# Mapping the hidden topology of software risk in scientific models

Arnald Puy

## Contents

# 1 Preliminary functions

```
#   PRELIMINARY FUNCTIONS ####################################################

sensobol::load_packages(c("data.table", "tidyverse", "openxlsx", "scales",
                          "cowplot", "readxl", "ggrepel", "tidytext", "here",
                          "tidygraph", "igraph", "foreach", "parallel", "ggraph",
                          "tools", "purrr"))

# Create custom theme ----------------------------------------------------------

theme_AP <- function() {
  theme_bw() +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          legend.background = element_rect(fill = "transparent", color = NA),
          legend.key = element_rect(fill = "transparent", color = NA),
          strip.background = element_rect(fill = "white"),
          legend.text = element_text(size = 7.3),
          axis.title = element_text(size = 10),
          legend.key.width = unit(0.4, "cm"),
          legend.key.height = unit(0.4, "cm"),
          legend.key.spacing.y = unit(0, "lines"),
          legend.box.spacing = unit(0, "pt"),
          legend.title = element_text(size = 7.3),
          axis.text.x = element_text(size = 7),
          axis.text.y = element_text(size = 7),
          axis.title.x = element_text(size = 7.3),
          axis.title.y = element_text(size = 7.3),
          plot.title = element_text(size = 8),
          strip.text.x = element_text(size = 7.4),
          strip.text.y = element_text(size = 7.4))
}

# Select color palette ---------------------------------------------------------

color_languages <- c("fortran" = "steelblue", "python" = "lightgreen")

# Source all .R files in the "functions" folder --------------------------------

r_functions <- list.files(path = here("functions"),
                          pattern = "\\.R$", full.names = TRUE)

lapply(r_functions, source)
```

```
## [[1]]
## [[1]]$value
## function (call_g)
```

```
## {
##     ig <- as.igraph(call_g)
##     V(ig)$name <- call_g %>% activate(nodes) %>% as_tibble() %>%
##         pull(name)
##     node_df <- call_g %>% activate(nodes) %>% as_tibble()
##     get_path_risks_fun <- function(nodes_vec) {
##         idx <- match(nodes_vec, node_df$name)
##         node_df$risk_node[idx]
##     }
##     entry_ids <- which(node_df$indeg == 0)
##     sink_ids <- which(node_df$outdeg == 0)
##     if (length(entry_ids) == 0 || length(sink_ids) == 0) {
##         return(tibble())
##     }
##     dist_mat <- igraph::distances(ig, v = entry_ids, to = sink_ids,
##         mode = "out")
##     pairs_st <- expand.grid(s = seq_along(entry_ids), t = seq_along(sink_ids))
##     pairs_st <- pairs_st[pairs_st$s != pairs_st$t, , drop = FALSE]
##     pairs_st <- subset(pairs_st, dist_mat[cbind(s, t)] < Inf)
##     if (nrow(pairs_st) == 0) {
##         return(tibble())
##     }
##     all_paths_nested <- lapply(seq_len(nrow(pairs_st)), function(i) {
##         from_v <- entry_ids[pairs_st$s[i]]
##         to_v <- sink_ids[pairs_st$t[i]]
##         igraph::all_simple_paths(ig, from = from_v, to = to_v,
##             mode = "out")
##     })
##     all_paths <- purrr::flatten(all_paths_nested)
##     if (length(all_paths) == 0)
##         return(tibble())
##     risk_of_path_fun <- function(path_vertices) {
##         idx <- as.integer(path_vertices)
##         node_names <- node_df$name[idx]
##         rn <- node_df$risk_node[idx]
##         tibble(path_nodes = list(node_names), path_str = paste(node_names,
##             collapse = " → "), hops = length(idx) - 1, risk_sum = sum(rn,
##             na.rm = TRUE), risk_mean = mean(rn, na.rm = TRUE))
##     }
##     paths_tbl <- purrr::map_dfr(all_paths, risk_of_path_fun)
##     if (nrow(paths_tbl) == 0)
##         return(paths_tbl)
##     paths_tbl <- paths_tbl %>% mutate(max_node_risk = map_dbl(path_nodes,
##         ~{
##             r <- get_path_risks_fun(.x)
##             max(r, na.rm = TRUE)
##         }), p_path_fail = map_dbl(path_nodes, ~{
##         r <- get_path_risks_fun(.x)
```

```
##         r <- r[is.finite(r) & !is.na(r)]
##         if (length(r) == 0)
##             0
##         else 1 - prod(pmax(0, pmin(1, 1 - r)))
##     }), gini_node_risk = map_dbl(path_nodes, ~{
##         r <- get_path_risks_fun(.x)
##         r <- r[is.finite(r) & !is.na(r)]
##         if (length(r) <= 1)
##             0
##         else ineq::Gini(r)
##     }), risk_slope = map_dbl(path_nodes, ~{
##         r <- get_path_risks_fun(.x)
##         r <- r[is.finite(r) & !is.na(r)]
##         if (length(r) <= 1)
##             0
##         else as.numeric(coef(stats::lm(r ~ seq_along(r)))[2])
##     }))
##     paths_tbl
## }
##
## [[1]]$visible
## [1] FALSE
##
##
## [[2]]
## [[2]]$value
## function (plot, legend = NULL)
## {
##     if (is.ggplot(plot)) {
##         gt <- ggplotGrob(plot)
##     }
##     else {
##         if (is.grob(plot)) {
##             gt <- plot
##         }
##         else {
##             stop("Plot object is neither a ggplot nor a grob.")
##         }
##     }
##     pattern <- "guide-box"
##     if (!is.null(legend)) {
##         pattern <- paste0(pattern, "-", legend)
##     }
##     indices <- grep(pattern, gt$layout$name)
##     not_empty <- !vapply(gt$grobs[indices], inherits, what = "zeroGrob",
##         FUN.VALUE = logical(1))
##     indices <- indices[not_empty]
##     if (length(indices) > 0) {
```

```
##         return(gt$grobs[[indices[1]]])
##     }
##     return(NULL)
## }
##
## [[2]]$visible
## [1] FALSE
##
##
## [[3]]
## [[3]]$value
## function (nodes_vec)
## {
##     node_df$risk_node[match(nodes_vec, node_df$name)]
## }
##
## [[3]]$visible
## [1] FALSE
##
##
## [[4]]
## [[4]]$value
## function (call_g, paths_tbl, model.name = "", language = "")
## {
##     if (nrow(paths_tbl) == 0) {
##         message("No paths in paths_tbl; skipping plot for: ",
##             model.name)
##         return(invisible(NULL))
##     }
##     top_paths <- paths_tbl %>% arrange(desc(p_path_fail)) %>%
##         slice_head(n = 10)
##     k <- min(10, nrow(top_paths))
##     top_k_paths <- top_paths %>% slice_head(n = k)
##     path_edges_all <- purrr::imap_dfr(top_k_paths$path_nodes,
##         function(nodes_vec, pid) {
##             tibble(from = head(nodes_vec, -1), to = tail(nodes_vec,
##                 -1), path_id = pid, risk_sum = top_k_paths$risk_sum[pid])
##         })
##     ig2 <- as.igraph(call_g)
##     edge_df_names <- igraph::as_data_frame(ig2, what = "edges") %>%
##         mutate(.edge_idx = dplyr::row_number())
##     path_edges_collapsed <- path_edges_all %>% dplyr::group_by(from,
##         to) %>% dplyr::summarise(path_freq = dplyr::n(), risk_mean_path = mean(risk_sum,
##         na.rm = TRUE), .groups = "drop")
##     edge_marks <- edge_df_names %>% dplyr::left_join(path_edges_collapsed,
##         by = c("from", "to")) %>% dplyr::mutate(on_top_path = !is.na(path_freq),
##         path_freq = ifelse(is.na(path_freq), 0L, path_freq),
##         risk_mean_path = ifelse(is.na(risk_mean_path), 0, risk_mean_path))
```

```
##     call_g_sugi <- call_g %>% activate(edges) %>% mutate(on_top_path = edge_marks$on_top_pat
##         path_freq = edge_marks$path_freq, risk_mean_path = edge_marks$risk_mean_path) %>%
##         activate(nodes) %>% mutate(indeg = indeg, cyclo_class = case_when(cyclomatic_complex
##         10 ~ "green", cyclomatic_complexity <= 20 ~ "orange",
##         cyclomatic_complexity <= 50 ~ "red", cyclomatic_complexity >
##             50 ~ "purple", TRUE ~ "grey"))
##     risky_nodes <- unique(unlist(top_k_paths$path_nodes))
##     call_g_sugi <- call_g_sugi %>% activate(nodes) %>% mutate(on_top_node = name %in%
##         risky_nodes)
##     node_df <- call_g_sugi %>% activate(nodes) %>% as_tibble()
##     risky_indeg <- node_df$indeg[node_df$on_top_node & is.finite(node_df$indeg)]
##     risky_indeg <- sort(unique(risky_indeg))
##     if (length(risky_indeg) == 0) {
##         risky_indeg <- sort(unique(node_df$indeg[is.finite(node_df$indeg)]))
##     }
##     if (length(risky_indeg) >= 3) {
##         min_indeg <- min(risky_indeg)
##         max_indeg <- max(risky_indeg)
##         mid_indeg <- risky_indeg[ceiling(length(risky_indeg)/2)]
##         legend_breaks <- c(min_indeg, mid_indeg, max_indeg)
##     }
##     else {
##         legend_breaks <- risky_indeg
##     }
##     legend_labels <- round(legend_breaks, 1)
##     p_sugi <- ggraph(call_g_sugi, layout = "sugiyama") + geom_edge_link0(aes(filter = !on_to
##         colour = "grey80", alpha = 0.05, width = 0.3) + geom_edge_link0(aes(filter = on_top_
##         colour = risk_mean_path, width = pmin(pmax(path_freq,
##             0.5), 3)), alpha = 0.9, arrow = grid::arrow(length = unit(1,
##         "mm"))) + scale_edge_colour_gradient(low = "orange",
##         high = "red", guide = "none") + scale_edge_width(range = c(0.3,
##         2.2), guide = "none") + geom_node_point(size = 1.2, colour = "#BDBDBD",
##         alpha = 0.35, show.legend = FALSE) + geom_node_point(aes(filter = on_top_node,
##         size = indeg, fill = cyclo_class), shape = 21, alpha = 0.95,
##         show.legend = TRUE) + scale_size_continuous(breaks = legend_breaks,
##         labels = legend_labels, name = "indegree") + scale_fill_manual(values = c(green = "
##         orange = "orange", red = "red", purple = "purple", grey = "#A0A0A0"),
##         guide = "none") + theme_AP() + labs(x = "", y = "") +
##         theme(axis.text.y = element_blank(), axis.ticks.y = element_blank(),
##             axis.text.x = element_blank(), axis.ticks.x = element_blank(),
##             legend.position = "right") + ggtitle(paste(model.name,
##         ": ", language, sep = ""))
##     print(p_sugi)
##     invisible(p_sugi)
## }
##
## [[4]]$visible
## [1] FALSE
```

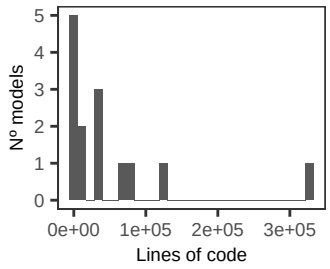
## 2 Results

```
# READ IN DATASET ##################################################################

# Get name of sheets -----------------------------------------------------------

sheets <- excel_sheets("./datasets/results_sqa.xlsx")

# Read all sheets --------------------------------------------------------------

dt <- lapply(sheets, function(x) data.table(read_excel("./datasets/results_sqa.xlsx",
                                                       sheet = x)))

# Name the slots ---------------------------------------------------------------

names(dt) <- sheets
```

### 2.1 Descriptive statistics

```
# PLOT LINES OF CODE ###############################################################

plot_lines_code <- dt$descriptive_stats[, .(total_lines_code = sum(lines_code)), model] %>%
  ggplot(., aes(total_lines_code)) +
  geom_histogram() +
  labs(x = "Lines of code", y = "Nº models") +
  theme_AP()

plot_lines_code
```

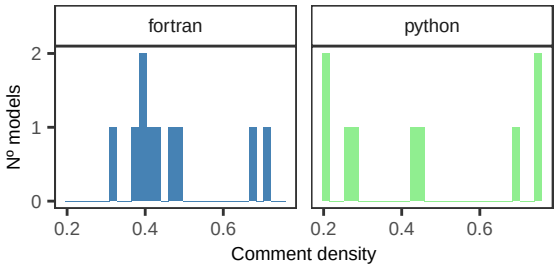
```
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```




```
# PLOT COMMENT DENSITY #############################################################

plot_comment_density <- dt$descriptive_stats[, .(total_lines_code = sum(lines_code),
       total_lines_comments = sum(lines_comments)), .(model, language)] %>%
  .[, comment_density:= total_lines_comments / total_lines_code] %>%
  ggplot(., aes(comment_density, fill = language)) +
  geom_histogram() +
  facet_wrap(~language) +
  scale_y_continuous(breaks = breaks_pretty(n = 3)) +
```

```
  scale_fill_manual(values = color_languages) +
  labs(x = "Comment density", y = "Nº models") +
  theme_AP() +
  theme(legend.position = "none")

plot_comment_density
```

```
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```



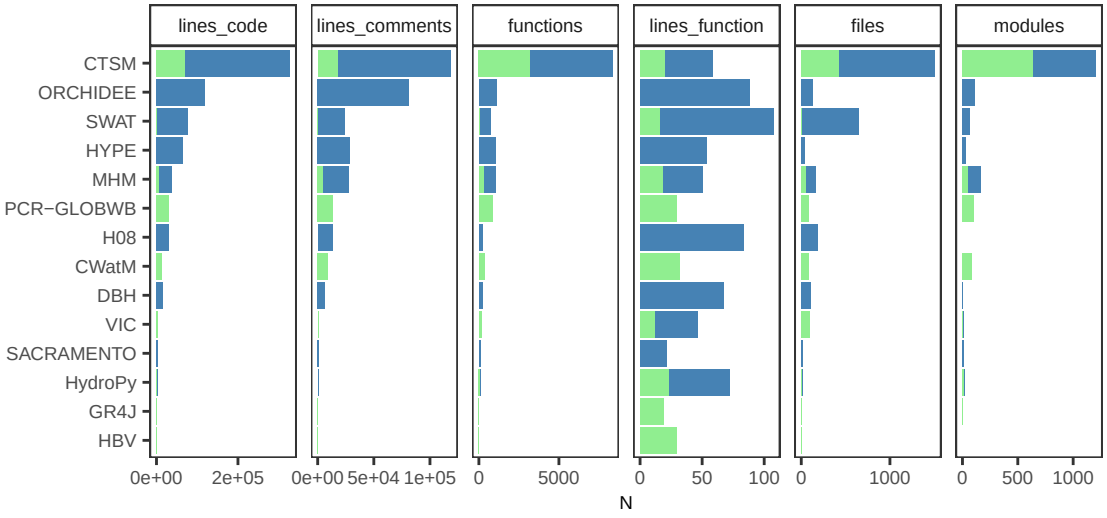

```
# PLOT PER MODEL ######################################################################

# Sort by model -------------------------------------------------------------------

model_ordered <- dt$descriptive_stats[, sum(lines), model] %>%
  .[order(V1)]

# Print ---------------------------------------------------------------------------

model_ordered
```

```
##          model     V1
##         <char>  <num>
##  1:        HBV    180
##  2:       GR4J    423
##  3:    HydroPy   3739
##  4: SACRAMENTO   5294
##  5:        VIC   5952
##  6:        DBH  24334
##  7:      CWatM  27745
##  8:        H08  42917
##  9: PCR-GLOBWB  52686
## 10:        MHM  76286
## 11:       HYPE  89137
## 12:       SWAT  99976
## 13:   ORCHIDEE 211871
## 14:       CTSM 491592
```

```
# Extract column names ------------------------------------------------------------

col_names <- colnames(dt$descriptive_stats)
```

```
# Order facets ------------------------------------------------------------

facet_order <- c("lines", "lines_code", "lines_comments", "functions",
                 "lines_function", "files", "modules")


# Plot --------------------------------------------------------------------

plot_per_model <- melt(dt$descriptive_stats, measure.vars = col_names[-c(1, length(col_names))]
  .[, variable:= factor(variable, levels = facet_order)] %>%
  .[, model:= factor(model, levels = model_ordered[, model])] %>%
  .[!variable == "lines"] %>%
  ggplot(., aes(model, value, fill = language)) +
  geom_col() +
  coord_flip() +
  scale_y_continuous(breaks = breaks_pretty(n = 2)) +
  scale_fill_manual(values = color_languages) +
  facet_wrap(~ variable, ncol = 7, scales = "free_x") +
  labs(x = "", y = "N") +
  theme_AP() +
  theme(legend.position = "none")

plot_per_model
```

```
## Warning: Removed 3 rows containing missing values or values outside the scale range
## (`geom_col()`).
```




```
# MERGE PLOTS ###################################################################

top <- plot_grid(plot_lines_code, plot_comment_density + labs(x = "Comment density", y  = ""),
          labels = "auto", rel_widths = c(0.4, 0.6))
```

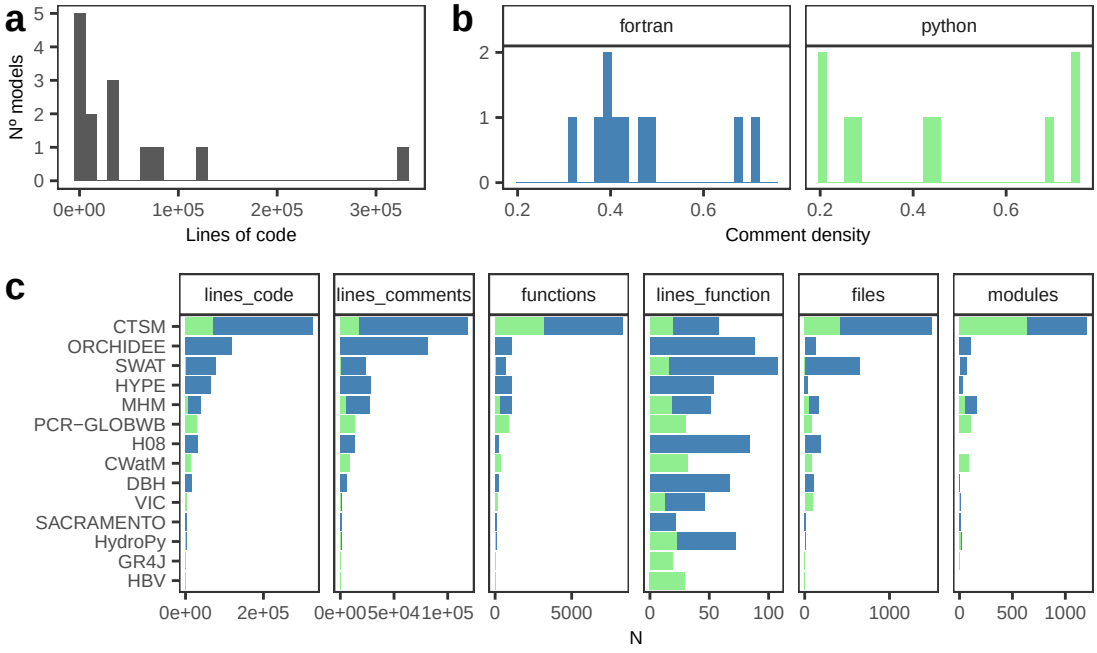
```
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```

```
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```

```
p1 <- plot_grid(top, plot_per_model, ncol = 1, labels = c("", "c"), rel_heights = c(0.4, 0.6))
```

```
## Warning: Removed 3 rows containing missing values or values outside the scale range
## (`geom_col()`).
```

```
p1
```



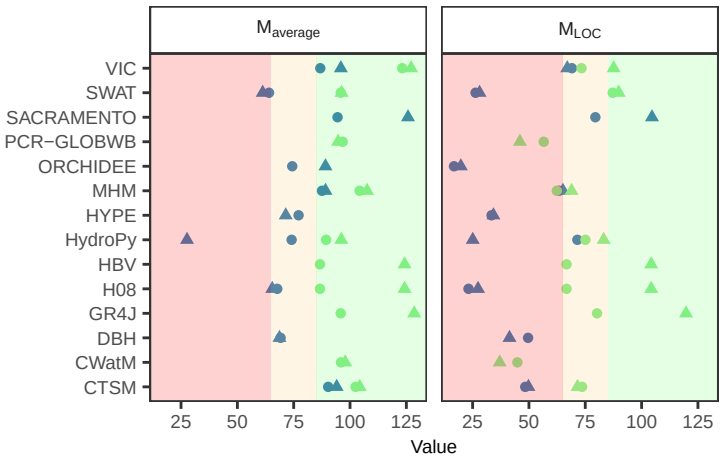

## 2.2 Maintainability index

```
# CALCULATE INTERPRETATIBILITY OF MAINTAINABILITY INDEX 3########################

# Define vector of interpretation ------------------------------------------

vec_interpretation <- c("low", "moderate", "high")

# Calculate ----------------------------------------------------------------

dt$maintainability_index %>%
  melt(., measure.vars = c("M_loc", "M_average")) %>%
  .[, interpretativity:= ifelse(value > 85, vec_interpretation[3],
                                ifelse(value <=85 & value >= 65, vec_interpretation[2],
                                       vec_interpretation[1]))] %>%
  .[, .N, .(language, interpretativity, variable)] %>%
  dcast(., variable + language ~ interpretativity, value.var = "N") %>%
  .[, total:= rowSums(.SD, na.rm = TRUE), .SDcols = vec_interpretation] %>%
  .[, paste(vec_interpretation, "prop", sep = "_"):= lapply(.SD, function(x)
    x / total), .SDcols = vec_interpretation] %>%
  print()
```

```
## Key: <variable, language>
##     variable language  high   low moderate total low_prop moderate_prop
##       <fctr>   <char> <int> <int>    <int> <num>    <num>          <num>
## 1:     M_loc  fortran     1    14        5    20     0.70           0.25
## 2:     M_loc   python     6     5        9    20     0.25           0.45
## 3: M_average  fortran     9     3        8    20     0.15           0.40
## 4: M_average   python    20    NA       NA    20       NA             NA
##    high_prop
##        <num>
## 1:      0.05
## 2:      0.30
## 3:      0.45
## 4:      1.00
```

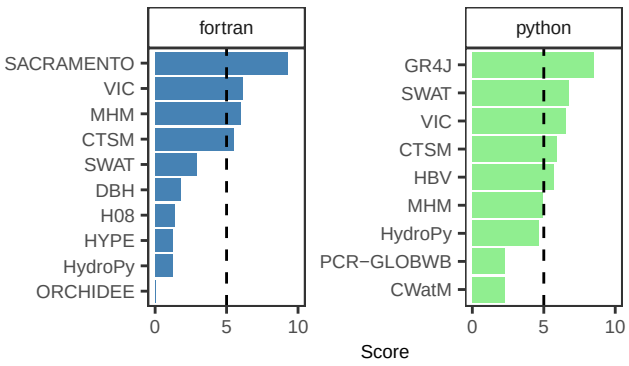
By combining the classic and extended versions of the maintainability index, our analysis reveals differences between Fortran and Python implementations. Using the weighted measure ($M_{\text{LOC}}$), 70% of Fortran code falls into the "low" maintainability category, compared with only 15% when using the unweighted average ($M_{\text{average}}$). This discrepancy indicates that a few complex, poorly maintainable routines dominate the overall profile of the Fortran codebase. In contrast, Python routines present a more favorable profile: 27% achieve high maintainability under $M_{\text{LOC}}$, and all are classified as "highly maintainable" under $M_{\text{average}}$.

```
# PLOT MAINTAINABILITY INDEX ###################################################

plot_maintainability_index <- dt$maintainability_index %>%
  melt(., measure.vars = c("M_loc", "M_average")) %>%
  .[, variable:= factor(variable, levels = c("M_average", "M_loc"))] %>%
  ggplot(., aes(model, value, color = language, shape = type)) +
  geom_point() +
  annotate("rect", xmin = -Inf, xmax = Inf, ymin = -Inf, ymax = 65,
           fill = "red", alpha = 0.18) +
  annotate("rect", xmin = -Inf, xmax = Inf, ymin = 65, ymax = 85,
           fill = "orange", alpha = 0.1) +
  annotate("rect", xmin = -Inf, xmax = Inf, ymin = 85, ymax = Inf,
           fill = "green", alpha = 0.1) +
  facet_wrap(~variable, labeller = as_labeller(c(M_loc = "M[LOC]",
                                                 M_average = "M[average]"),
                                               default = label_parsed)) +
  labs(x = "", y = "Value") +
  scale_color_manual(values = color_languages, guide = "none") +
  theme_AP() +
  theme(legend.position = "none") +
  coord_flip()

plot_maintainability_index
```

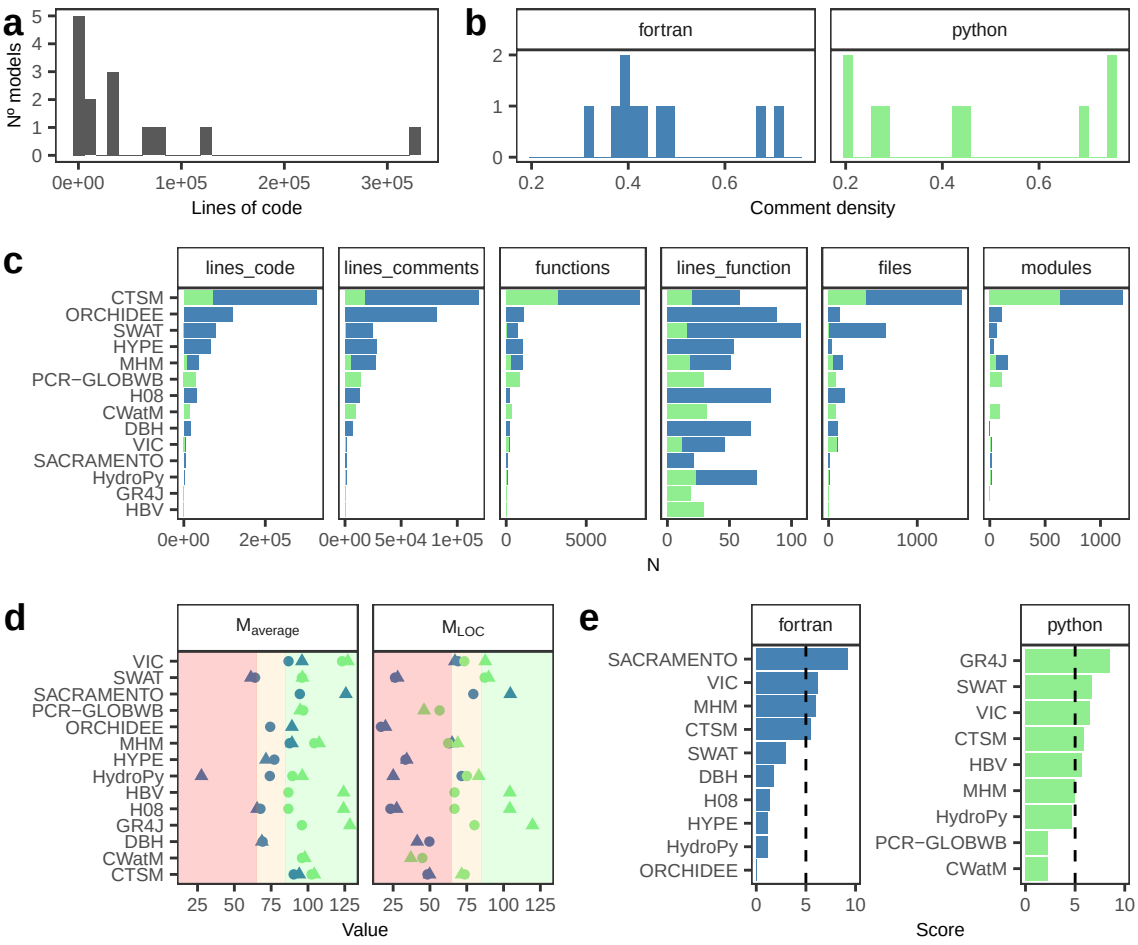



## 2.3 Score

```
# PLOT SCORE ##################################################################

plot_score <- dt$score %>%
  ggplot(aes(x = reorder_within(model, score, language), y = score, fill = language)) +
  geom_bar(stat = "identity") +
  facet_wrap(~ language, scales = "free_y") +
  labs(x = "", y = "Score") +
  scale_fill_manual(values = color_languages) +
  geom_hline(yintercept = 5, lty = 2) +
  scale_x_reordered() +
  scale_y_continuous(limits = c(0, 10), breaks = c(0, 5, 10)) +
  coord_flip() +
  theme_AP() +
  theme(legend.position = "none")

plot_score
```




```
# MERGE PLOTS #################################################################

bottom <- plot_grid(plot_maintainability_index, plot_score, ncol = 2, labels = c("d", "e"))

plot_grid(p1, bottom, ncol = 1, rel_heights = c(0.62, 0.38))
```




## 2.4 Metrics at the function level

```
# METRICS AT THE FILE AND FUNCTION LEVEL ########################################

folder <- "./datasets/results_function"

# Get names of files ------------------------------------------------------------

csv_files <- list.files(path = folder, pattern = "\\.csv$", full.names = TRUE)

# Split into file_metrics and func_metrics --------------------------------------

file_metric_files <- grep("file_metrics", csv_files, value = TRUE)
func_metric_files <- grep("func_metrics", csv_files, value = TRUE)

# Build one named list ----------------------------------------------------------

list_metrics <- list(file_metrics = setNames(lapply(file_metric_files, fread),
                                             basename(file_metric_files)),
                     func_metrics = setNames(lapply(func_metric_files, fread),
```

```
                                                basename(func_metric_files)))

# Create function to combine files ----------------------------------------

make_combined <- function(subset_list, pattern) {
  rbindlist(subset_list[grep(pattern, names(subset_list))], idcol = "source_file")
}

# Combine files -----------------------------------------------------------

metrics_combined <- list(file_fortran = make_combined(list_metrics$file_metrics,  "fortran"),
                         file_python = make_combined(list_metrics$file_metrics,  "python"),
                         func_fortran = make_combined(list_metrics$func_metrics, "fortran"),
                         func_python = make_combined(list_metrics$func_metrics, "python"))

# Functions to extract name of model and language from file ---------------

extract_model <- function(x)
  sub("^(file|func)_metrics_\\d+_([A-Za-z0-9-]+)_(fortran|python).*", "\\2", x)

extract_lang  <- function(x)
  sub("^(file|func)_metrics_\\d+_([A-Za-z0-9-]+)_(fortran|python).*", "\\3", x)

# Extract name of model and language --------------------------------------

metrics_combined <- lapply(metrics_combined, function(dt) {
  dt[, source_file:= sub("\\.csv$", "", basename(source_file))]
  dt[, model:= extract_model(source_file)]
  dt[, language:= extract_lang(source_file)]
  dt
})

# Add column of complexity category ---------------------------------------

metrics_combined <- lapply(names(metrics_combined), function(nm) {
  dt <- as.data.table(metrics_combined[[nm]])
  if (grepl("^func_", nm) && "cyclomatic_complexity" %in% names(dt)) {
    dt[, complexity_category := cut(
      cyclomatic_complexity,
      breaks = c(-Inf, 10, 20, 50, Inf),
      labels = c("b1","b2","b3","b4")
    )]
  }
  dt
}) |> setNames(names(metrics_combined))

# Define labels -----------------------------------------------------------
```

```r
lab_expr <- c(
  b1 = expression(C %in% "(" * 0 * ", 10" * "]"),
  b2 = expression(C %in% "(" * 10 * ", 20" * "]"),
  b3 = expression(C %in% "(" * 20 * ", 50" * "]"),
  b4 = expression(C %in% "(" * 50 * ", " * infinity * ")")
)

# EXPORT DATA TO .CSV ##########################################################

# set output folder inside "datasets" ----------------------------------------

outdir <- file.path("datasets", "merged_results")

# write each slot to its own CSV ---------------------------------------------

lapply(names(metrics_combined), function(nm) {
  out_file <- file.path(outdir, paste0(nm, ".csv"))
  fwrite(metrics_combined[[nm]], out_file)
})
```

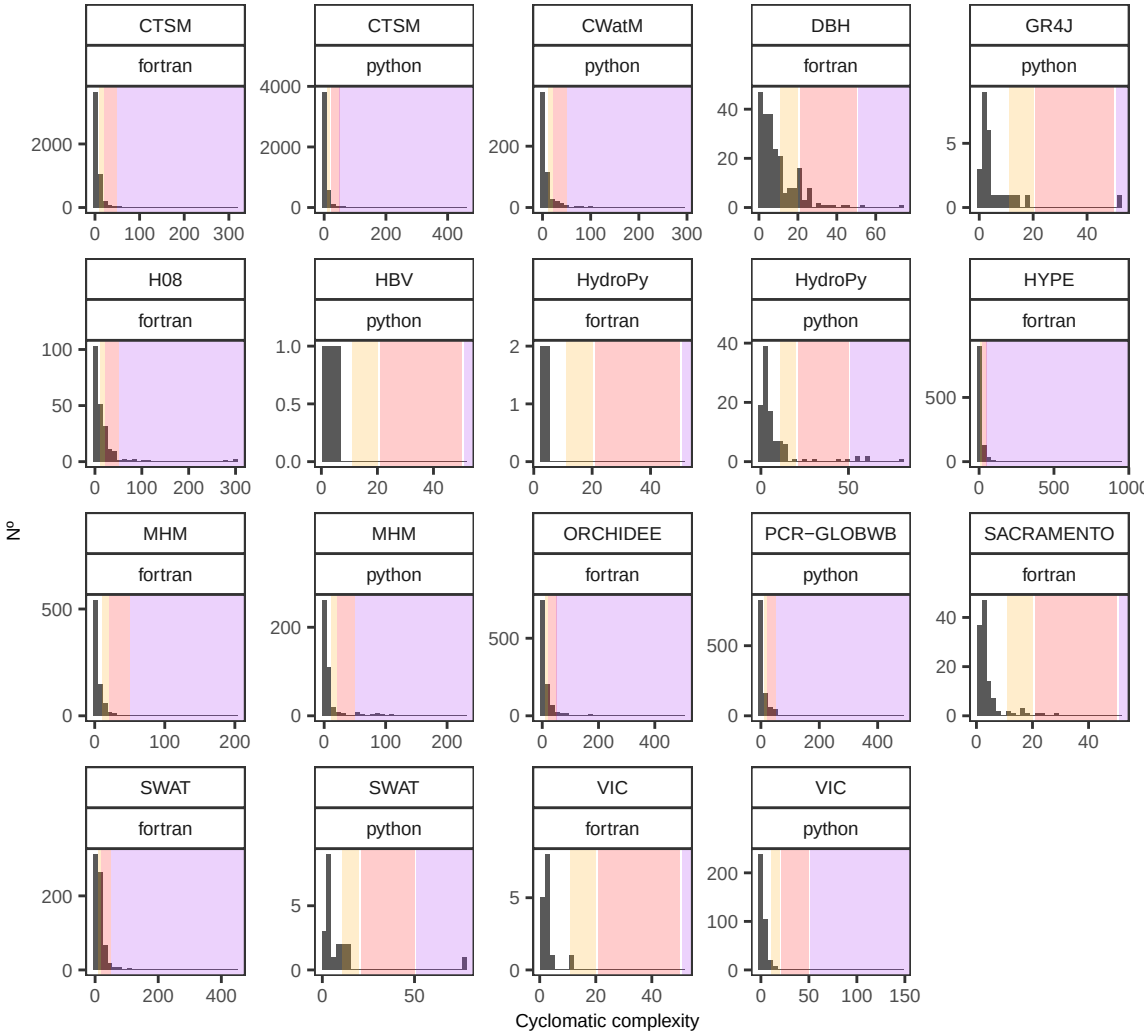
```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
```

```r
# Define vector of interest for cyclomatic complexity

vector_cyclomatic <- c("SUBROUTINE", "FUNCTION")
```

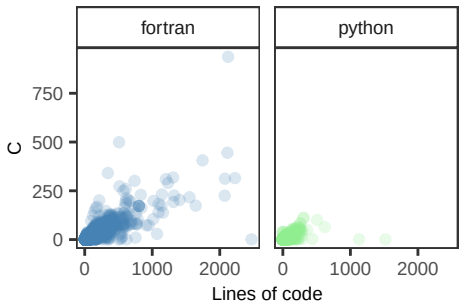
```r
# PLOT ###########################################################################

# Cyclomatic complexity at the model level -----------------------------------

metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x) x[, .(cyclomatic_complexity, model, language)]) %>%
  rbindlist() %>%
  ggplot(., aes(cyclomatic_complexity)) +
  geom_histogram() +
  annotate("rect",
           xmin = 11, xmax = 20,
           ymin = -Inf, ymax = Inf,
```

```
          fill = "orange", alpha = 0.2) +
  annotate("rect",
          xmin = 21, xmax = 50,
          ymin = -Inf, ymax = Inf,
          fill = "red", alpha = 0.2) +
  annotate("rect",
          xmin = 51, xmax = Inf,
          ymin = -Inf, ymax = Inf,
          fill = "purple", alpha = 0.2) +
  facet_wrap(model ~ language, scales = "free") +
  scale_x_continuous(breaks = breaks_pretty(n = 3)) +
  scale_y_continuous(breaks = breaks_pretty(n = 2)) +
  labs(x = "Cyclomatic complexity", y = "Nº") +
  theme_AP()
```

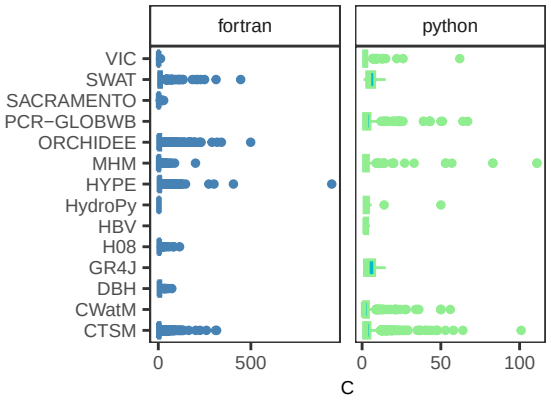
```
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```



```
plot_scatterplot <- metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x)
    x[, .(model, language, `function`, cyclomatic_complexity, loc, bugs, type)]) %>%
  rbindlist() %>%
  .[type %in% vector_cyclomatic] %>%
  ggplot(., aes(cyclomatic_complexity, loc, color = language)) +
  geom_point(alpha = 0.2) +
  scale_color_manual(values = color_languages) +
  coord_flip() +
  scale_y_continuous(breaks = pretty_breaks(n = 3)) +
  facet_wrap(~language) +
  labs(x = "C", y = "Lines of code") +
  theme_AP() +
  theme(legend.position = "none")

plot_scatterplot
```

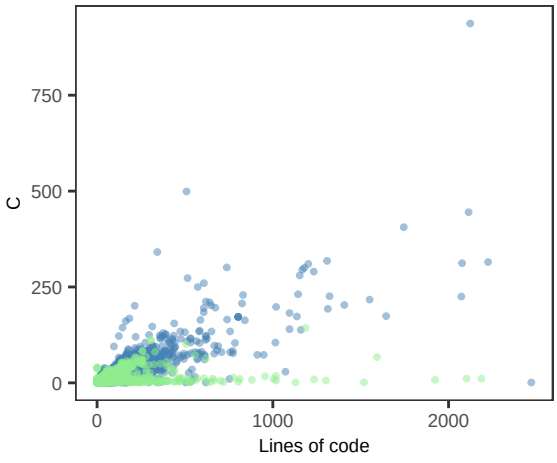



```
plot_c_model <- metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x)
    x[, .(model, language, `function`, cyclomatic_complexity, loc, bugs, type)]) %>%
  rbindlist() %>%
  .[type %in% vector_cyclomatic] %>%
  ggplot(., aes(model, cyclomatic_complexity, fill = language, color = language)) +
  geom_boxplot(outlier.size = 1) +
  coord_flip() +
  scale_y_continuous(breaks = scales::breaks_pretty(n = 2)) +
  facet_wrap(~language, scales = "free_x") +
  labs(x = "", y = "C") +
  theme_AP() +
  scale_color_manual(values = color_languages) +
  theme(legend.position = "none")

plot_c_model
```



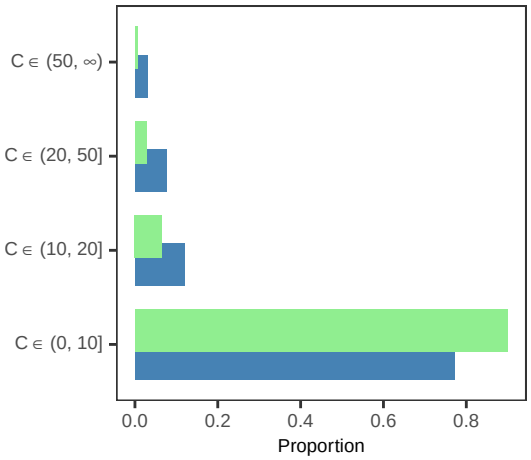

```
# Scatterplot cyclomatic vs lines of code ------------------------------------

plot_c_vs_loc <- metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x) x[, .(loc, cyclomatic_complexity, language)]) %>%
  rbindlist() %>%
  ggplot(., aes(loc, cyclomatic_complexity, color = language)) +
  geom_point(alpha = 0.5, size = 0.7) +
  scale_x_continuous(breaks = breaks_pretty(n = 3)) +
  labs(x = "Lines of code", y = "C") +
  scale_color_manual(values = color_languages) +
  theme_AP() +
  theme(legend.position = "none")

plot_c_vs_loc
```

```
## Warning: Removed 1195 rows containing missing values or values outside the scale range
## (`geom_point()`).
```




```
# Count & proportion ---------------------------------------------------------

plot_bar_cyclomatic <- metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x) x[, .(complexity_category, language, type)]) %>%
  rbindlist() %>%
```

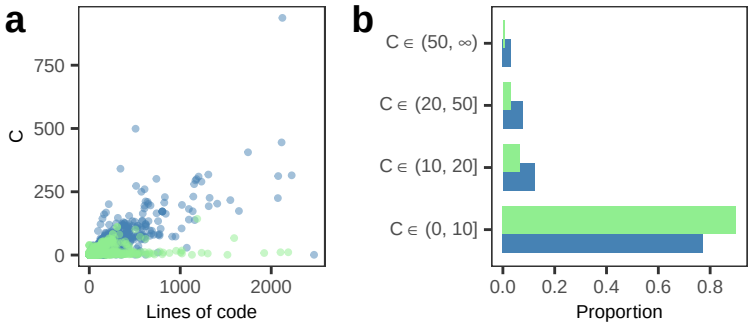
```
  .[type %in% vector_cyclomatic] %>%
  .[, .N, .(complexity_category, language)] %>%
  .[, proportion := N / sum(N), language] %>%
  ggplot(., aes(complexity_category, proportion, fill = language)) +
  geom_bar(stat = "identity", position = position_dodge(0.6)) +
  scale_fill_manual(values = color_languages) +
  scale_y_continuous(breaks = scales::breaks_pretty(n = 4)) +
  scale_x_discrete(labels = lab_expr) +
  labs(x = "", y = "Proportion") +
  coord_flip() +
  theme_AP() +
  theme(legend.position = "none")

plot_bar_cyclomatic
```



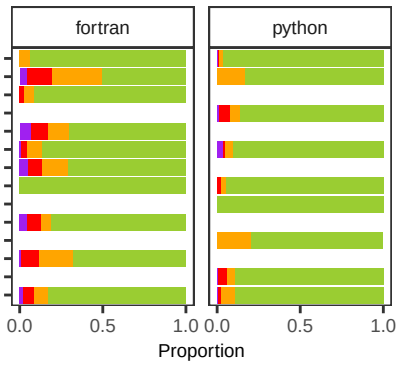

```
# MERGE ########################################################################


 plot_cyclomatic <- plot_grid(plot_c_vs_loc, plot_bar_cyclomatic, ncol = 2, labels = "auto",
          rel_widths = c(0.45, 0.55))
```

```
## Warning: Removed 1195 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

```
plot_cyclomatic
```



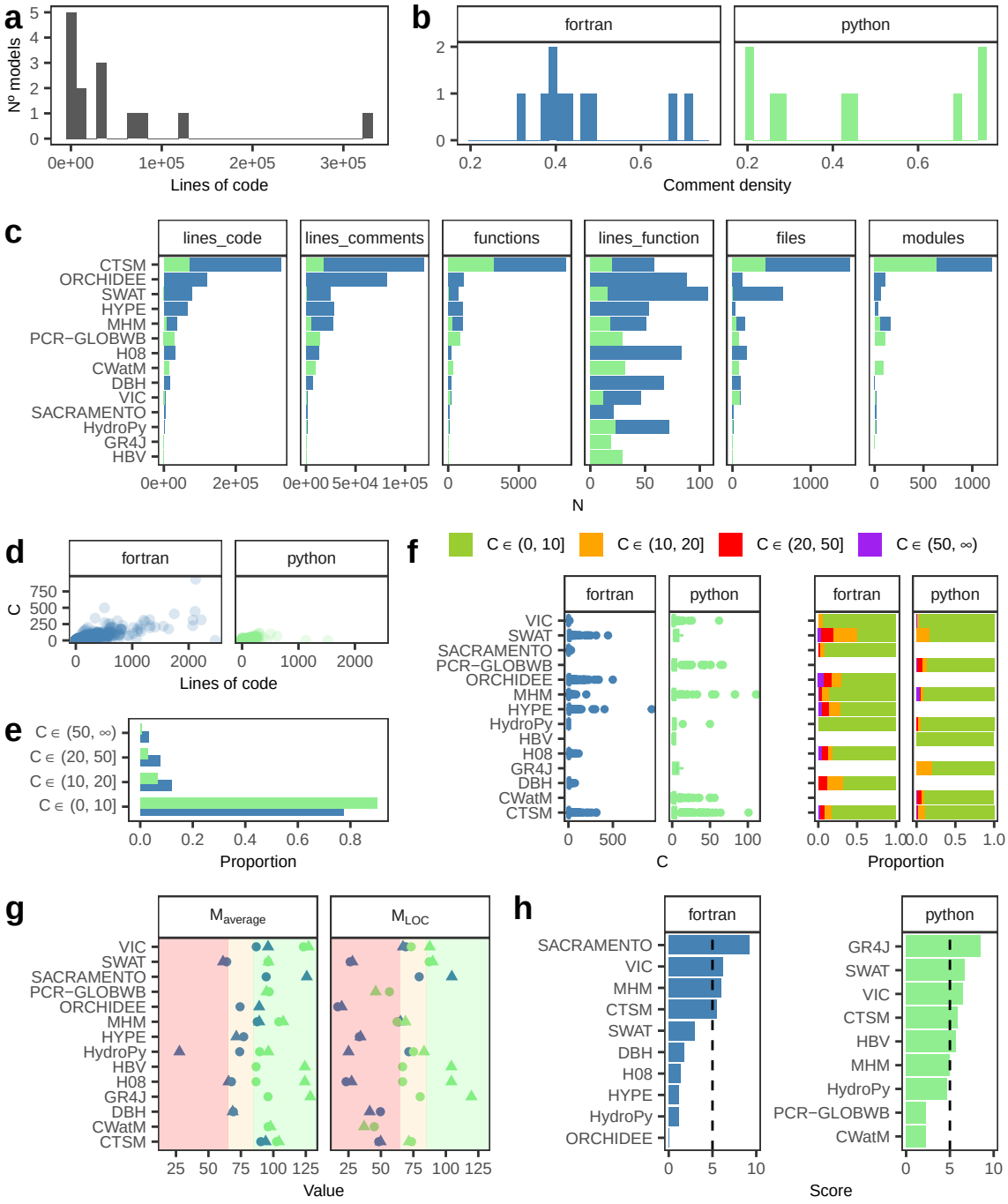
```
plot_bar_category <- metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x)
    x[, .(model, language, complexity_category, type)]) %>%
  rbindlist() %>%
  .[type %in% vector_cyclomatic] %>%
  .[, .N, .(model, language, complexity_category)] %>%
  .[, proportion := N / sum(N), .(language, model)] %>%
  ggplot(., aes(model, proportion, fill = complexity_category)) +
  geom_bar(stat = "identity") +
  scale_fill_manual(values = c("yellowgreen", "orange", "red", "purple"),
                    labels = lab_expr,
                    name = "") +
  facet_wrap(~language) +
  labs(x = "", y = "Proportion") +
  coord_flip() +
  scale_y_continuous(breaks = scales::breaks_pretty(n = 3)) +
  theme_AP() +
  theme(legend.position = "none") +
  theme(axis.text.y = element_blank(),
        legend.text = element_text(size = 7))

plot_bar_category
```




```
di <- plot_grid(plot_scatterplot, plot_bar_cyclomatic, ncol = 1, labels = c("d", "e"))
legend <- get_legend_fun(plot_bar_category + theme(legend.position = "top"))
```

```
## Warning: `is.ggplot()` was deprecated in ggplot2 3.5.2.
## i Please use `is_ggplot()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
dada <- plot_grid(plot_c_model, plot_bar_category, ncol = 2, rel_widths = c(0.61, 0.39))
dada2 <- plot_grid(legend, dada, ncol = 1, rel_heights = c(0.1, 0.9), labels = "f")
dada3 <- plot_grid(di, dada2, ncol = 2, rel_widths = c(0.4, 0.6))
dada4 <- plot_grid(plot_maintainability_index, plot_score, ncol = 2, labels = c("g", "h"))
dada5 <- plot_grid(p1, dada3, ncol = 1, rel_heights = c(0.6, 0.4))
plot_grid(dada5, dada4, rel_heights = c(0.73, 0.27), ncol = 1)
```




# 3 Risky paths

```
# NETWORK CITATION ANALYSIS ----------------------------------------------------

# Path to folder ---------------------------------------------------------------

path <- "./datasets/call_metrics"
```

```r
# List CSV files ------------------------------------------------------------

files <- list.files(path, pattern = "\\.csv$", full.names = TRUE)

# Split by language ---------------------------------------------------------

python_files  <- grep("python",  files, value = TRUE, ignore.case = TRUE)
fortran_files <- grep("fortran", files, value = TRUE, ignore.case = TRUE)

base_fortran <- file_path_sans_ext(basename(fortran_files))
base_python <- file_path_sans_ext(basename(python_files))

model_names_fortran <- models <- sub(".*_", "", base_fortran)
model_names_python <- models <- sub(".*_", "", base_python)

# Load and name files -------------------------------------------------------

python_list  <- lapply(python_files, fread)
fortran_list <- lapply(fortran_files, fread)

names(python_list) <- model_names_python
names(fortran_list) <- model_names_fortran

# RBIND ---------------------------------------------------------------------

make_callgraph <- function(lst, lang) {
  rbindlist(lst, idcol = "model") %>%
    .[, language := lang] %>%
    .[, .(model, language, `function`, call)] %>%
    setnames(., c("function", "call"), c("from", "to"))
}

python_callgraphs  <- make_callgraph(python_list,  "python")
fortran_callgraphs <- make_callgraph(fortran_list, "fortran")

all_callgraphs <- rbind(python_callgraphs, fortran_callgraphs)

# CREATE THE NETWORK ##########################################################

# Define the weights to characterize risky nodes ----------------------------

alpha <- 0.6  # Weight to cyclomatic complexity
beta  <- 0.3  # Weight to in-degree (impact of bug upstream)
gamma <- 0.1  # Weight to betweenness (critical bridge)

# Ensure unique names complexity dataset ------------------------------------
```

```r
cc_unique <- metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x)
    x[, .(model, language, `function`, cyclomatic_complexity, complexity_category, type)]) %>%
  rbindlist() %>%
  setnames(., "function", "name") %>%
  arrange(desc(cyclomatic_complexity)) %>%
  distinct(name, .keep_all = TRUE)

# CREATE NETWORKS FROM CALL GRAPHS ###############################################

all_graphs <- all_callgraphs[, .(graph = list(as_tbl_graph(.SD, directed = TRUE))), .(model, la
  .[, graph:= Map(function(g, m, lang) {

  comp_sub <- cc_unique[model == m & language == lang]

  g %>%
    activate(nodes) %>%

    # Left join with dataset with cyclomatic complexity values -----------------

  left_join(comp_sub, by = "name") %>%

    # Remove Python MODULE_AGG / CLASS_AGG nodes from this graph
    # because they are not callable -------------------------------------------

  filter(!(language == "python" & type %in% c("MODULE_AGG", "CLASS_AGG"))) %>%

    # Calculation of key network metrics ---------------------------------------

    mutate(indeg = centrality_degree(mode = "in"),
           outdeg = centrality_degree(mode = "out"),
           btw = centrality_betweenness(directed = TRUE, weights = NULL),
           cyclo_scaled = rescale(cyclomatic_complexity),
           indeg_scaled = rescale(indeg),
           btw_scaled = rescale(btw),
           risk_node = alpha * cyclo_scaled + beta * indeg_scaled + gamma * btw_scaled
    )},

    graph, model, language
  )
]
```

```
## Warning: There were 2 warnings in `mutate()`.
## The first warning was:
## i In argument: `cyclo_scaled = rescale(cyclomatic_complexity)`.
## Caused by warning in `min()`:
## ! no non-missing arguments to min; returning Inf
```

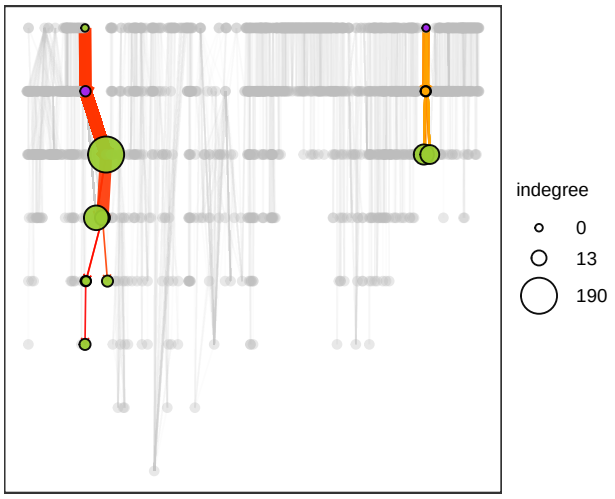
```
## i Run `dplyr::last_dplyr_warnings()` to see the 1 remaining warning.
```

```
# COMPUTE ALL PATHS AND THEIR RISK SCORES ###################################

all_graphs <- all_graphs[, paths_tbl:= lapply(graph, all_paths_fun)]
```

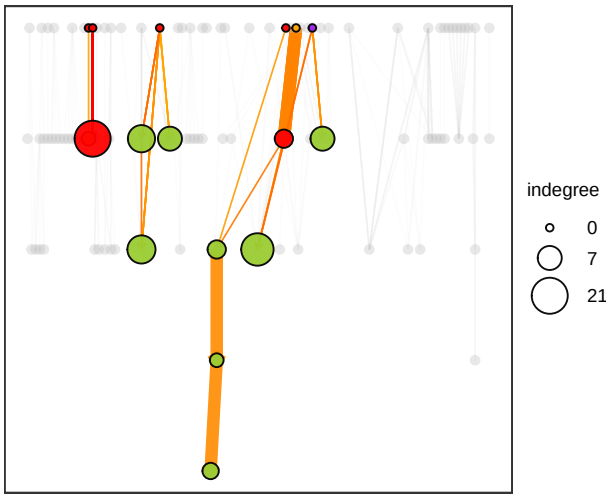
```
## Warning: There was 1 warning in `mutate()`.
## i In argument: `max_node_risk = map_dbl(...)`.
## