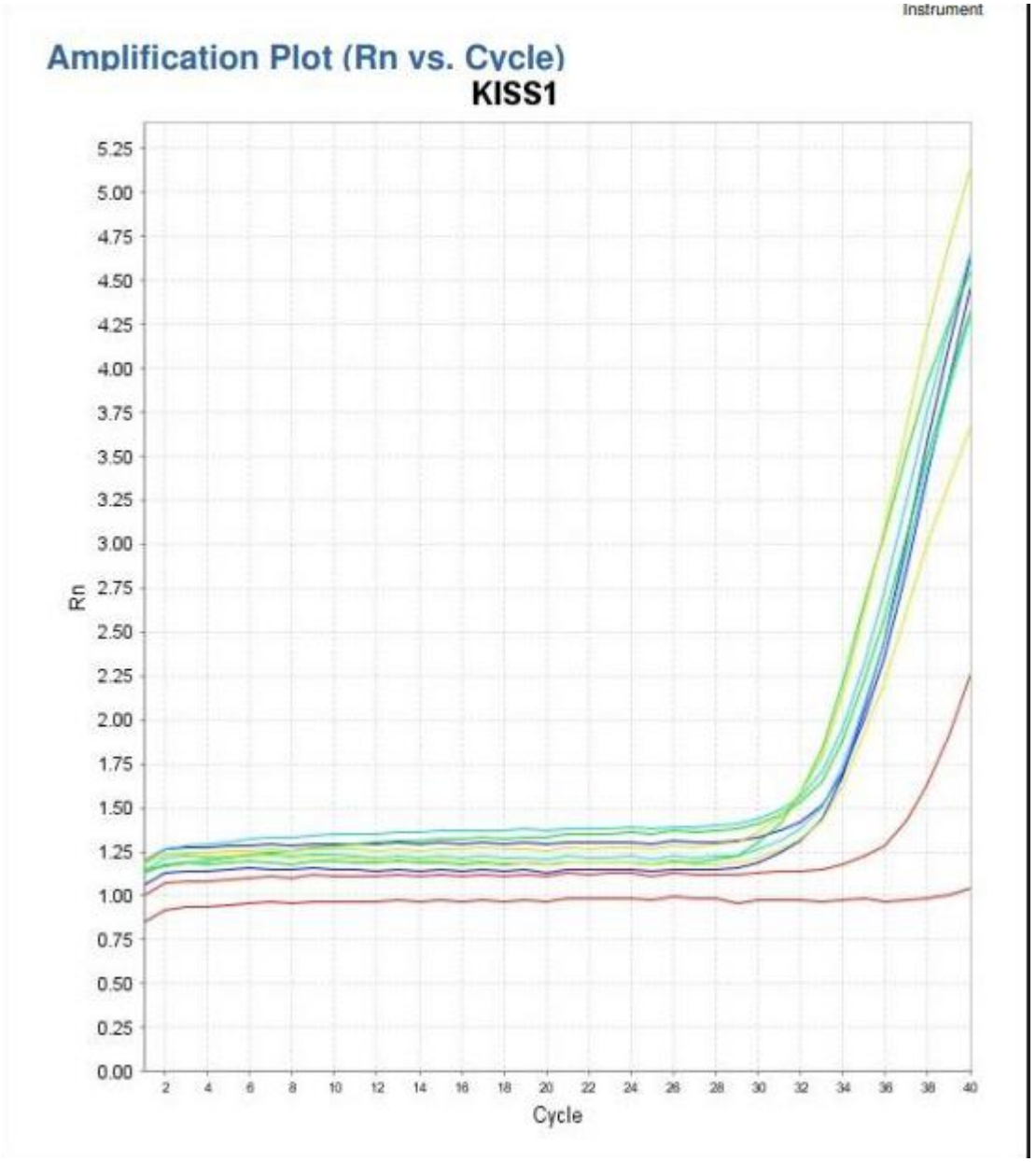
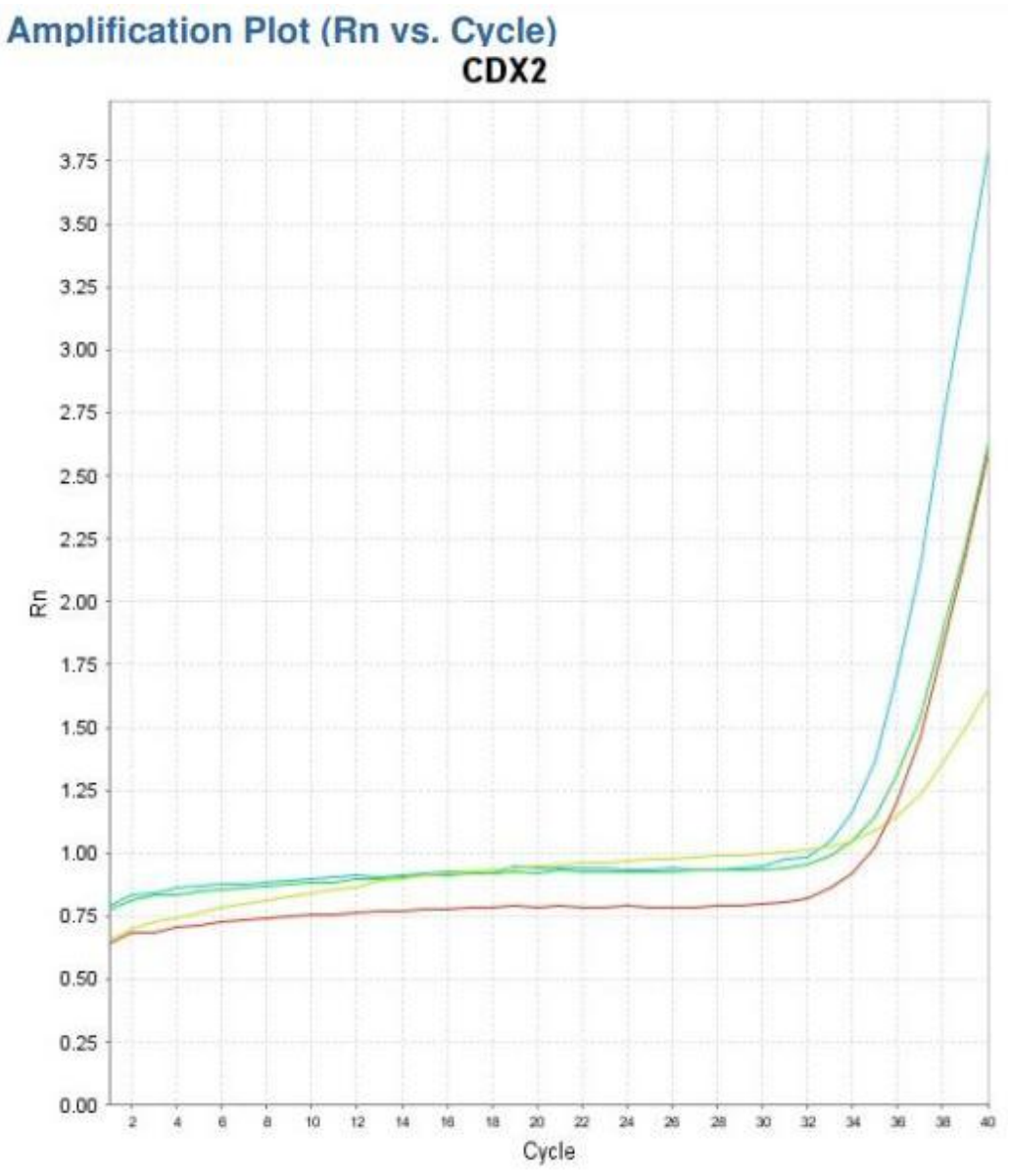
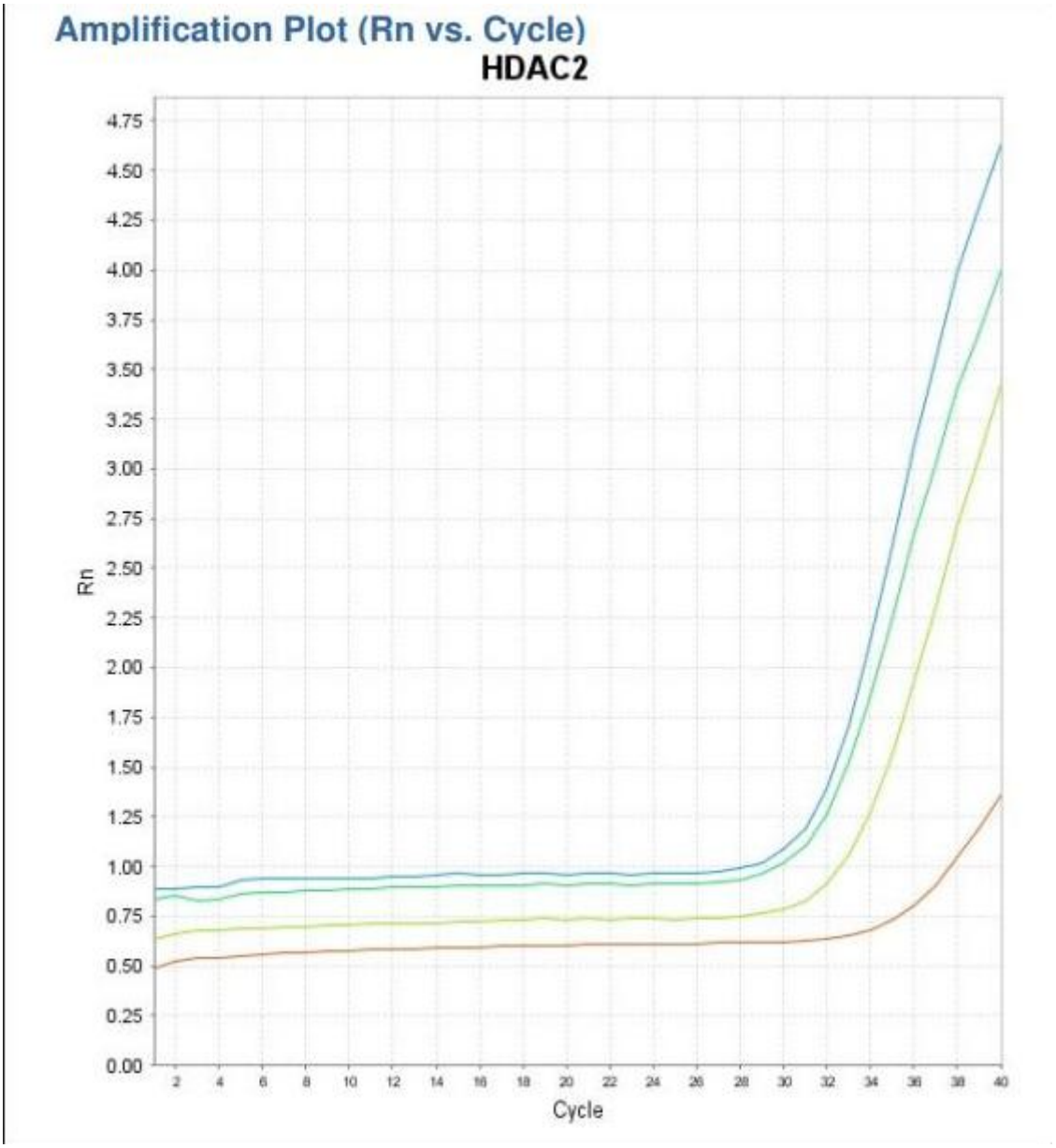
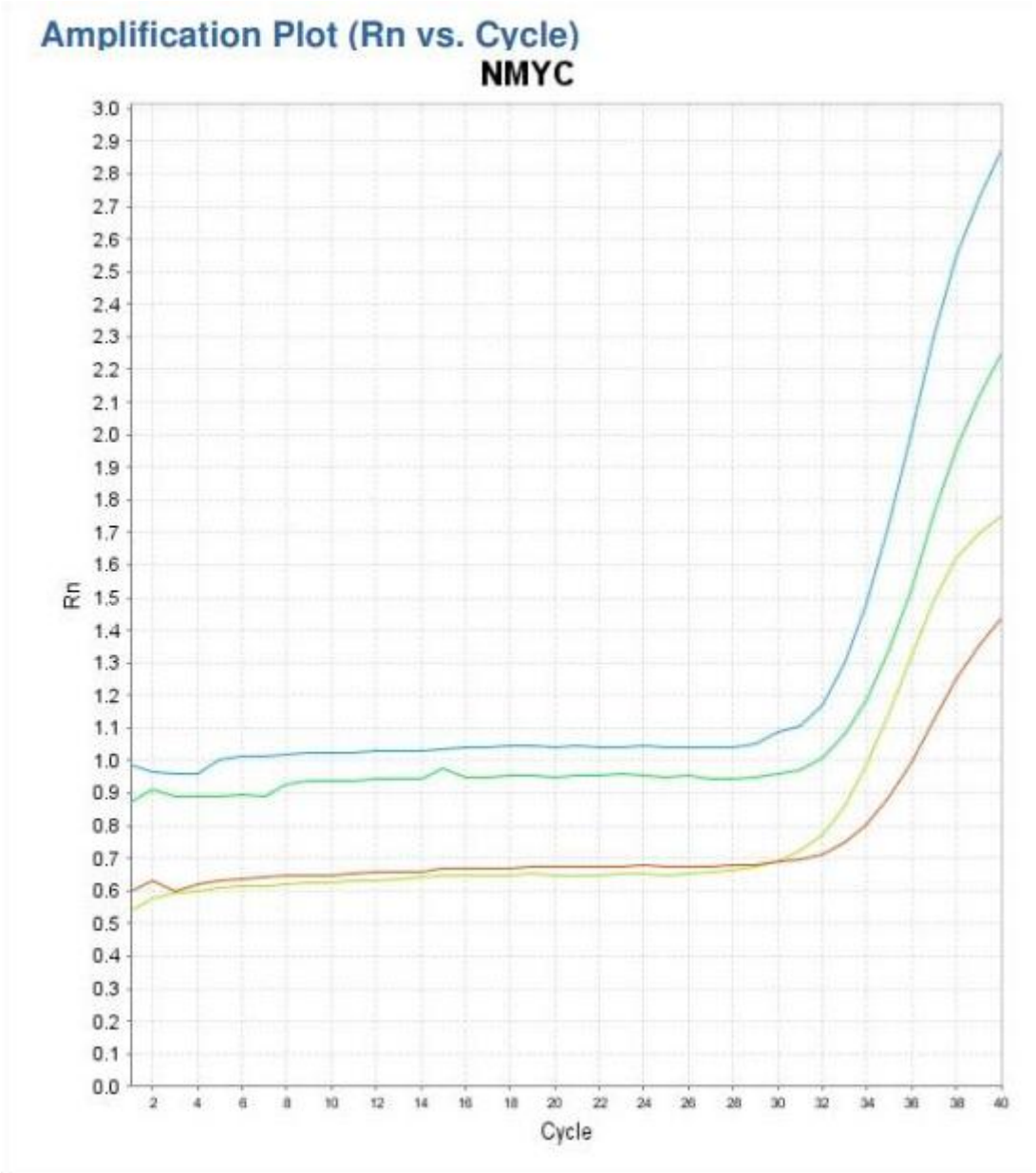
APPENDIX-1

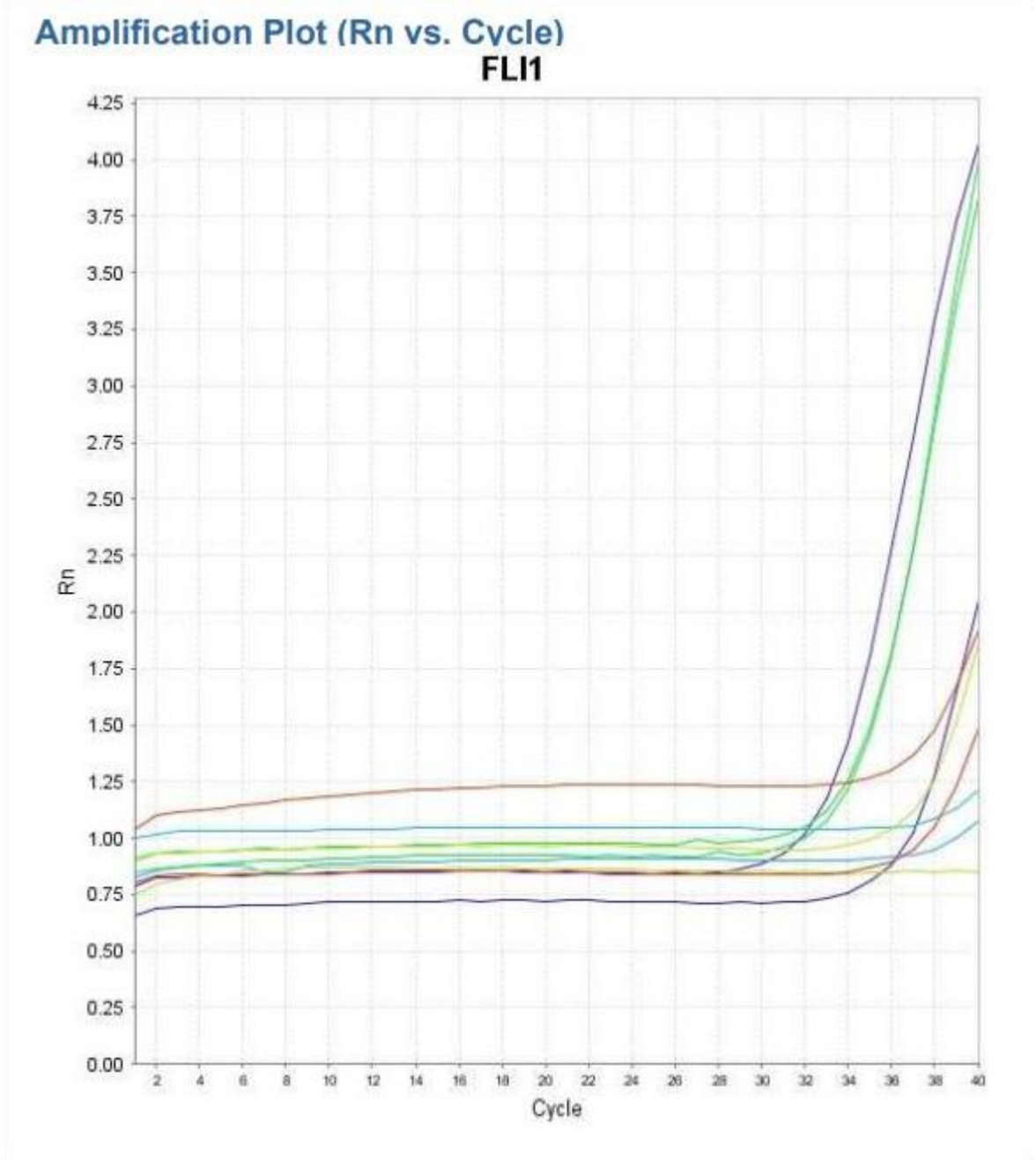
Amplification Curve for qRTPCR

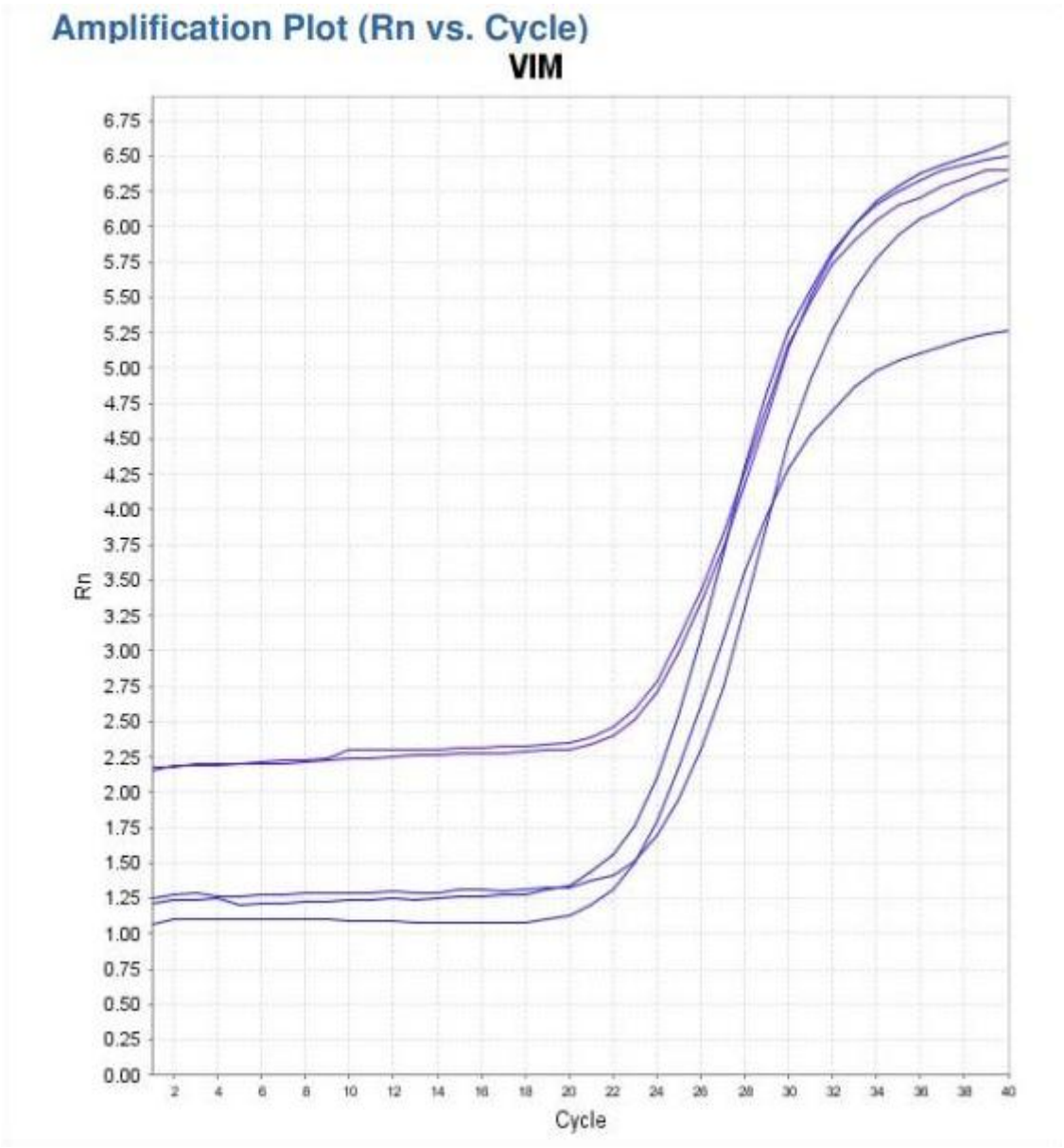


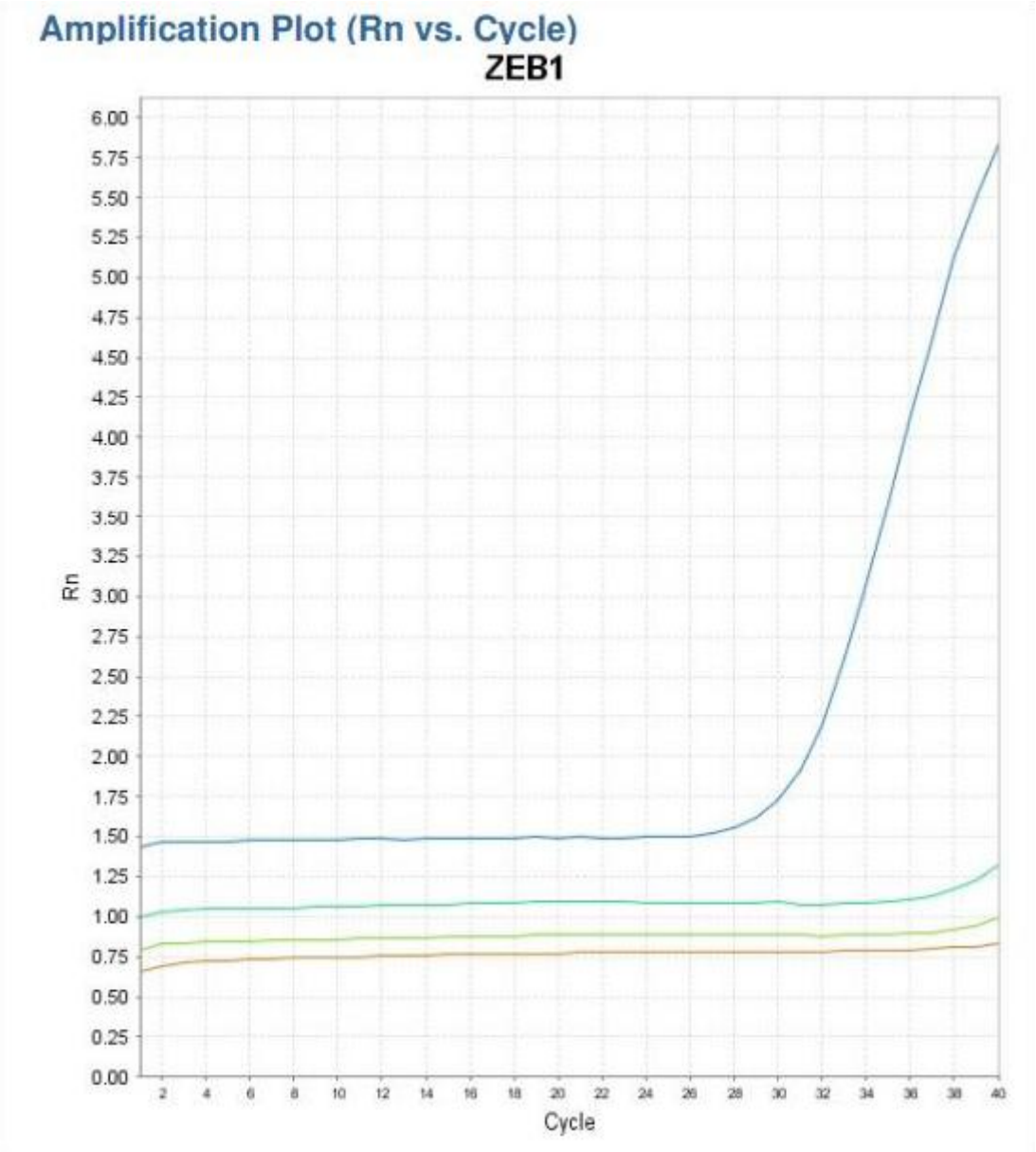

Amplification Plot (Rn vs. Cycle)
CDX2
Rn
Cycle
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

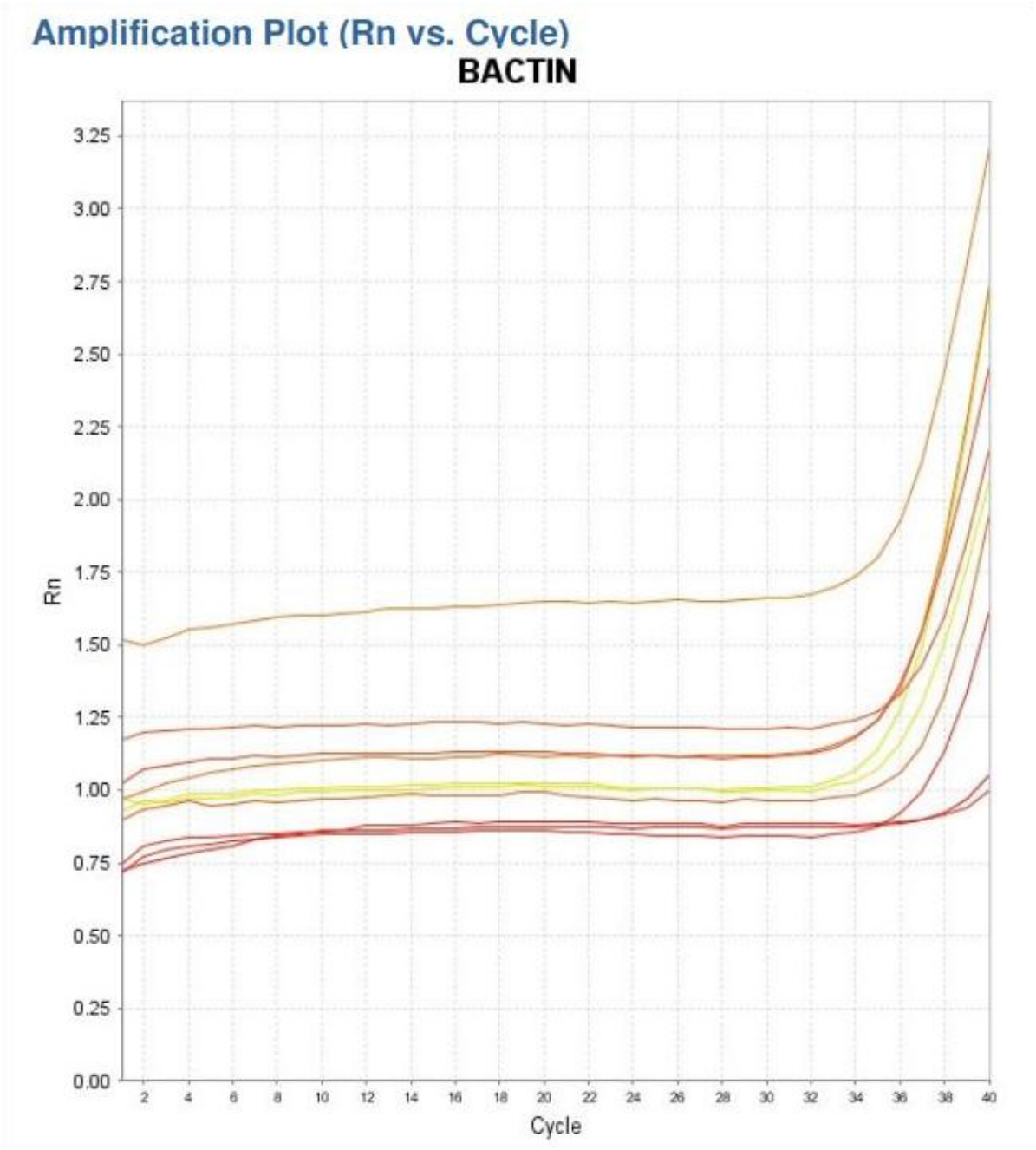

Amplification Plot (Rn vs. Cycle)
HDAC2
Rn
Cycle
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

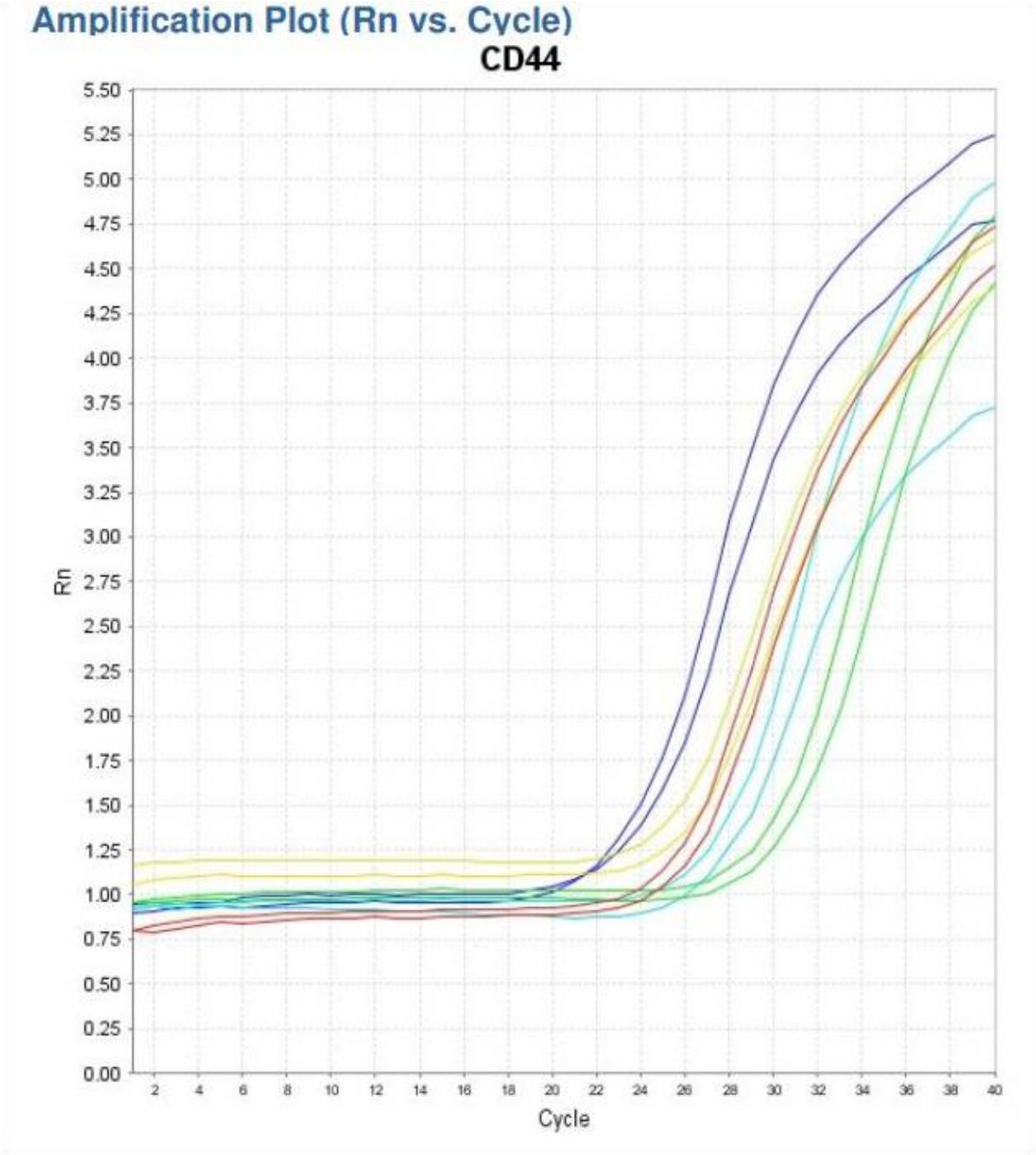

Amplification Plot (Rn vs. Cycle)
NMYC
Rn
Cycle
3.0
2.9
2.8
2.7
2.6
2.5
2.4
2.3
2.2
2.1
2.0
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

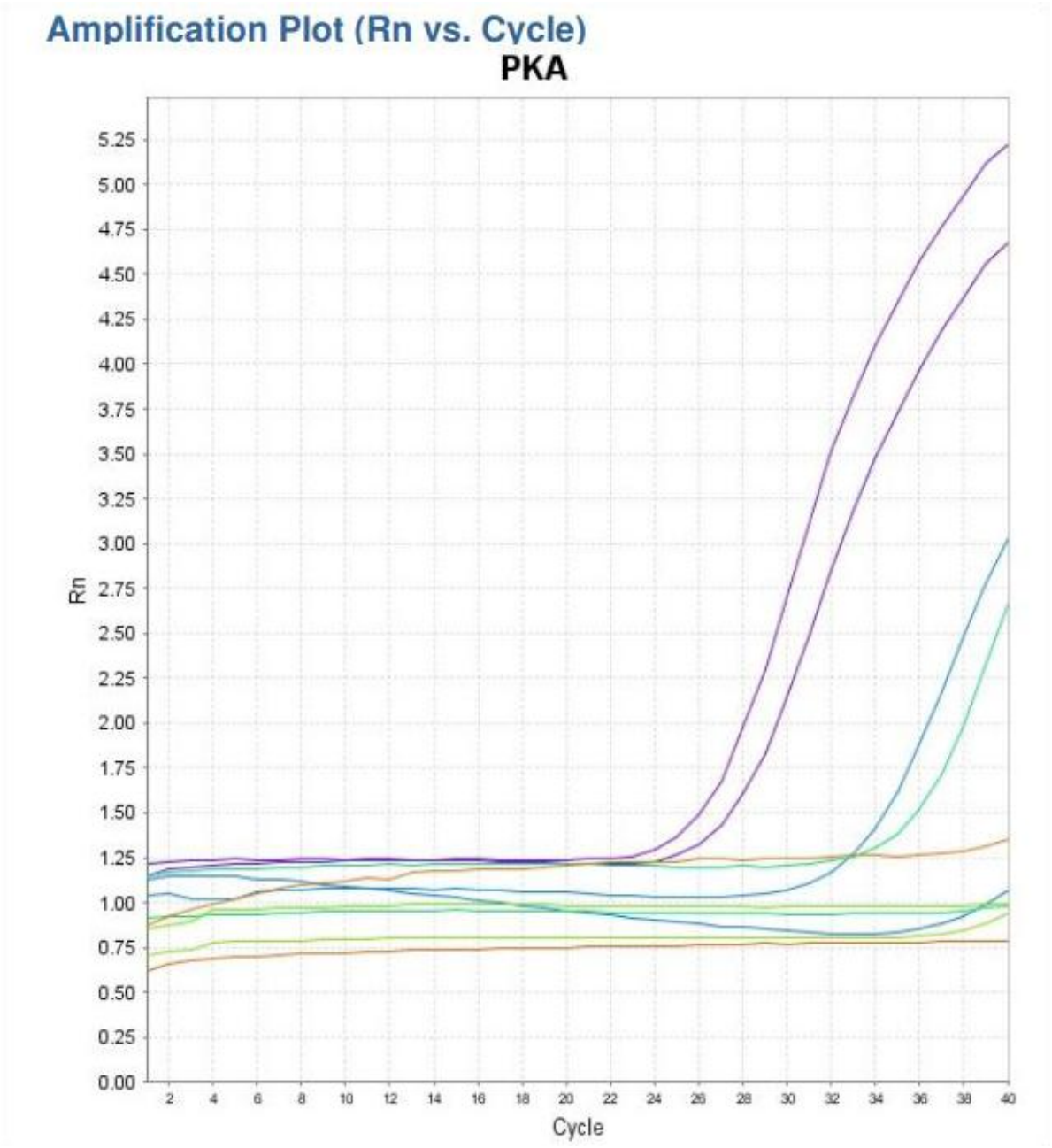

Amplification Plot (Rn vs. Cycle)
FLI1
Rn
Cycle
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

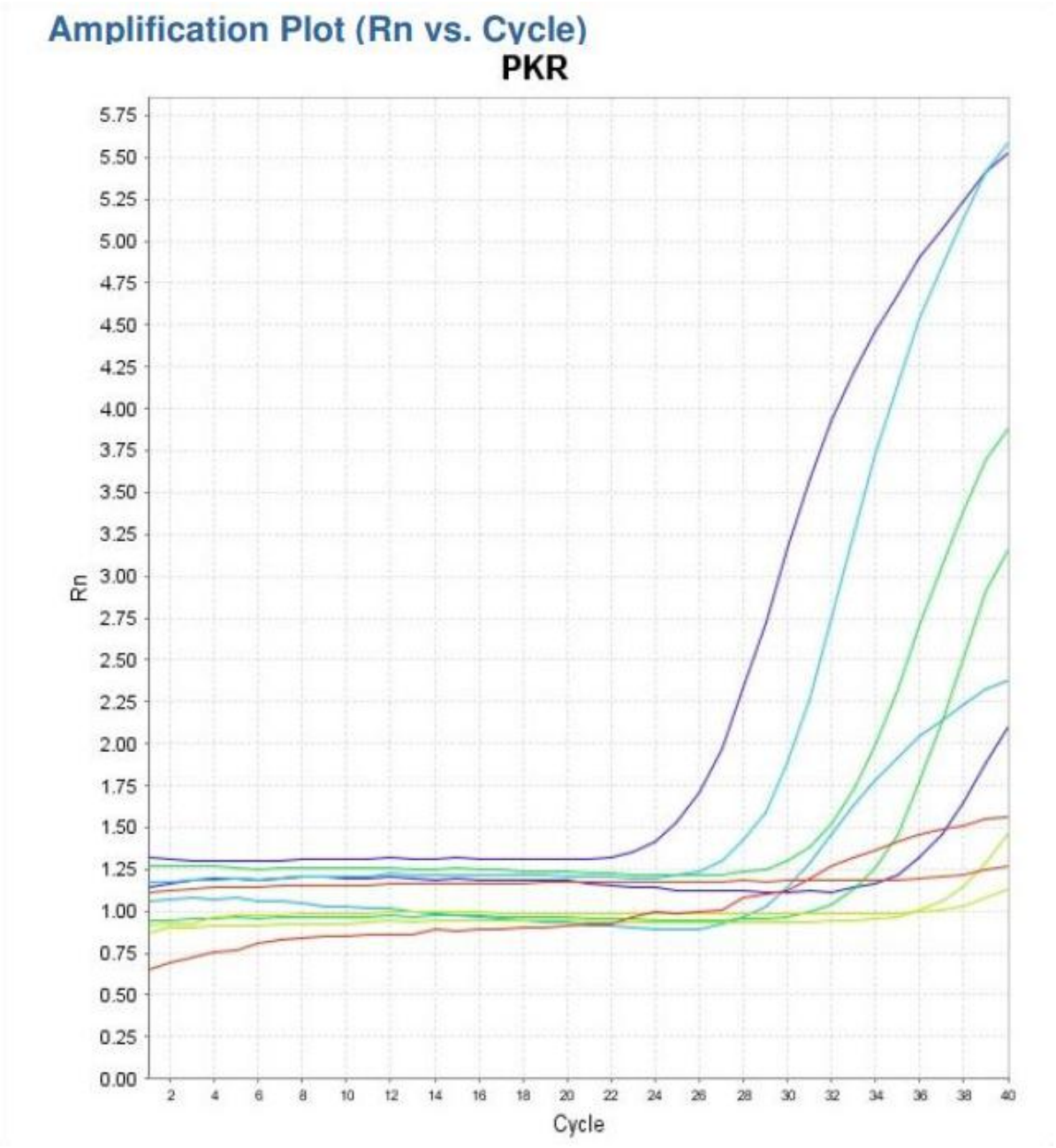

Amplification Plot (Rn vs. Cycle)
VIM
Rn
Cycle
6.75
6.50
6.25
6.00
5.75
5.50
5.25
5.00
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

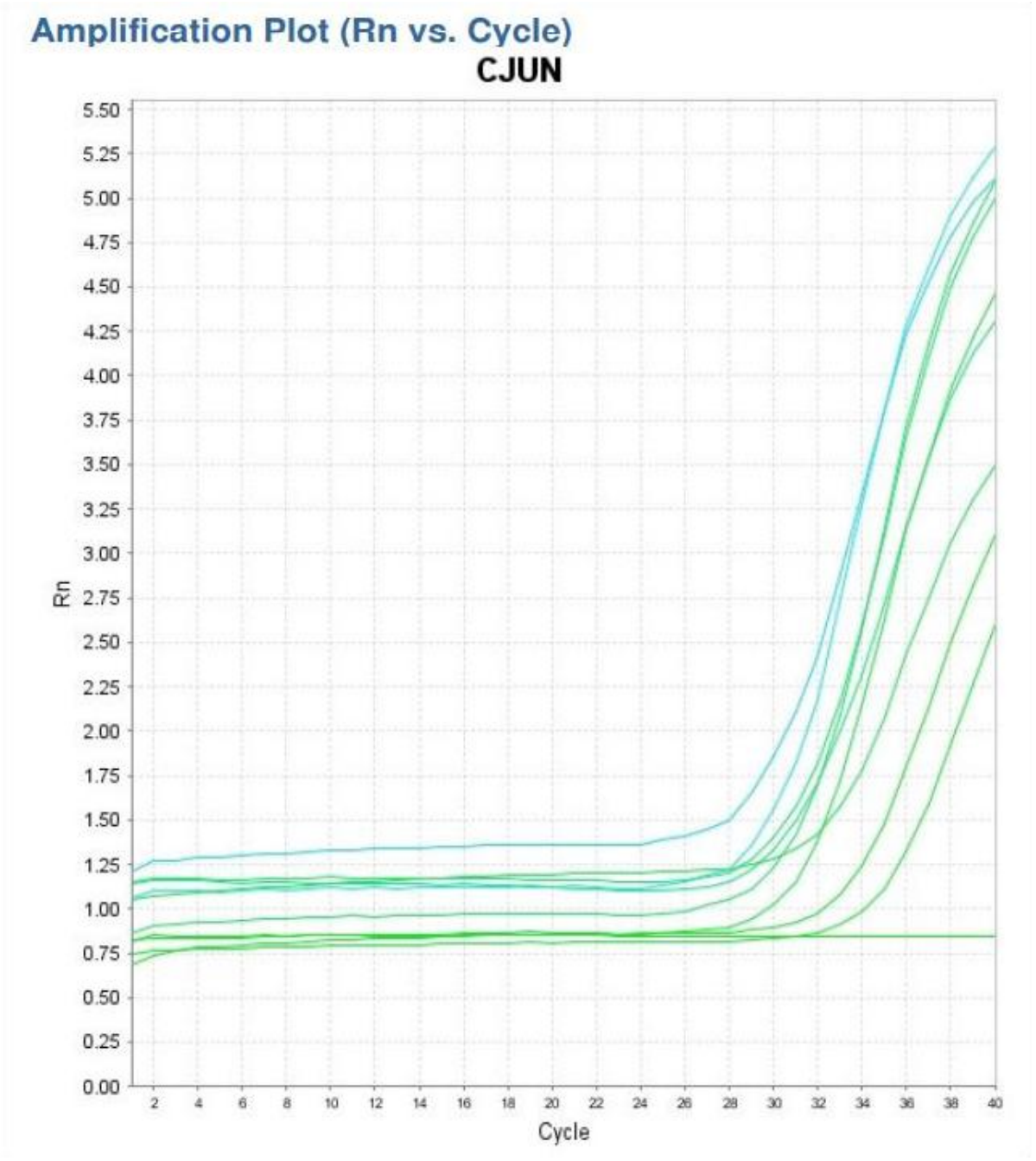

Amplification Plot (Rn vs. Cycle)
ZEB1
Rn
Cycle
6.00
5.75
5.50
5.25
5.00
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

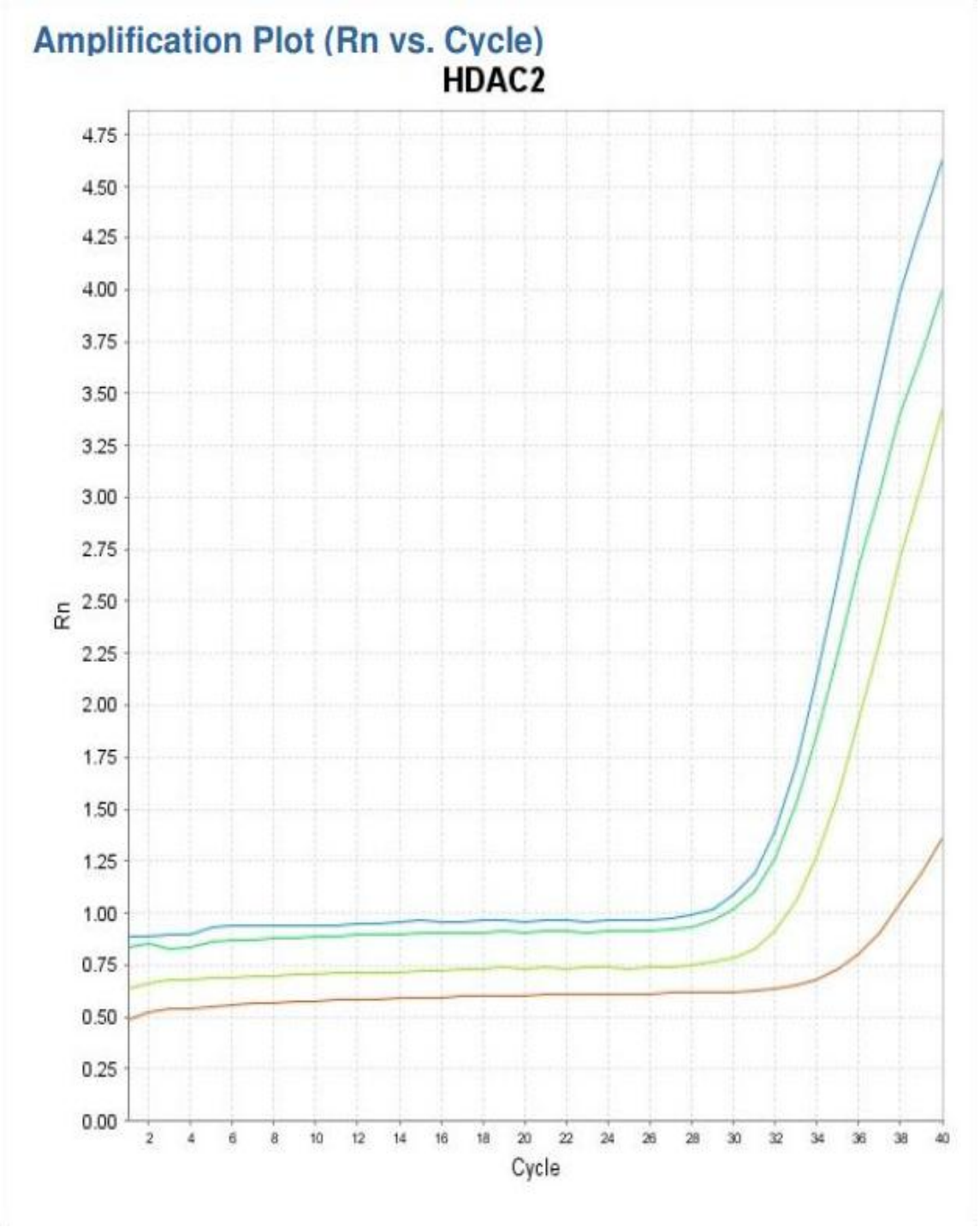

Amplification Plot (Rn vs. Cycle)
BACTIN
Rn
Cycle
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

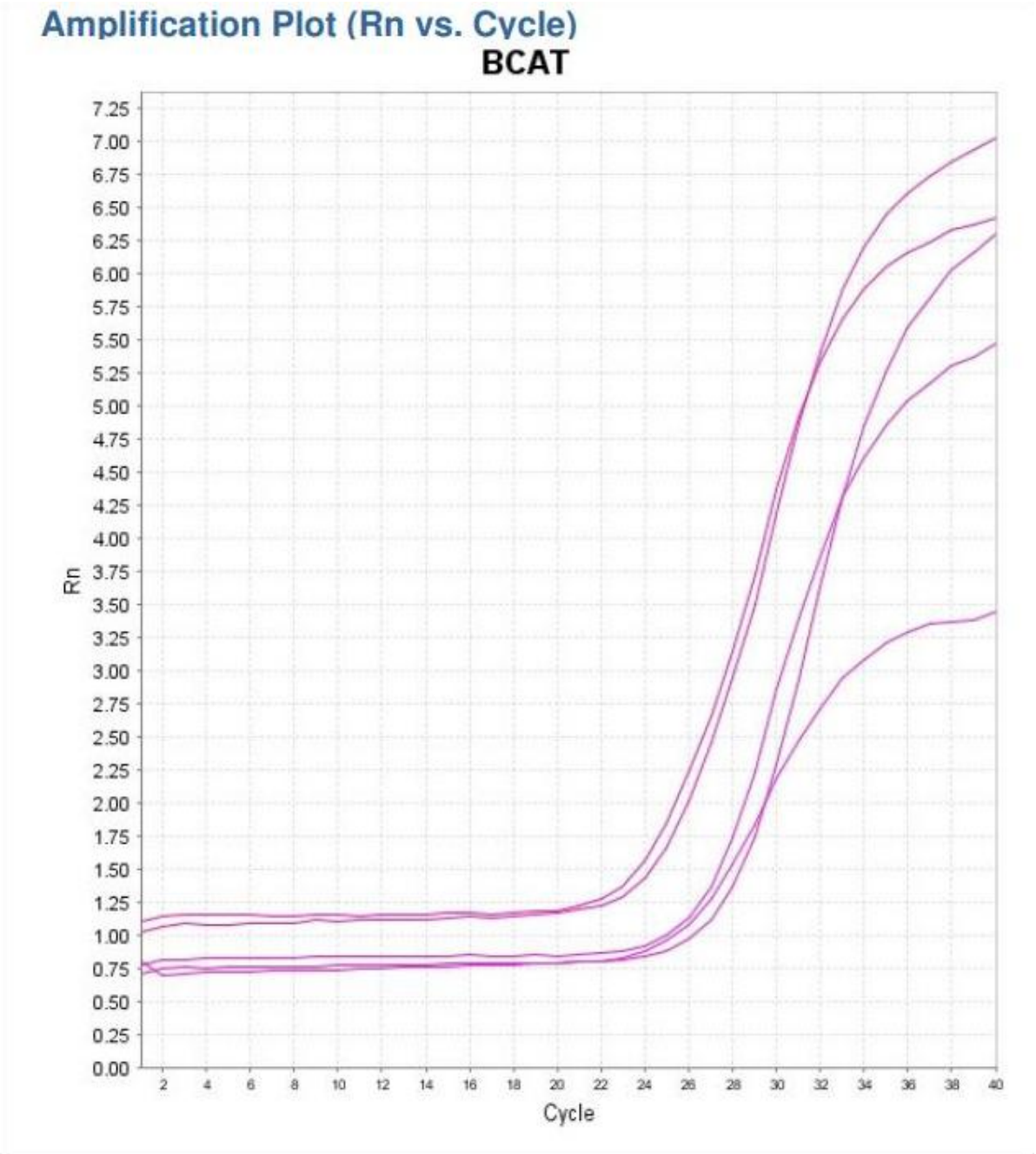

Amplification Plot (Rn vs. Cycle)
CD44
Rn
Cycle
5.50
5.25
5.00
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

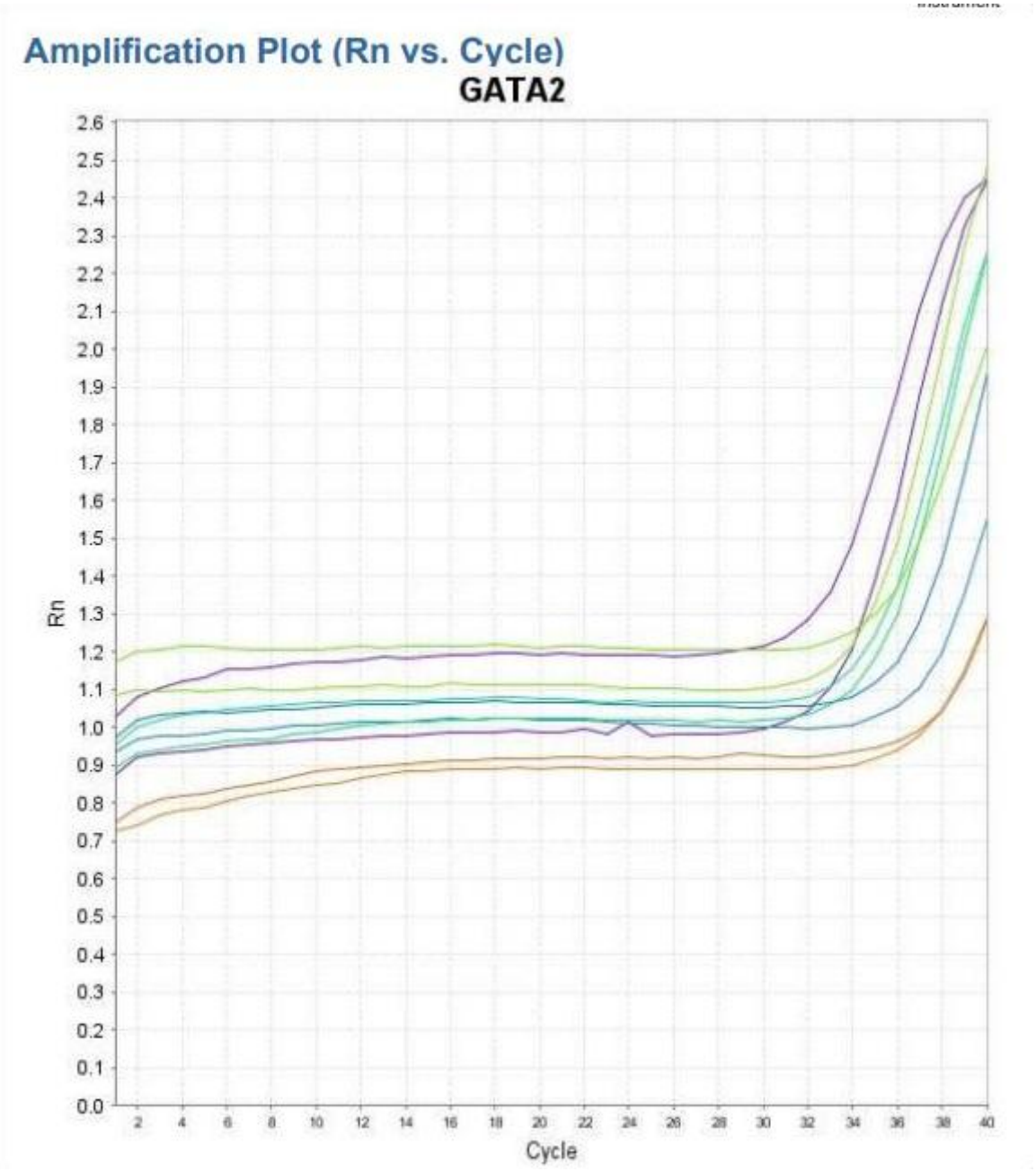

Amplification Plot (Rn vs. Cycle)
PKA
Rn
Cycle
5.25
5.00
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

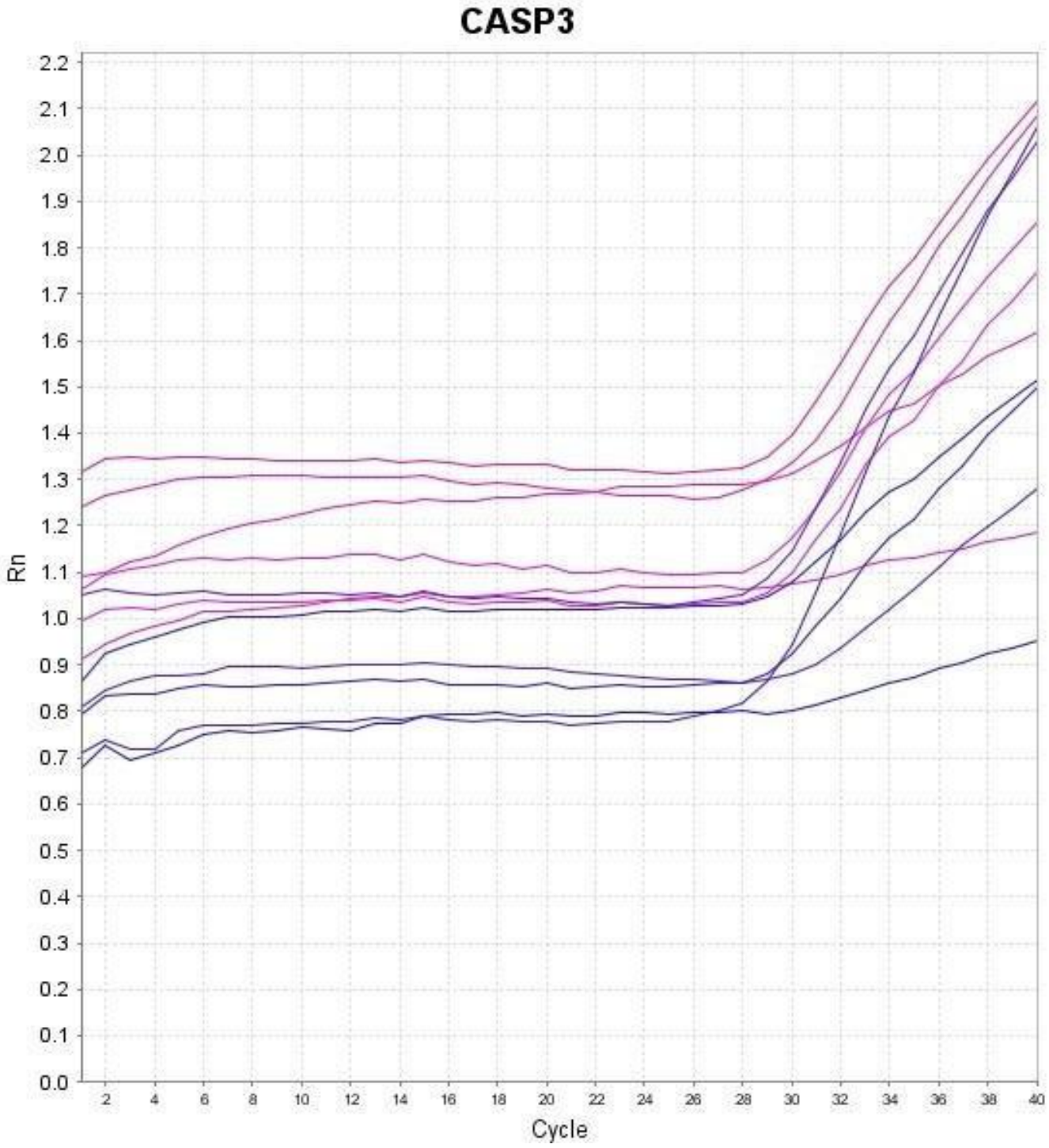

Amplification Plot (Rn vs. Cycle)
PKR
Rn
Cycle
5.75
5.50
5.25
5.00
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

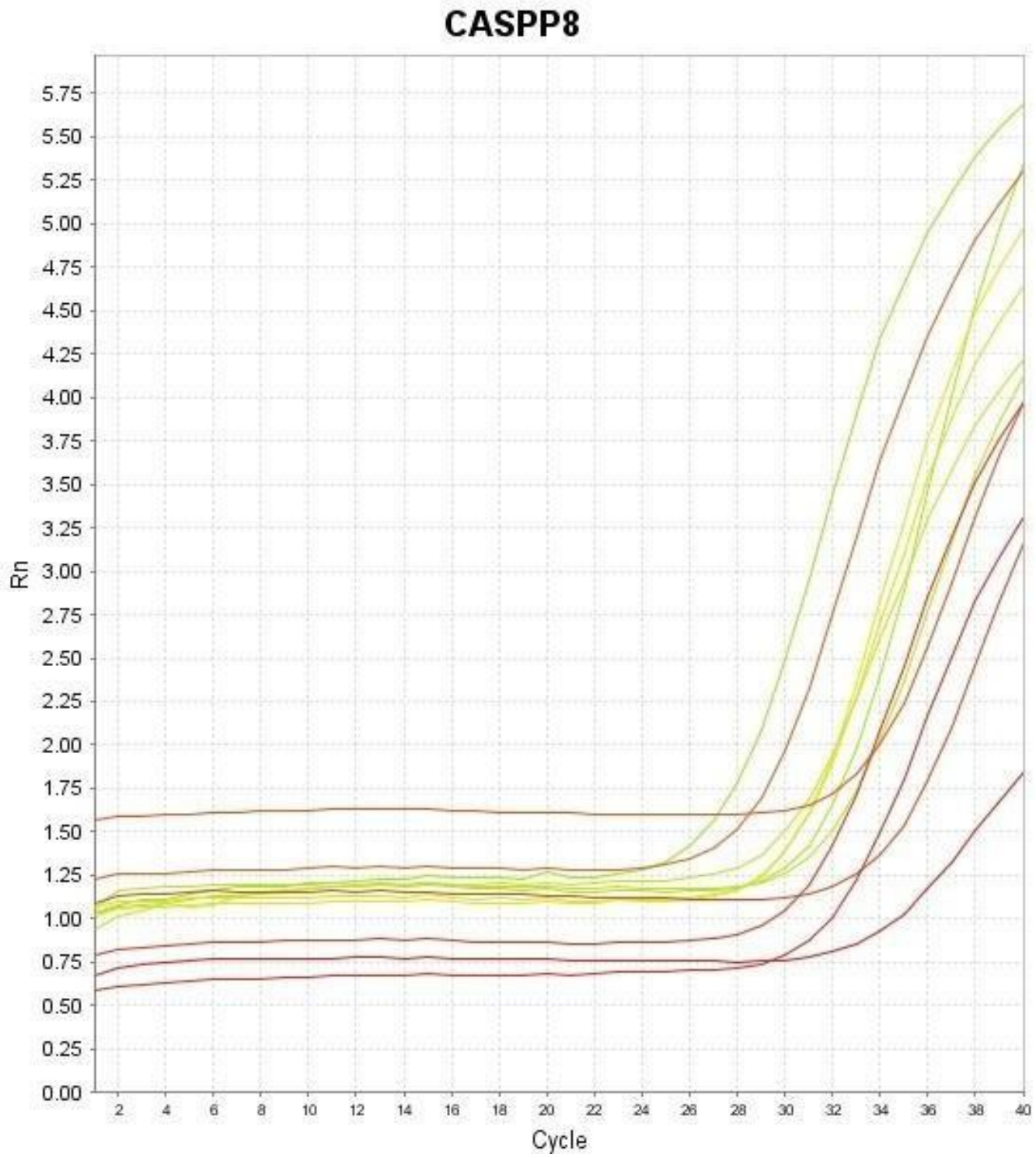

Amplification Plot (Rn vs. Cycle)
CJUN
Rn
Cycle
5.50
5.25
5.00
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

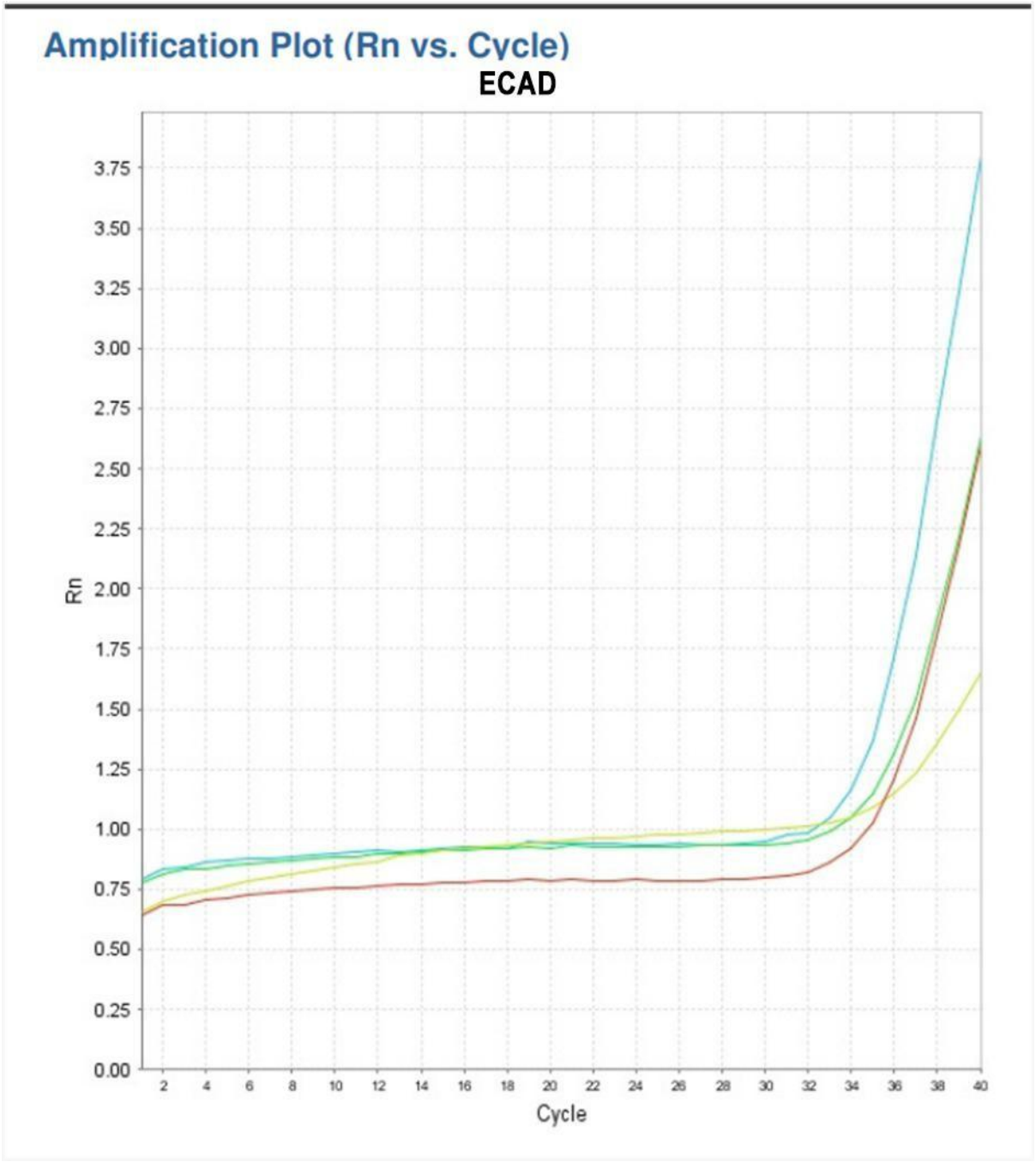

Amplification Plot (Rn vs. Cycle)
HDAC2
Rn
Cycle
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

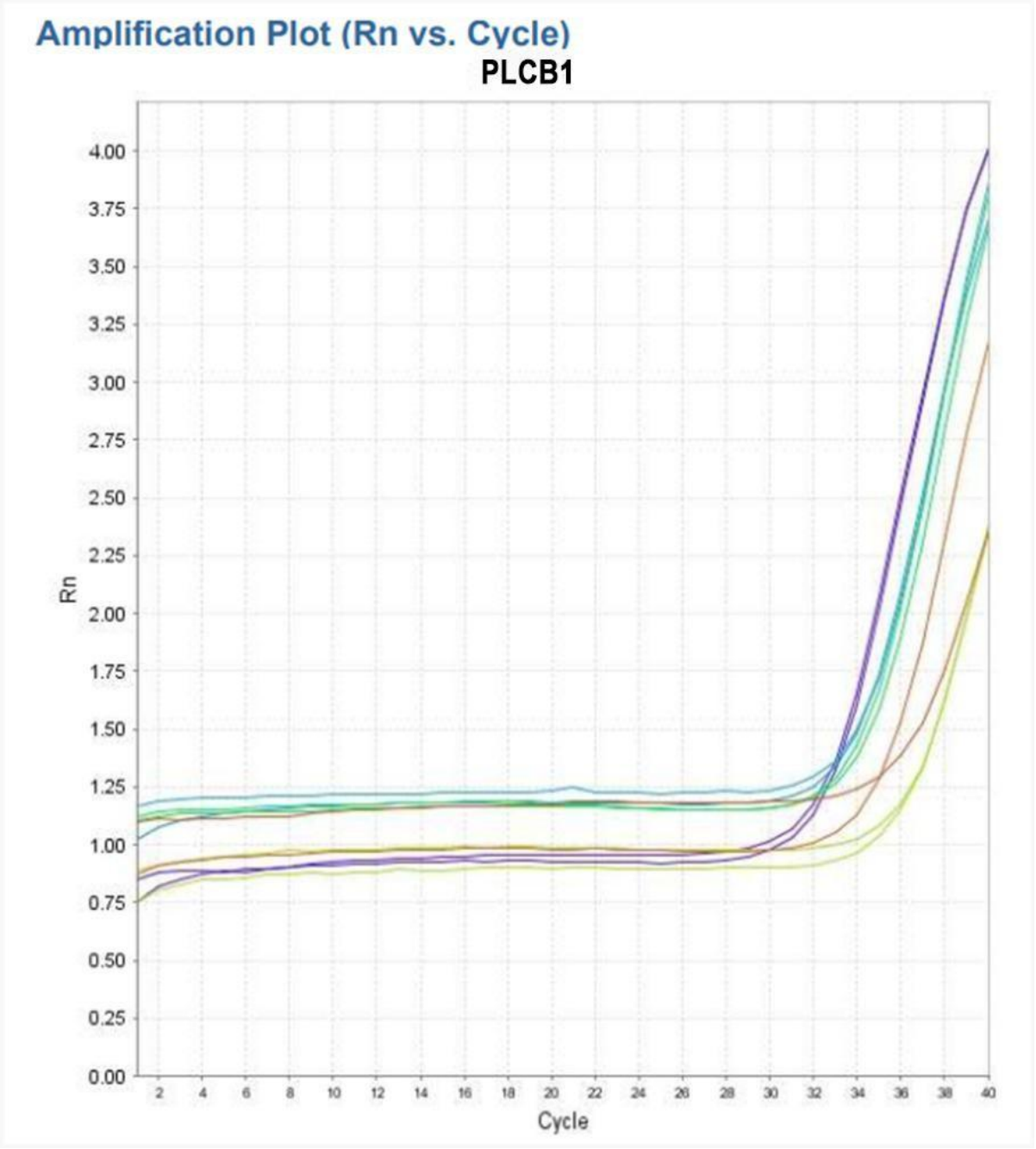

Amplification Plot (Rn vs. Cycle)
BCAT
Rn
Cycle
7.25
7.00
6.75
6.50
6.25
6.00
5.75
5.50
5.25
5.00
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40

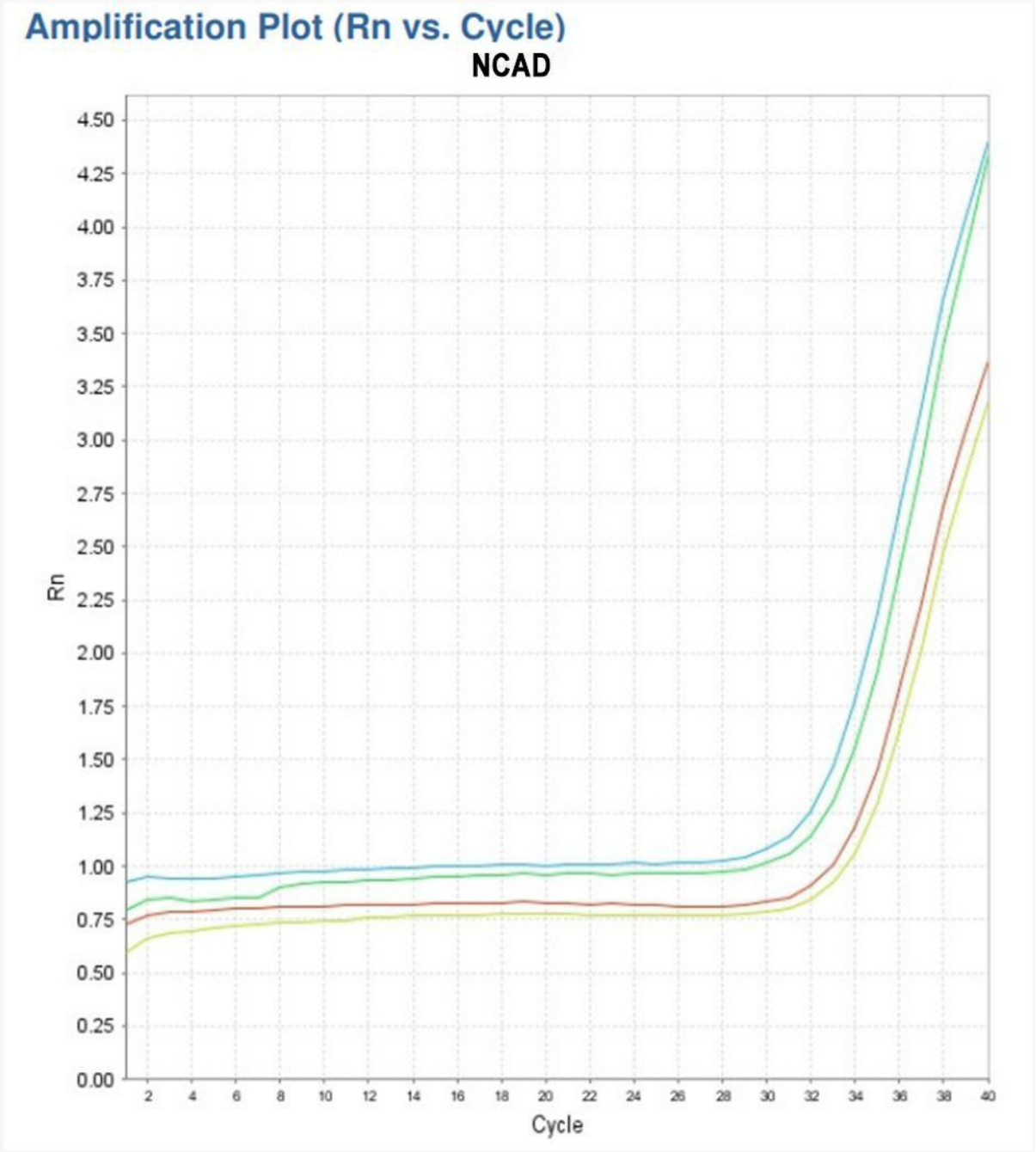

Amplification Plot (Rn vs. Cycle)
GATA2
Rn
Cycle
2.6
2.5
2.4
2.3
2.2
2.1
2.0
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

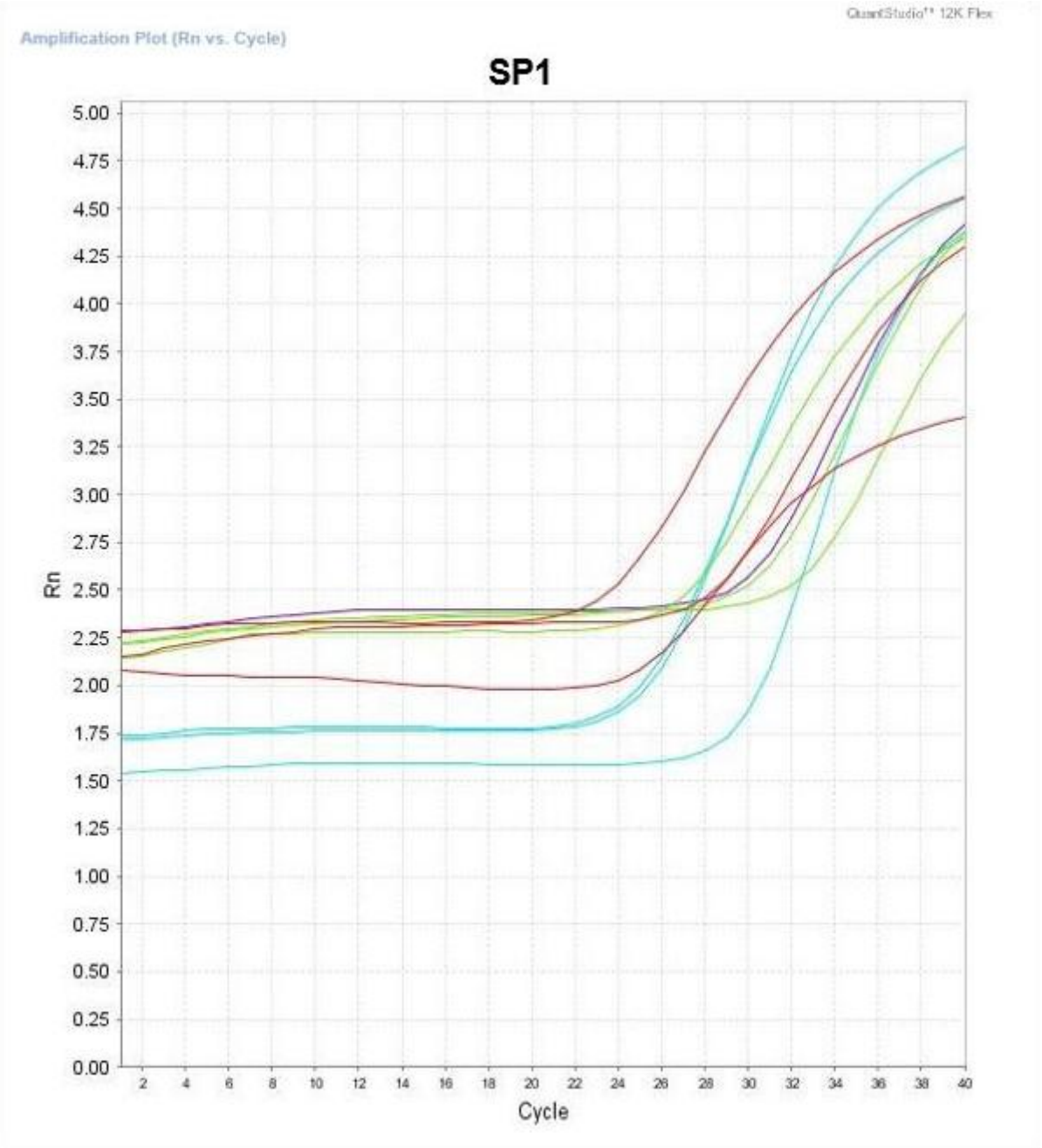

CASP3
Rn
Cycle
2.2
2.1
2.0
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

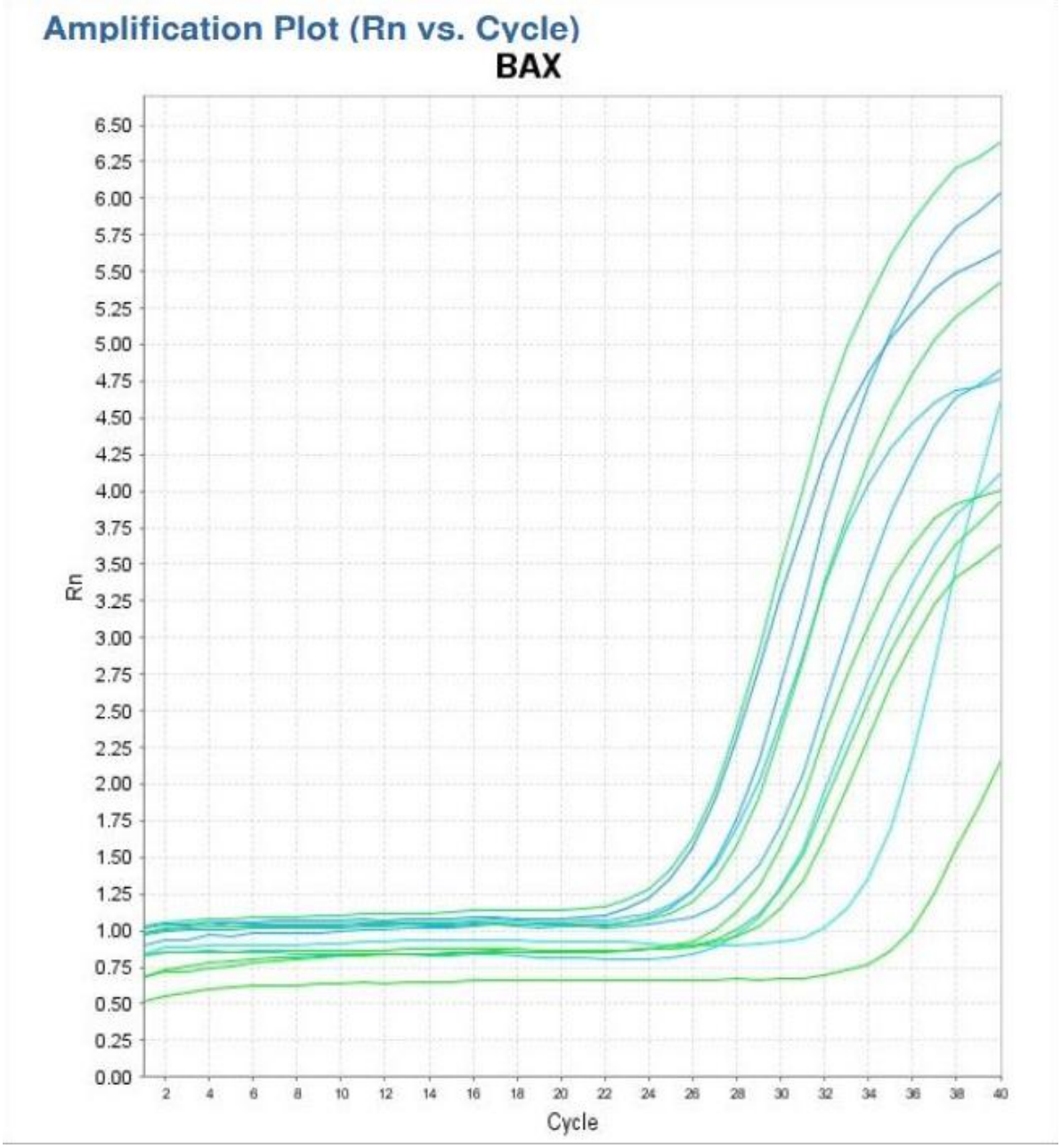

CASPP8
Rn
Cycle
5.75
5.50
5.25
5.00
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

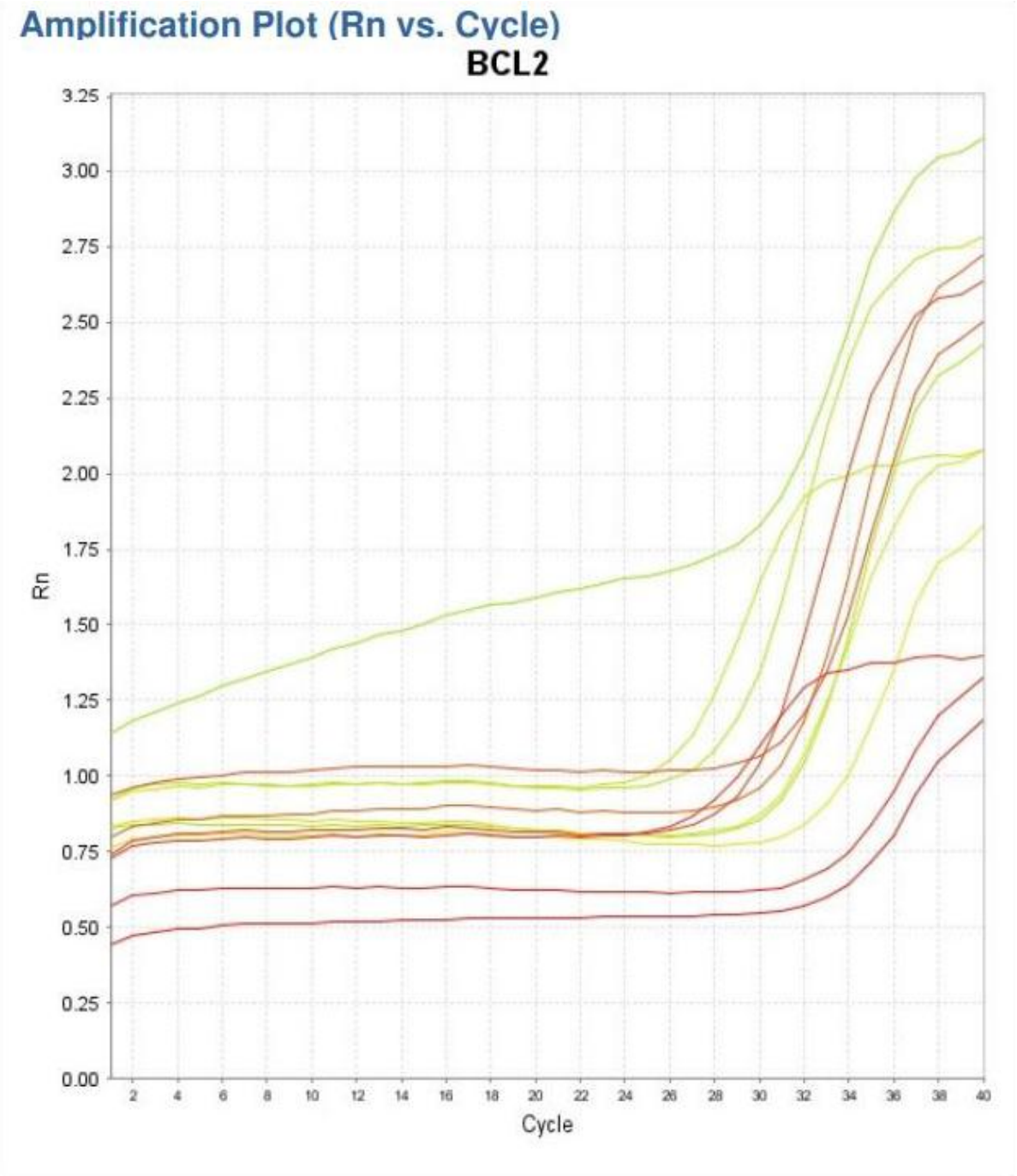

Amplification Plot (Rn vs. Cycle)
ECAD
Rn
Cycle
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

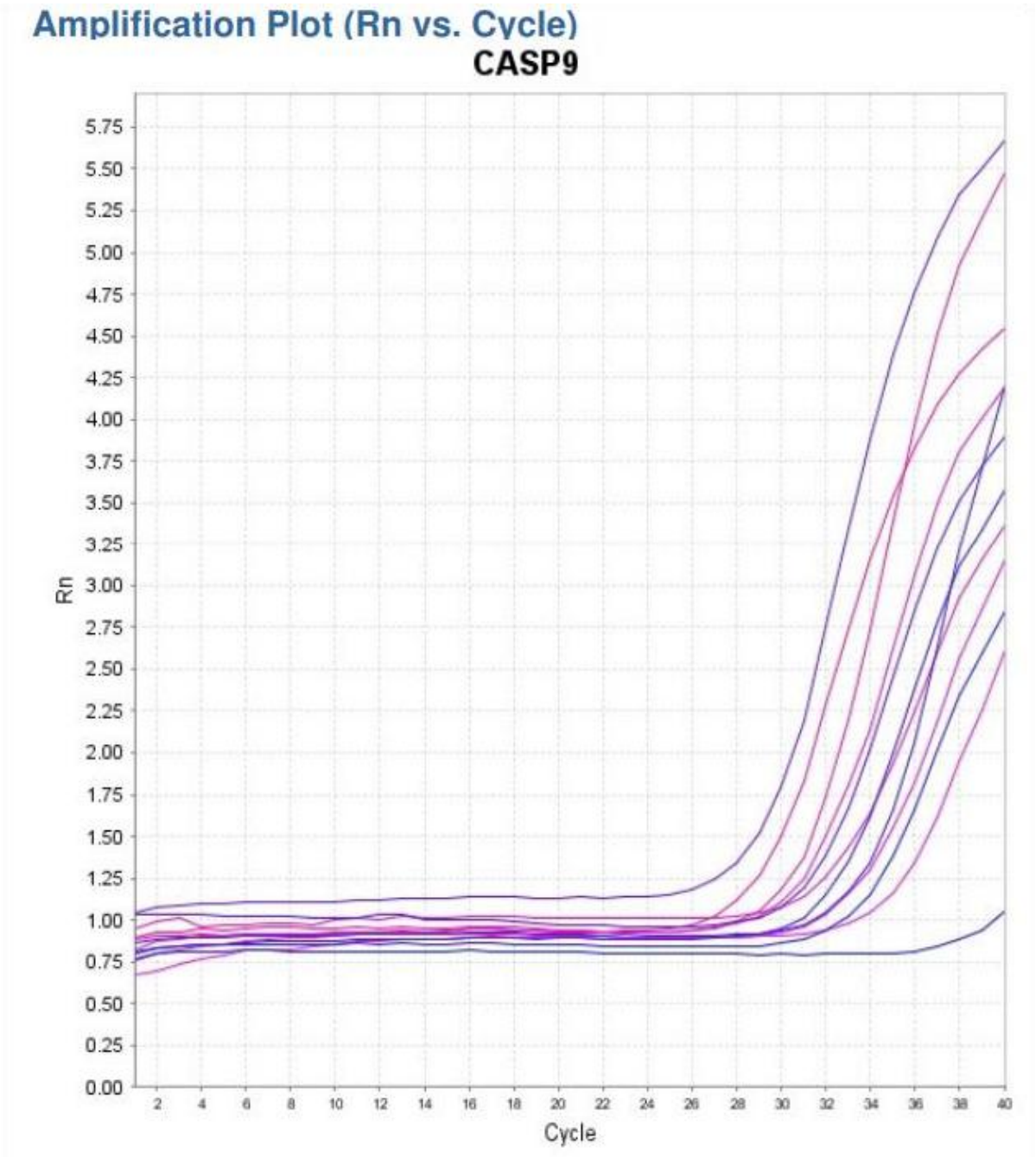

Amplification Plot (Rn vs. Cycle)
PLCB1
Rn
Cycle
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

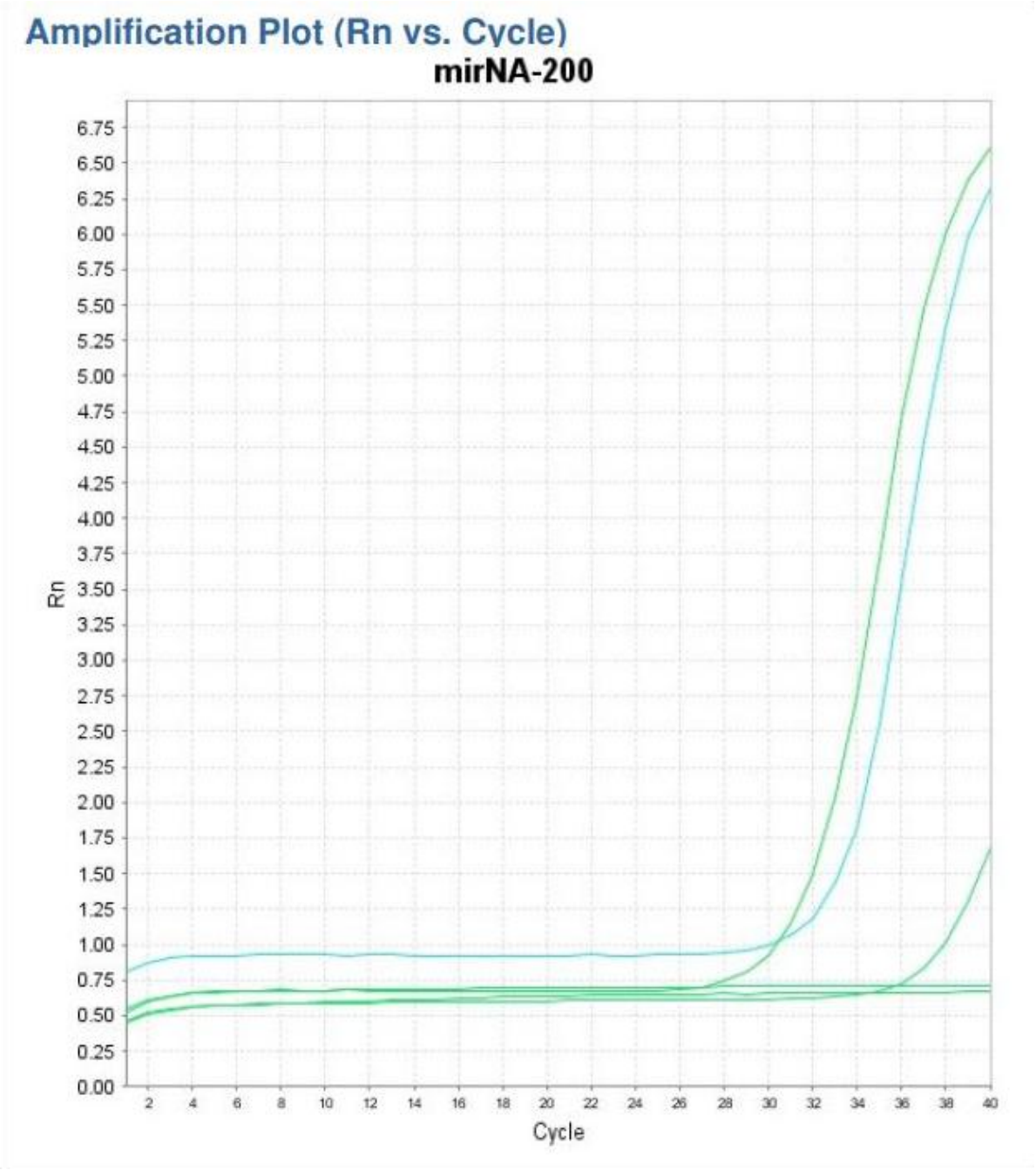

Amplification Plot (Rn vs. Cycle)
NCAD
Rn
Cycle
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

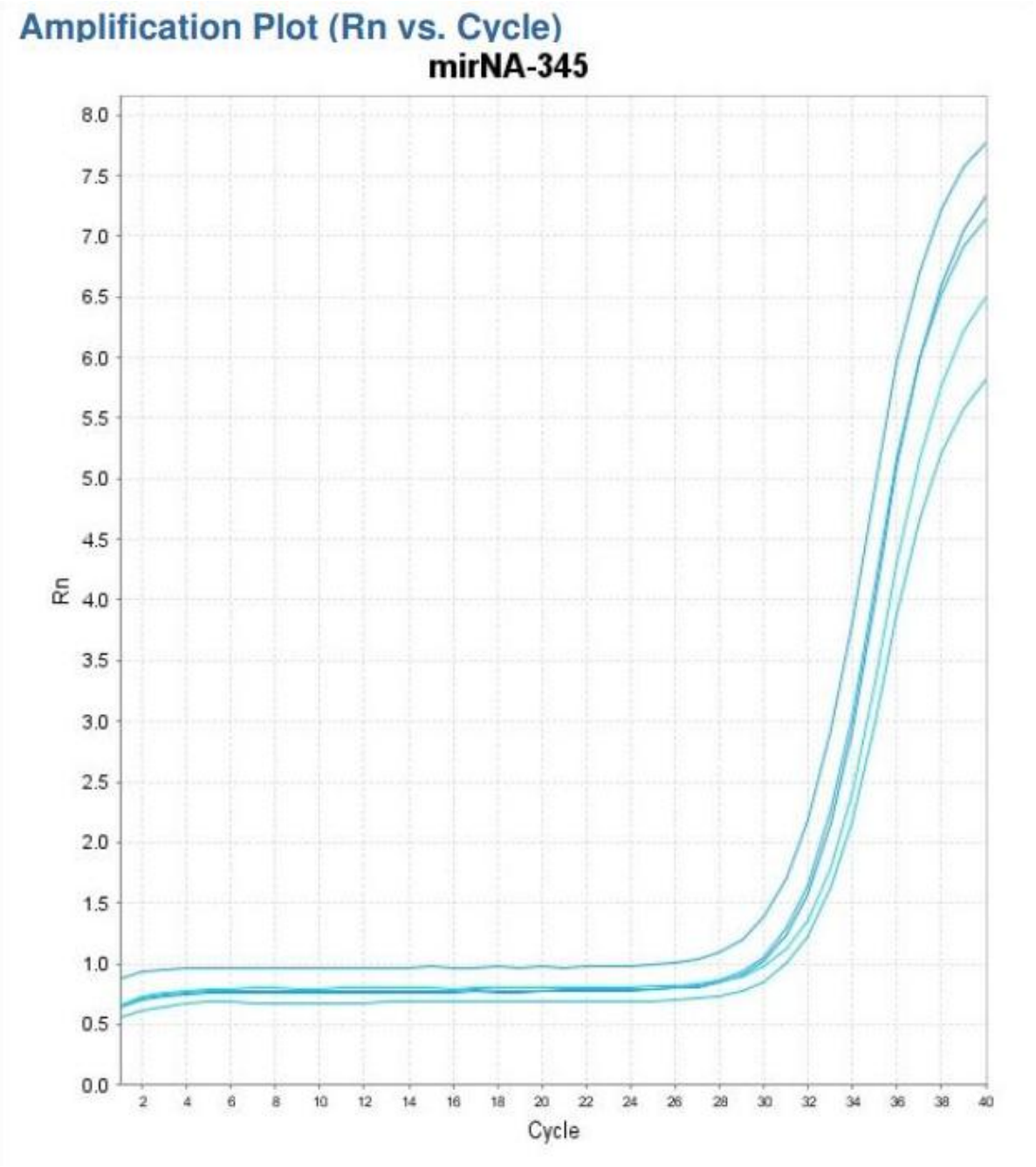

QuantStudio™ 12K Flex
Amplification Plot (Rn vs. Cycle)
SP1
Rn
Cycle
5.00
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

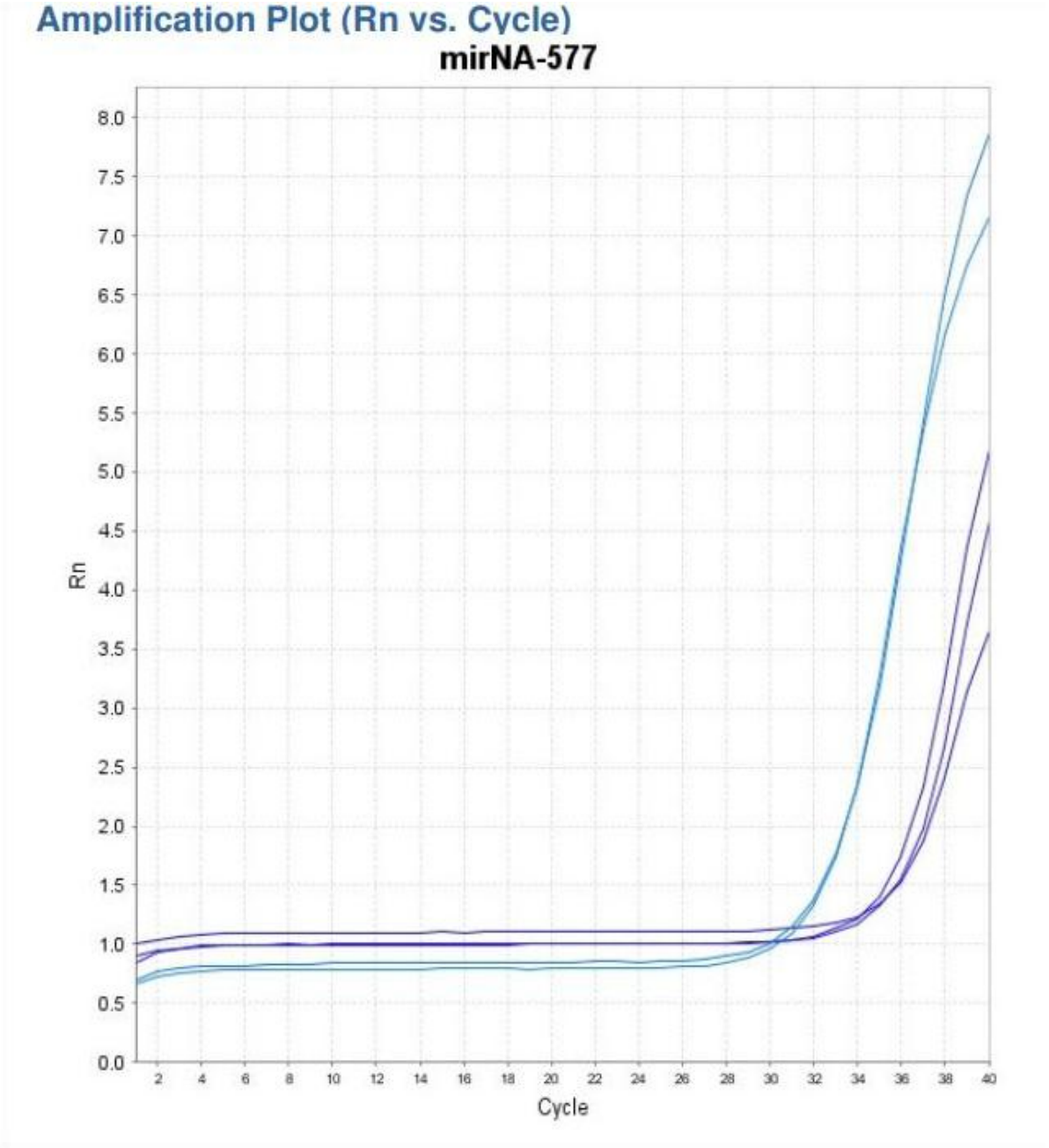

Amplification Plot (Rn vs. Cycle)
BAX
Rn
Cycle
6.50
6.25
6.00
5.75
5.50
5.25
5.00
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40


Amplification Plot (Rn vs. Cycle)
BCL2
Rn
Cycle
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40


Amplification Plot (Rn vs. Cycle)
CASP9
Rn
Cycle
5.75
5.50
5.25
5.00
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40


Amplification Plot (Rn vs. Cycle)
mirNA-200
Rn
Cycle
6.75
6.50
6.25
6.00
5.75
5.50
5.25
5.00
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40


Amplification Plot (Rn vs. Cycle)
mirNA-345
Rn
Cycle
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40


Amplification Plot (Rn vs. Cycle)
mirNA-577
Rn
Cycle
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

APPENDIX-2:

1. PRO-Dy script for Molecular Dynamics Simulations:

A) Radius of Gyration:

```
from prody import *
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Parse the PDB structure
structure = parsePDB('test.pdb')
if structure is None:
    raise ValueError("PDB structure could not be loaded.")
print(f"Structure loaded with {structure.numAtoms()} atoms.")

# Step 2: Parse the trajectory
trajectory = Trajectory('trajectoryfile.dcd')  # Replace with your DCD file
trajectory.link(structure)
print(f"Trajectory loaded with {trajectory.numFrames()} frames and {structure.numAtoms()} atoms.")

# Step 3: Select atoms for Rg calculations
protein_atoms = structure.select('protein')  # Select the protein
dna_atoms = structure.select('nucleic')  # Select only the DNA
complex_atoms = structure.select('protein or nucleic')  # The entire complex (protein + DNA)

# Print residue and atom counts for each selection
selections = {
    'Protein': protein_atoms,
    'DNA': dna_atoms,
    'Complex': complex_atoms
}

for name, atoms in selections.items():
    if atoms is None:
        raise ValueError(f"No atoms found for {name}.")
    num_residues = len(set(atoms.getResindices()))  # Count unique residue indices
    print(f"{name} selection contains {num_residues} residues and {atoms.numAtoms()} atoms.")

# Step 4: Prepare Rg arrays for each component
rgyr_values = {
    'Protein': np.zeros(trajectory.numFrames()-1),
    'DNA': np.zeros(trajectory.numFrames()-1),
    'Complex': np.zeros(trajectory.numFrames()-1)
}

# Step 5: Loop through frames and calculate Rg for each component (ignoring frame 0)
for i, frame in enumerate(trajectory[1:]):
    coords = frame.getCoords()
    for name, atoms in selections.items():
        atom_coords = coords[atoms.getIndices()]
        rgyr_values[name][i] = calcGyradius(atom_coords)

print("Radius of gyration calculations completed for all components.")
```

```
# Step 6: Plot Rg values for Protein only
plt.figure(figsize=(10, 7))
plt.plot(rgyr_values['Protein'], color='blue', linewidth=2)
plt.xlabel('Frame Index')
plt.ylabel('Radius of Gyration (Å)')
plt.title('Radius of Gyration Over Trajectory (Protein Only)')
plt.grid(True)
plt.savefig('radius_of_gyration_protein_only.png')  # Save the protein-only Rg plot
print("Protein-only radius of gyration plot saved as 'radius_of_gyration_protein_only.png'.")
plt.show()

# Step 7: Plot the normal Rg for the entire complex alone
plt.figure(figsize=(10, 7))
plt.plot(rgyr_values['Complex'], color='red', linewidth=2)
plt.xlabel('Frame Index')
plt.ylabel('Radius of Gyration (Å)')
plt.title('Radius of Gyration Over Trajectory (Entire Complex)')
plt.grid(True)
plt.savefig('radius_of_gyration_complex_only.png')  # Save the normal Rg plot
print("Normal radius of gyration plot saved as 'radius_of_gyration_complex_only.png'.")
plt.show()

# Step 8: Save Rg values for each component to CSV files
for name in rgyr_values:
    csv_filename = f'{name.lower()}_rg.csv'
    np.savetxt(csv_filename, rgyr_values[name], delimiter=',', header='Frame Index,Radius of Gyration
(Å)', comments='')
    print(f"Radius of gyration values for {name} saved to '{csv_filename}'.")
```

B) PCA-FEL:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
import matplotlib.patches as mpatches

# Load PCA projections (PC1 & PC2)
pca_data = pd.read_csv('pca_projections.csv', header=0)  # Assuming PC1 is in first column, PC2 in
second
pc1 = pca_data.iloc[:, 0]  # Extract PC1
pc2 = pca_data.iloc[:, 1]  # Extract PC2

# Combine PC1 & PC2 into a single DataFrame
data = pd.concat([pc1, pc2], axis=1)
data.columns = ['PC1', 'PC2']

# Normalize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)

# Determine optimal k using the Elbow Method and Silhouette Score
```

```
inertia = []
sil_scores = []
k_range = range(2, 11)  # Checking k from 2 to 10

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(scaled_data)
    inertia.append(kmeans.inertia_)
    score = silhouette_score(scaled_data, kmeans.labels_)
    sil_scores.append(score)

# Plot Elbow Method
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(k_range, inertia, marker='o', linestyle='--')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')

# Plot Silhouette Score
plt.subplot(1, 2, 2)
plt.plot(k_range, sil_scores, marker='o', linestyle='--', color='red')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score for Optimal k')
plt.tight_layout()
plt.show()

# Suggest optimal k based on the highest Silhouette Score
suggested_k = k_range[np.argmax(sil_scores)]
print(f"Suggested optimal number of clusters based on Silhouette Score: {suggested_k}")

# Ask user to select optimal k
optimal_k = int(input("Enter the optimal number of clusters based on the graphs: "))

# Apply K-means clustering with optimal k
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
kmeans_labels = kmeans.fit_predict(scaled_data)

# Add cluster labels to the DataFrame
cluster_df = data.copy()
cluster_df['Cluster'] = kmeans_labels

# Calculate cluster centroids
centroids = kmeans.cluster_centers_
centroid_values = scaler.inverse_transform(centroids)  # Denormalize to original scale

# Find the frame closest to each cluster centroid
centroid_frames = []
for cluster_idx in range(optimal_k):
    cluster_points = data[cluster_df['Cluster'] == cluster_idx]

    # Convert normalized cluster points back to original scale before computing distances
    original_cluster_points = scaler.inverse_transform(cluster_points[['PC1', 'PC2']])
```

```
    # Compute distances using correctly scaled values
    distances = np.linalg.norm(original_cluster_points - centroid_values[cluster_idx], axis=1)

    # Find the closest frame to the centroid
    closest_frame = cluster_points.index[np.argmin(distances)]
    centroid_frames.append(closest_frame)

# Define distinct colors for clusters
distinct_colors = ['blue', 'green', 'red', 'purple', 'orange', 'brown', 'pink', 'gray', 'olive', 'cyan']  # Extend if
needed
cluster_colors = [distinct_colors[label] for label in kmeans_labels]  # Assign distinct color per cluster

# Plot the clustered data with centroids
plt.figure(figsize=(10, 7))

# Scatter plot using manually assigned distinct colors
scatter = plt.scatter(data['PC1'], data['PC2'], c=cluster_colors, alpha=0.7, s=50)

# Mark centroids with a black cross (without legend entry)
for idx, centroid in enumerate(centroid_values):
    centroid_pc1, centroid_pc2 = centroid
    plt.scatter(centroid_pc1, centroid_pc2, color='black', marker='x', s=150, linewidths=2)  # Black X
for centroids

# Create legend for clusters using distinct colors
legend_patches = [mpatches.Patch(color=distinct_colors[idx], label=f'Cluster {idx + 1}') for idx in
range(optimal_k)]
plt.legend(handles=legend_patches, title="Clusters", loc='upper left', bbox_to_anchor=(1, 1))

# Add titles and labels
plt.title(f"K-means Clustering on PC1 & PC2 (k={optimal_k})")
plt.xlabel("PC1 (Principal Component 1)")
plt.ylabel("PC2 (Principal Component 2)")
plt.grid(True)

# Show plot
plt.show()

# Save the clustered data to a CSV file
cluster_result = pd.DataFrame({
    'Cluster': cluster_df['Cluster'],
    'Frame': cluster_df.index,
    'PC1': cluster_df['PC1'],
    'PC2': cluster_df['PC2']
})

centroid_info = pd.DataFrame({
    'Cluster': range(optimal_k),
    'Centroid Frame': centroid_frames,
    'Centroid PC1': centroid_values[:, 0],
    'Centroid PC2': centroid_values[:, 1]
})

cluster_result.to_csv('k_clustering_pca_result.csv', index=False)
centroid_info.to_csv('centroid_pca_info.csv', index=False)
```

```
print("Clustering results saved as 'k_clustering_pca_result.csv'.")
print("Centroid details saved as 'centroid_pca_info.csv'.")
```

C) Script for Heatmap:

```
# Load required libraries
library(clusterProfiler)
library(enrichplot)
library(pheatmap)

# Assume enrichGO has been performed and result stored in 'ego'
# For example:
# ego <- enrichGO(gene = gene_entrez$ENTREZID,
#                 OrgDb = org.Hs.eg.db,
#                 ont = "BP",
#                 pvalueCutoff = 0.05,
#                 readable = TRUE)

# Select top enriched GO terms
top_terms <- head(ego, n = 15)

# Extract gene lists
gs_matrix <- setReadable(ego, OrgDb = org.Hs.eg.db)@result
gs_matrix <- gs_matrix[gs_matrix$Description %in% top_terms$Description, ]

# Create binary matrix of gene-term relationships
gene_lists <- strsplit(gs_matrix$geneID, "/")
names(gene_lists) <- gs_matrix$Description
unique_genes <- unique(unlist(gene_lists))
binary_mat <- matrix(0, nrow = length(unique_genes), ncol = length(gene_lists))
rownames(binary_mat) <- unique_genes
colnames(binary_mat) <- names(gene_lists)
for (term in names(gene_lists)) {
binary_mat[gene_lists[[term]], term] <- 1
}

# Plot heatmap
pheatmap(binary_mat,
        cluster_rows = TRUE,
        cluster_cols = TRUE,
        display_numbers = FALSE,
        color = colorRampPalette(c("white", "#0072B2"))(50),
        main = "Gene-Term Heatmap (GO Enrichment)",
        fontsize = 10)
```

D) Volcano Plot Script:

```
# Volcano plot from GEO-style Excel matrix
# File: volcano_GSE3001.R
# How to run (in R):
# install.packages(c("readxl","limma","ggplot2"))
# setwd("PATH/TO/WHERE/THE/EXCEL/IS")
# source("volcano_GSE3001.R")

suppressPackageStartupMessages({
  if (!requireNamespace("readxl", quietly = TRUE)) install.packages("readxl")
  if (!requireNamespace("limma", quietly = TRUE)) install.packages("limma")
  if (!requireNamespace("ggplot2", quietly = TRUE)) install.packages("ggplot2")
})

library(readxl)
library(limma)
library(ggplot2)

# === USER: set these ===
excel_path <- "GSE3001_series_matrix.xlsx"  # Put the path to your file (same name as uploaded)
sheet_name <- "GSE3001_series_matrix"       # Sheet to read (detected from your file)
# If you know sample grouping, edit this vector accordingly.
# For this file there are 10 GSM columns; by default we assume first 5 are GroupA and last 5 are GroupB.
group_labels <- c(rep("GroupA", 5), rep("GroupB", 5))
# Volcano thresholds
lfc_thresh <- 1          # absolute log2 fold-change cutoff
p_adj_thresh <- 0.05     # FDR cutoff

# === Read Excel and find header row (where first column is 'ID_REF') ===
raw <- read_excel(excel_path, sheet = sheet_name, col_names = FALSE)
header_row <- which(raw[[1]] == "ID_REF")[1]
if (is.na(header_row)) stop("Could not find a header row labeled 'ID_REF' in the first column.")

# Re-read with proper header starting at header_row
dat <- read_excel(excel_path, sheet = sheet_name, skip = header_row - 1)
# Expect first column 'ID_REF' + 10 GSM columns in this file
if (!"ID_REF" %in% names(dat)) {
  stop("Header row detected, but 'ID_REF' column not found after re-reading.")
}

# Convert to expression matrix
expr <- as.data.frame(dat, check.names = FALSE)
# keep feature IDs as rownames
rownames(expr) <- as.character(expr$ID_REF)
expr$ID_REF <- NULL

# Coerce all remaining columns to numeric
expr_mat <- as.matrix(sapply(expr, function(x) as.numeric(as.character(x))))
if (any(is.na(expr_mat))) {
  message("Note: Some non-numeric values were coerced to NA.")
}

# Basic sanity check
```

```
n_samples <- ncol(expr_mat)
if (length(group_labels) != n_samples) {
  stop(sprintf("Length of group_labels (%d) must equal number of samples (%d). Edit 'group_labels'
above.",
               length(group_labels),  n_samples))
}
group <- factor(group_labels)
design <- model.matrix(~ 0 + group)    # no intercept, design columns per group
colnames(design) <- levels(group)

# Fit limma model
fit <- lmFit(expr_mat, design)
# contrast: GroupB vs GroupA (edit if you swapped labels)
contrast.matrix <- makeContrasts(GroupB - GroupA, levels = design)
fit2 <- contrasts.fit(fit, contrast.matrix)
fit2 <- eBayes(fit2)

tt <- topTable(fit2, number = Inf, sort.by = "P")
tt$Symbol <- rownames(tt)
# Ensure columns named logFC and adj.P.Val exist (limma provides them)
res <- tt[, c("Symbol", "logFC", "P.Value", "adj.P.Val")]
rownames(res) <- NULL

# Save results
write.csv(res, "DE_results_limma.csv", row.names = FALSE)

# Volcano data
res$negLog10FDR <- -log10(res$adj.P.Val)
res$Signif <- with(res, ifelse(adj.P.Val < p_adj_thresh & abs(logFC) >= lfc_thresh,
                               ifelse(logFC > 0, "Up", "Down"),
                               "NS"))

# Plot
p <- ggplot(res, aes(x = logFC, y = negLog10FDR)) +
  geom_point(aes(shape = Signif), alpha = 0.7, size = 1.6) +
  geom_vline(xintercept = c(-lfc_thresh, lfc_thresh), linetype = "dashed") +
  geom_hline(yintercept = -log10(p_adj_thresh), linetype = "dashed") +
  labs(title = "Volcano plot (GroupB vs GroupA)",
       x = "log2 fold change",
       y = expression(-log[10]("FDR (adj.P.Val)"))) +
  theme_minimal()

ggsave("volcano_plot.png", plot = p, width = 7, height = 5, dpi = 300)

message("Done. Files written: DE_results_limma.csv and volcano_plot.png")
```

E) Gene Ontology diagram

```
# Load required packages
if (!requireNamespace("clusterProfiler", quietly = TRUE)) install.packages("BiocManager")
if (!requireNamespace("org.Hs.eg.db", quietly = TRUE)) BiocManager::install("org.Hs.eg.db")
if (!requireNamespace("igraph", quietly = TRUE)) install.packages("igraph")
if (!requireNamespace("ggraph", quietly = TRUE)) install.packages("ggraph")
if (!requireNamespace("tidyverse", quietly = TRUE)) install.packages("tidyverse")

library(clusterProfiler)
library(org.Hs.eg.db)
library(igraph)
library(ggraph)
library(tidyverse)

# 1. Define your gene list (use official gene symbols)
genes <- c("SP1", "NMYC", "GATA2", "CDX2", "HDAC2", "FLI1", "ZEB1",
          "CDH1", "CDH2", "VIM", "CTNNB1", "CD44", "PRKACA", "EIF2AK2",
          "PLCB1", "JUN", "BCL2", "BAX", "CASP3", "CASP8", "CASP9")

# 2. Convert to Entrez IDs
entrez_ids <- bitr(genes, fromType = "SYMBOL", toType = "ENTREZID", OrgDb = org.Hs.eg.db)

# 3. Run GO Biological Process enrichment
ego <- enrichGO(gene          = entrez_ids$ENTREZID,
                OrgDb         = org.Hs.eg.db,
                keyType       = "ENTREZID",
                ont           = "BP",
                pAdjustMethod = "BH",
                qvalueCutoff  = 0.05,
                readable      = TRUE)

# 4. Extract top GO terms and gene links
go_terms <- ego@result %>%
filter(p.adjust < 0.05) %>%
  head(20) %>%
  separate_rows(geneID, sep = "/") %>%
  select(geneID, Description)

# 5. Create edge list
edges <- go_terms %>%
  rename(from = geneID, to = Description)

# 6. Create node list with type (Gene or GO)
nodes <- data.frame(name = unique(c(edges$from, edges$to))) %>%
  mutate(type = ifelse(name %in% genes, "Gene", "GO Term"))

# 7. Create igraph object
g <- graph_from_data_frame(d = edges, vertices = nodes, directed = FALSE)

# 8. Plot using ggraph
ggraph(g, layout = "fr") +
  geom_edge_link(alpha = 0.5) +
  geom_node_point(aes(color = type), size = 5) +
```

```
  geom_node_text(aes(label = name), repel = TRUE, size = 3) +
  scale_color_manual(values = c("Gene" = "forestgreen", "GO Term" = "deeppink")) +
  theme_minimal() +
  ggtitle("GO Biological Process Network in GBM Genes")
```

F) Enrichment script:

```
# Load required libraries
library(clusterProfiler)
library(enrichplot)
library(pheatmap)

# Assume enrichGO has been performed and result stored in 'ego'
# For example:
# ego <- enrichGO(gene = gene_entrez$ENTREZID,
#                 OrgDb = org.Hs.eg.db,
#                 ont = "BP",
#                 pvalueCutoff = 0.05,
#                 readable = TRUE)

# Select top enriched GO terms
top_terms <- head(ego, n = 15)

# Extract gene lists
gs_matrix <- setReadable(ego, OrgDb = org.Hs.eg.db)@result
gs_matrix <- gs_matrix[gs_matrix$Description %in% top_terms$Description, ]

# Create binary matrix of gene-term relationships
gene_lists <- strsplit(gs_matrix$geneID, "/")
names(gene_lists) <- gs_matrix$Description
unique_genes <- unique(unlist(gene_lists))
binary_mat <- matrix(0, nrow = length(unique_genes), ncol = length(gene_lists))
rownames(binary_mat) <- unique_genes
colnames(binary_mat) <- names(gene_lists)
for (term in names(gene_lists)) {
binary_mat[gene_lists[[term]], term] <- 1
}

# Plot heatmap
pheatmap(binary_mat,
         cluster_rows = TRUE,
         cluster_cols = TRUE,
         display_numbers = FALSE,
         color = colorRampPalette(c("white", "#0072B2"))(50),
         main = "Gene-Term Heatmap (GO Enrichment)",
         fontsize = 10)
```