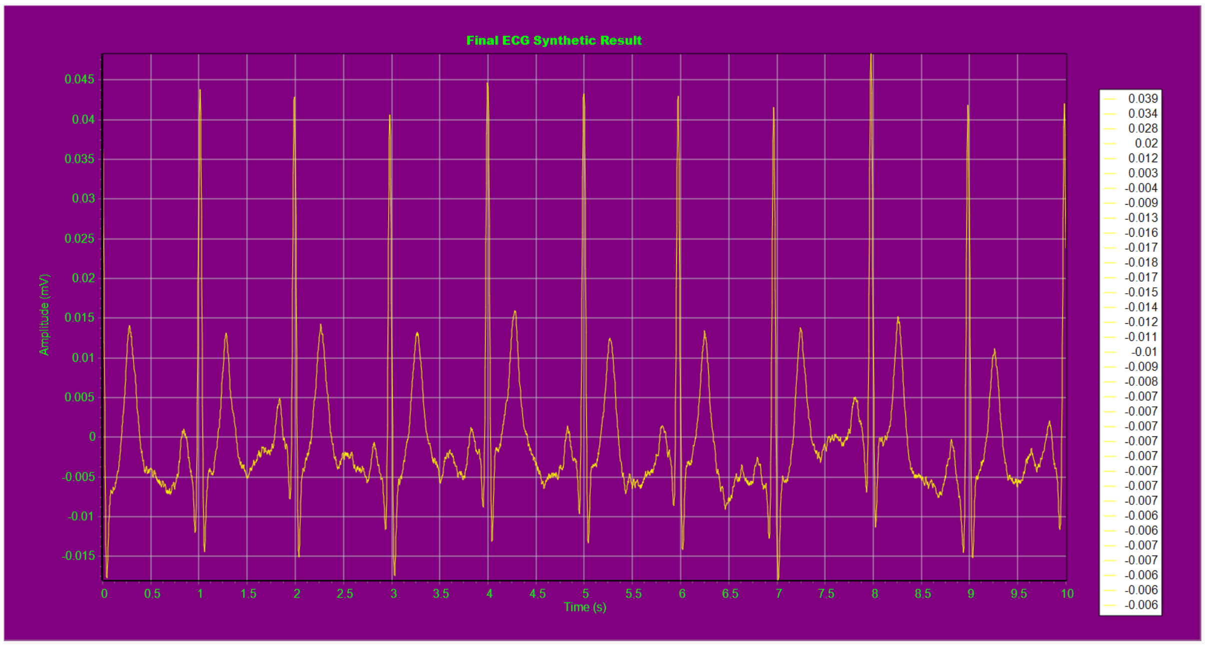


Figure 2.27. *The ECG Signal Output with only Noise Addition (Noise Amp = 0.0005 mV)*

Figure 2.28 (baseline only) shows slow low-frequency drift that will bias DC-coupled measurements and can confound naive QRS detectors.

