Supporting Information

Characterizing DPPM Inhibition of Butyrylcholinesterase: Integrated Enzymatic Kinetics and Raman Spectroscopy with Chemometric Analysis

*Aubrey Barney[1], Ashley Newland[1], Abraham Olayeri[1], Jan Halámek[1,2], Lenka Halámková[1]**

[1] Department of Environmental Toxicology, Texas Tech University, Lubbock, Texas 79409, USA

[2] Department of Chemistry, Tennessee State University, Murfreesboro, TN 37132, USA

KEYWORDS: Butyrylcholinesterase, fentanyl, Raman spectroscopy, machine learning

ENZYME KINETICS ANALYSIS USING "renz" PACKAGE

Using dir.MM() function for DPMM inhibition analysis

Load required libraries

library(renz)

library(ggplot2)

library(dplyr)

library(knitr)

#==========

DATA PREPARATION - YOUR UPDATED DATA

#==========

No inhibitor data

```
df_no_inhib <- data.frame(
  S = c(100, 150, 200, 250, 250, 500, 500, 750, 750, 1000, 1000),
  v = c(718.0802555, 899.0396459, 1121.29999, 1241.799467, 1247.8, 1978.098493,
        1887.9, 2324.197687, 2067.6, 2506.598621, 2195.5),
  I = 0
)
# DPMM inhibitor data
df_inhib <- data.frame(
  S = c(100, 150, 200, 250, 250, 500, 500, 750, 750, 1000, 1000),
  v = c(414.2400828, 567.8401462, 695.0400552, 779.0400824, 685.11, 1122.699869,
        1148.5, 1251.799462, 1341.1, 1481.799794, 1550.7),
  I = 566.123
)
# Prepare data in renz format (average duplicates for cleaner analysis)
df_no_inhib_avg <- df_no_inhib %>%
  group_by(S) %>%
  summarise(v = mean(v), .groups = 'drop')
df_inhib_avg <- df_inhib %>%
  group_by(S) %>%
  summarise(v = mean(v), .groups = 'drop')
# Create combined data frame for renz (similar to your original format)
renz_data <- data.frame(
  S = df_no_inhib_avg$S,
  V_Control = df_no_inhib_avg$v,
  V_DPMM = df_inhib_avg$v
)
```

```
#=================================================================
==========
# RENZ ANALYSIS - NO INHIBITOR (Control)
#=================================================================
==========
# Analyze control data using dir.MM
control_analysis <- dir.MM(renz_data[, c("S", "V_Control")], unit_v = "microM/min")
control_analysis
#=================================================================
==========
# RENZ ANALYSIS - WITH DPMM INHIBITOR
#=================================================================
==========
# Analyze DPMM inhibitor data using dir.MM
dpmm_analysis <- dir.MM(renz_data[, c("S", "V_DPMM")], unit_v = "microM/min")
dpmm_analysis

# ENZYME KINETICS ANALYSIS WITH BALANCED BIOCHEMICAL CONSTRAINTS
# Analysis of DPMM inhibition using restrictive constraints
# Using robust parametric confidence intervals

# Load required libraries
library(minpack.lm)  # For bounded non-linear fitting
library(ggplot2)     # For plotting
library(gridExtra)   # For multiple plots
library(dplyr)       # For data manipulation
library(knitr)       # For nice tables
```

```
#============================================================
====================
# DATA PREPARATION
#============================================================
====================

# No inhibitor data
df_no_inhib <- data.frame(
  S = c(100, 150, 200, 250, 250, 500, 500, 750, 750, 1000, 1000),
  v = c(718.0802555, 899.0396459, 1121.29999, 1241.799467, 1247.8, 1978.098493,
        1887.9, 2324.197687, 2067.6, 2506.598621, 2195.5),
  I = 0
)

# DPMM inhibitor data
df_inhib <- data.frame(
  S = c(100, 150, 200, 250, 250, 500, 500, 750, 750, 1000, 1000),
  v = c(414.2400828, 567.8401462, 695.0400552, 779.0400824, 685.11, 1122.699869,
        1148.5, 1251.799462, 1341.1, 1481.799794, 1550.7),
  I = 566.123
)

# Combine datasets
df_all <- rbind(df_no_inhib, df_inhib)
df_all$condition <- ifelse(df_all$I == 0, "No Inhibitor", "DPMM 566 μM")

print("Data prepared successfully")
```

```
#=============================================================
====================
# BASELINE MODEL FITTING (NO INHIBITOR)
#=============================================================
====================

cat("\n=== BASELINE MODEL FITTING ===\n")

# Fit baseline Michaelis-Menten model
fit_baseline <- nls(v ~ (Vm * S) / (Km + S),
                    data = df_no_inhib,
                    start = list(Vm = max(df_no_inhib$v), Km = median(df_no_inhib$S)),
                    control = nls.control(maxiter = 500))

# Extract baseline parameters
baseline_Vm <- coef(fit_baseline)["Vm"]
baseline_Km <- coef(fit_baseline)["Km"]

cat("Baseline Parameters:\n")
cat("Vmax =", round(baseline_Vm, 1), "μM/min\n")
cat("Km =", round(baseline_Km, 1), "μM\n")

#=============================================================
====================
# INHIBITION MODEL FITTING WITH BALANCED BIOCHEMICAL CONSTRAINTS
#=============================================================
====================
```

```
cat("\n=== INHIBITION MODEL FITTING WITH BALANCED CONSTRAINTS ===\n")

# Model 1: COMPETITIVE INHIBITION
# Expected: Km increases moderately, Vmax stays roughly constant
# v = (Vm * S) / (Km * (1 + I/Ki) + S)

fit_competitive <- nlsLM(
  v ~ (Vm * S) / (Km * (1 + I/Ki) + S),
  data = df_all,
  start = list(Vm = baseline_Vm, Km = baseline_Km * 1.1, Ki = 500),
  lower = c(Vm = baseline_Vm * 0.90,          # Vmax: allow 10% decrease max
            Km = baseline_Km * 1.01,          # Km: must increase slightly
            Ki = 50),
  upper = c(Vm = baseline_Vm * 1.10,          # Vmax: allow 10% increase max
            Km = baseline_Km * 2.5,           # Km: allow significant increase
            Ki = 3000),
  control = nls.control(maxiter = 1000)
)

# Model 2: NONCOMPETITIVE INHIBITION
# Expected: Km roughly unchanged, Vmax decreases
# v = (Vm * S) / ((Km + S) * (1 + I/Ki))

fit_noncompetitive <- nlsLM(
  v ~ (Vm * S) / ((Km + S) * (1 + I/Ki)),
  data = df_all,
  start = list(Vm = baseline_Vm * 0.65, Km = baseline_Km, Ki = 800),
```

```
  lower = c(Vm = baseline_Vm * 0.45,          # Vmax: must decrease significantly
            Km = baseline_Km * 0.90,          # Km: allow 10% decrease max
            Ki = 100),
  upper = c(Vm = baseline_Vm * 0.85,          # Vmax: limit increase
            Km = baseline_Km * 1.10,          # Km: allow 10% increase max
            Ki = 2500),
  control = nls.control(maxiter = 1000)
)

# Model 3: UNCOMPETITIVE INHIBITION
# Expected: Both Km and Vmax decrease proportionally
# v = (Vm * S) / (Km + S * (1 + I/Ki))

fit_uncompetitive <- nlsLM(
  v ~ (Vm * S) / (Km + S * (1 + I/Ki)),
  data = df_all,
  start = list(Vm = baseline_Vm * 0.65, Km = baseline_Km * 0.65, Ki = 400),
  lower = c(Vm = baseline_Vm * 0.45,          # Both must decrease
            Km = baseline_Km * 0.45,          # Both must decrease
            Ki = 100),
  upper = c(Vm = baseline_Vm * 0.85,          # Limit how high they can go
            Km = baseline_Km * 0.85,          # Limit how high they can go
            Ki = 2500),
  control = nls.control(maxiter = 1000)
)

# Model 4: MIXED INHIBITION
```

```
# Expected: Vmax decreases, Km can increase or decrease
# v = (Vm * S) / (Km * (1 + I/Kic) + S * (1 + I/Kiu))

fit_mixed <- nlsLM(
  v ~ (Vm * S) / (Km * (1 + I/Kic) + S * (1 + I/Kiu)),
  data = df_all,
  start = list(Vm = baseline_Vm * 0.65, Km = baseline_Km, Kic = 400, Kiu = 600),
  lower = c(Vm = baseline_Vm * 0.45,          # Vmax must decrease
            Km = baseline_Km * 0.60,          # Km has flexibility
            Kic = 100,
            Kiu = 100),
  upper = c(Vm = baseline_Vm * 0.90,          # Limit Vmax increase
            Km = baseline_Km * 1.8,           # Allow Km variation
            Kic = 3000,
            Kiu = 3000),
  control = nls.control(maxiter = 1000)
)

#============================================================
# EXTRACT PARAMETERS AND CONFIDENCE INTERVALS
#============================================================

cat("\n=== CALCULATING CONFIDENCE INTERVALS ===\n")

# Function to extract parameters with robust confidence intervals
extract_model_info <- function(model, model_name) {
```

```
params <- coef(model)
model_summary <- summary(model)

cat("Processing", model_name, "model...\n")

# Find the correct parameter names
vm_name <- names(params)[grep("Vm", names(params))][1]
km_name <- names(params)[grep("Km", names(params))][1]

# Extract estimates
vm_est <- params[vm_name]
km_est <- params[km_name]

# Extract standard errors from coefficient table
coeff_table <- model_summary$coefficients
vm_se <- coeff_table[vm_name, "Std. Error"]
km_se <- coeff_table[km_name, "Std. Error"]

# Calculate 95% confidence intervals using t-distribution
df_residual <- model_summary$df[2]
t_val <- qt(0.975, df = df_residual)

vm_ci <- c(vm_est - t_val * vm_se, vm_est + t_val * vm_se)
km_ci <- c(km_est - t_val * km_se, km_est + t_val * km_se)

# Calculate model statistics
rse <- model_summary$sigma
```

```
  aic <- AIC(model)

  # Create parameter table for Vm and Km only
  param_df <- data.frame(
    Model = rep(model_name, 2),
    Parameter = c("Vm", "Km"),
    Estimate = round(c(vm_est, km_est), 1),
    CI_Lower = round(c(vm_ci[1], km_ci[1]), 1),
    CI_Upper = round(c(vm_ci[2], km_ci[2]), 1),
    RSE = round(rse, 2),
    AIC = round(aic, 2),
    stringsAsFactors = FALSE
  )

  cat("Successfully calculated CIs for", model_name, "\n")
  return(list(params = param_df, aic = aic, rse = rse))
}

# Extract information for all models
models <- list(
  "Competitive" = fit_competitive,
  "Noncompetitive" = fit_noncompetitive,
  "Uncompetitive" = fit_uncompetitive,
  "Mixed" = fit_mixed
)

model_results <- lapply(names(models), function(name) {
```

```
  extract_model_info(models[[name]], name)
})
names(model_results) <- names(models)

# Combine all parameter estimates into one table
all_params <- do.call(rbind, lapply(model_results, function(x) x$params))

# Create summary table for model comparison
summary_table <- data.frame(
  Model = names(models),
  AIC = round(sapply(model_results, function(x) x$aic), 2),
  RSE = round(sapply(model_results, function(x) x$rse), 2),
  n_params = sapply(models, function(x) length(coef(x))),
  stringsAsFactors = FALSE
)

# Calculate delta AIC
best_aic <- min(summary_table$AIC)
summary_table$Delta_AIC <- round(summary_table$AIC - best_aic, 2)
summary_table$Support <- ifelse(summary_table$Delta_AIC < 2, "Strong",
                         ifelse(summary_table$Delta_AIC < 4, "Moderate",
                             ifelse(summary_table$Delta_AIC < 10, "Weak", "None")))

# Identify best model
best_model_name <- summary_table$Model[which.min(summary_table$AIC)]
best_model <- models[[best_model_name]]
```

```
#============================================================
====================
# BIOCHEMICAL VALIDATION
#============================================================
====================

cat("\n=== BIOCHEMICAL VALIDATION ===\n")

validate_biochemistry <- function(model, model_name, baseline_Vm, baseline_Km) {
  params <- coef(model)

  # Find correct parameter names
  vm_name <- names(params)[grep("Vm", names(params))][1]
  km_name <- names(params)[grep("Km", names(params))][1]

  vm_change <- (params[vm_name] - baseline_Vm) / baseline_Vm * 100
  km_change <- (params[km_name] - baseline_Km) / baseline_Km * 100

  cat(sprintf("\n%s Inhibition:\n", model_name))
  cat(sprintf("Vmax: %.1f μM/min (%.1f%% change from baseline)\n",
              params[vm_name], vm_change))
  cat(sprintf("Km: %.1f μM (%.1f%% change from baseline)\n",
              params[km_name], km_change))

  # Validation criteria with inclusive inequalities
  if(model_name == "Competitive") {
    valid <- abs(vm_change) <= 12 && km_change >= 1
    cat(sprintf("Expected: Vmax ~unchanged, Km increases - Valid: %s\n",
```

```
            ifelse(valid, "YES", "NO")))
  } else if(model_name == "Noncompetitive") {
    valid <- vm_change <= -10 && abs(km_change) <= 12
    cat(sprintf("Expected: Vmax decreases, Km ~unchanged - Valid: %s\n",
            ifelse(valid, "YES", "NO")))
  } else if(model_name == "Uncompetitive") {
    valid <- vm_change <= -10 && km_change <= -10
    cat(sprintf("Expected: Both Vmax and Km decrease - Valid: %s\n",
            ifelse(valid, "YES", "NO")))
  } else if(model_name == "Mixed") {
    valid <- vm_change <= -5
    cat(sprintf("Expected: Vmax decreases - Valid: %s\n",
            ifelse(valid, "YES", "NO")))
  }

  return(valid)
}

# Validate all models
for(name in names(models)) {
  validate_biochemistry(models[[name]], name, baseline_Vm, baseline_Km)
}

#============================================================
======================
# VISUALIZATION - ALL FOUR MODELS
#============================================================
======================
```

```
cat("\n=== GENERATING COMPREHENSIVE PLOTS ===\n")

# Function to generate predictions
generate_predictions <- function(model, model_name, s_range = seq(50, 1200, by = 20)) {
  params <- coef(model)

  # Find correct parameter names
  vm_name <- names(params)[grep("Vm", names(params))][1]
  km_name <- names(params)[grep("Km", names(params))][1]
  ki_name <- names(params)[grep("Ki", names(params))][1]

  # Create prediction data for both conditions
  pred_list <- list()

  for(I in c(0, 566.123)) {
    if(model_name == "Competitive") {
      v_pred <- (params[vm_name] * s_range) / (params[km_name] * (1 + I/params[ki_name]) +
s_range)
    } else if(model_name == "Noncompetitive") {
      v_pred <- (params[vm_name] * s_range) / ((params[km_name] + s_range) * (1 +
I/params[ki_name]))
    } else if(model_name == "Uncompetitive") {
      v_pred <- (params[vm_name] * s_range) / (params[km_name] + s_range * (1 +
I/params[ki_name]))
    } else if(model_name == "Mixed") {
      kic_name <- names(params)[grep("Kic", names(params))][1]
      kiu_name <- names(params)[grep("Kiu", names(params))][1]
```

```
    v_pred <- (params[vm_name] * s_range) / (params[km_name] * (1 + I/params[kic_name]) +
                                            s_range * (1 + I/params[kiu_name]))
  }

  condition <- ifelse(I == 0, "No Inhibitor", "DPMM 566 μM")
  pred_list[[condition]] <- data.frame(
    S = s_range,
    v = v_pred,
    condition = condition,
    model = model_name
  )
 }

 return(do.call(rbind, pred_list))
}

# Generate predictions for all models
all_predictions <- do.call(rbind, lapply(names(models), function(name) {
 generate_predictions(models[[name]], name)
}))

# Create individual plots for each model
plot_list <- list()

for(model_name in names(models)) {
 model_pred <- all_predictions[all_predictions$model == model_name, ]
 model_aic <- summary_table$AIC[summary_table$Model == model_name]
```

```
  p <- ggplot() +
    geom_line(data = model_pred, aes(x = S, y = v, color = condition),
              size = 1.2) +
    geom_point(data = df_all, aes(x = S, y = v, color = condition),
               size = 2.5, alpha = 0.7) +
    scale_color_manual(values = c("No Inhibitor" = "#1E3A8A", "DPMM 566 μM" =
"#922B21")) +
    labs(
      title = paste(model_name, "Inhibition"),
      subtitle = paste("AIC =", round(model_aic, 1)),
      x = "Substrate Concentration (μM)",
      y = "Velocity (μM/min)",
      color = "Condition"
    ) +
    theme_minimal() +
    theme(
      plot.title = element_text(size = 12, face = "bold"),
      legend.position = "bottom",
      legend.title = element_text(size = 10),
      legend.text = element_text(size = 9)
    ) +
    ylim(0, max(df_all$v) * 1.1)

  plot_list[[model_name]] <- p
}
```

```
# Arrange all four plots
combined_plot <- do.call(grid.arrange, c(plot_list, ncol = 2))

# AIC comparison plot
aic_plot <- ggplot(summary_table, aes(x = reorder(Model, AIC), y = AIC)) +
  geom_col(aes(fill = Support), alpha = 0.8, width = 0.6) +
  geom_text(aes(label = paste("AIC =", AIC)), vjust = -0.3, size = 3.5) +
  scale_fill_manual(values = c("Strong" = "#27AE60", "Moderate" = "#F39C12",
                               "Weak" = "#E67E22", "None" = "#E74C3C")) +
  labs(
    title = "Model Comparison: AIC Values",
    subtitle = "Lower AIC indicates better model fit",
    x = "Inhibition Model",
    y = "AIC Value",
    fill = "Statistical Support"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 14, face = "bold"),
    axis.text.x = element_text(angle = 45, hjust = 1)
  )

print(aic_plot)

#============================================================
====================
# RESULTS TABLES
```

```
#=============================================================
====================

cat("\n=== RESULTS TABLES ===\n")

# Create formatted parameter table
param_wide <- all_params %>%
  select(Model, Parameter, Estimate, CI_Lower, CI_Upper) %>%
  tidyr::pivot_wider(names_from = Parameter,
                     values_from = c(Estimate, CI_Lower, CI_Upper),
                     names_sep = "_")

# Merge with summary statistics
final_table <- merge(param_wide, summary_table[, c("Model", "AIC", "Delta_AIC", "Support",
"RSE")],
                     by = "Model")

# Reorder columns for readability
final_table <- final_table[, c("Model", "Estimate_Vm", "CI_Lower_Vm", "CI_Upper_Vm",
                               "Estimate_Km", "CI_Lower_Km", "CI_Upper_Km",
                               "AIC", "Delta_AIC", "Support", "RSE")]

# Print tables
cat("\nParameter Estimates with 95% Confidence Intervals:\n")
print(kable(final_table,
            col.names = c("Model", "Vmax", "CI Lower", "CI Upper",
                          "Km", "CI Lower", "CI Upper",
                          "AIC", "ΔAIC", "Support", "RSE"),
```

```
        digits = c(0, 1, 1, 1, 1, 1, 1, 2, 2, 0, 2),
        format = "simple"))

cat("\nModel Comparison Summary:\n")
print(kable(summary_table, format = "simple", digits = 2))
```