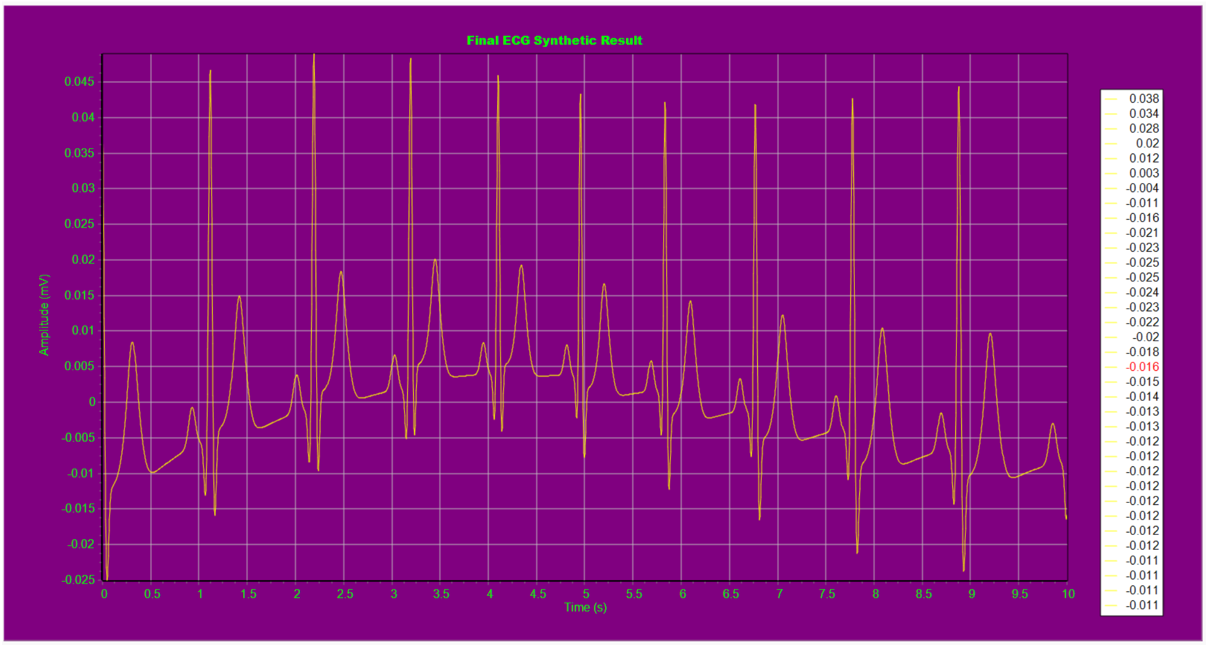


Figure 2.28. *The ECG Signal Output with only Baseline Wander Addition (Wander Amp = 0.005 mV, Wander Freq = 0.07 Hz)*

Figure 2.29 (both) combines both effects and is useful to evaluate filtering and detection robustness. These results illustrate how the program can be used to test downstream algorithms. For example, QRS detection or baseline removal using controlled synthetic inputs.

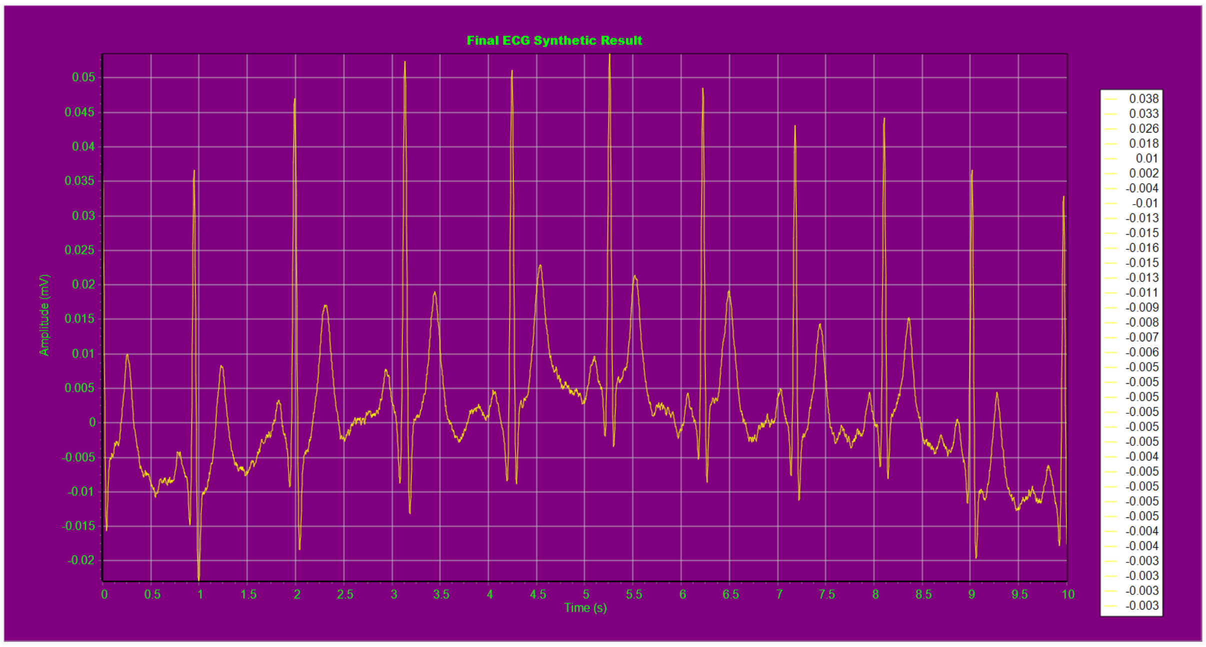


Figure 2.29. *The ECG Signal Output with Both Noise and Baseline Wander Addition (Baseline Amp = 0.0005 mV, Wander Amp = 0.005 mV, Wander Freq = 0.07 Hz)*

2.3.9. Evaluation

Based on the comprehensive results, the program is a robust, flexible and pedagogical ECG-synthesis framework that successfully implements the full pipeline from spectral design of RR variability through time-domain tachogram construction to phase-driven morphology generation and RK4 integration. It produces three physiologically distinct scenarios (Normal, Stress, Relaxed) and computes standard time-domain HRV metrics for quantitative comparison.

1. The code implements explicit power-spectrum design, the two Gaussian components for RSA and Mayer-like activity, random-phase assignment, IDFT synthesis and normalization to target mean RR and SD. The intermediate plots such as the magnitude spectrum, complex coefficients, raw IDFT, and scaled RR confirm each stage and make the processing transparent for validation and teaching.

2. The three scenario presets (Normal, Stress, Relaxed) are applied by changing only a small set of parameters (mean HR and RR standard deviation, optionally HF/LF ratio). That design produces predictable, physiologically plausible changes in ECG timing and HRV metrics (SDNN, RMSSD, NN50/pNN50), enabling controlled experiments and comparisons.

3. The 3-D limit-cycle oscillator plus phase-localized Gaussian kernels produces clear P–Q–R–S–T morphology whose timing compresses/expands with instantaneous frequency $\omega(t) = 2\pi/T(t)$. The modulation factor α maps mean HR to morphology, so shapes remain plausible across heart rates.

4. Using a classical fourth-order Runge–Kutta integrator yields numerically stable and accurate ECG traces at typical sampling rates. The code structure (separated HRV generator, morphology grid, integrator and HRV calculator) makes it straightforward to extend.

While the program is highly effective, it is important to acknowledge its limitations within the context of this implementation as follow:

1. Computational performance of IDFT**.** The inverse transform is computed via explicit nested summations ($O(N^2)$). For large tachogram sizes this is slow; replacing the naive IDFT with an FFT/IFFT will dramatically improve performance and scalability.

2. The SDANN segmentation logic currently uses a beat-count expression that does not robustly implement time-based 5-minute windows, this can produce misleading SDANN estimates. SDANN should be recomputed by partitioning by elapsed time (accumulate beats until 5 minutes are reached), not by a fixed beat count.

3. Modelling and reproducibility choices are empirical**.** The morphology scaling (different exponent choices for each wave) and kernel shapes are heuristic and not derived from a biophysical action-potential model; they work well visually but should be explicitly documented and, if clinical fidelity is required, validated against annotated ECG databases. Also, random-phase draws are not seeded by default, so two runs produce different realizations, add an optional RNG seed to support reproducible figures.

CHAPTER III. CONCLUSION

This project implemented a complete, end-to-end synthetic ECG generator that links physiologically motivated RR spectral design to a dynamical oscillator that produces realistic P–Q–R–S–T morphology, and it demonstrates controlled scenario synthesis for Normal, Stress (low-HRV) and Relaxed (high-HRV) conditions. Quantitative HRV time-domain metrics (SDNN, RMSSD, NN50/pNN50 and a segmented SDANN implementation) provide objective measures that reflect the scenario changes, while the RK4 integrator and modular code organization make the system numerically sound and easy to extend. The work validates the core conceptual theory from Chapter I and shows that modest, well-documented parameter changes produce predictable physiological effects. For future work, replacing the explicit IDFT with an FFT/IFFT, fixing the SDANN time-windowing to true 5-minute segments, adding a reproducible RNG seed, and comparing generated beats against annotated ECG databases would strengthen both performance and clinical validity. Overall, the program is a practical and educational tool for ECG synthesis, HRV experimentation, and method development in signal processing and cardiac modelling.

REFERENCES

[1] McSharry PE, Clifford GD, Tarassenko L, Smith LA. A dynamical model for generating synthetic electrocardiogram signals. IEEE Trans Biomed Eng. 2003 Mar;50(3):289–294.

[2] Task Force of the European Society of Cardiology and the North American Society of Pacing and Electrophysiology. Heart rate variability: standards of measurement, physiological interpretation, and clinical use. Circulation. 1996 Mar 1;93(5):1043–1065.

[3] Goldberger AL, Amaral LA, Glass L, Hausdorff JM, Ivanov PC, Mark RG, Mietus JE, Moody GB, Peng CK, Stanley HE. PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals. Circulation. 2000 Jun 13;101(23):e215–e220.

[4] Pan J, Tompkins WJ. A real-time QRS detection algorithm. IEEE Trans Biomed Eng. 1985 Mar;32(3):230–236. doi:10.1109/TBME.1985.325532.

[5] Julien C. The enigma of Mayer waves: facts and models. Cardiovasc Res. 2006 Apr 1;70(1):12–21. doi:10.1016/j.cardiores.2005.11.008.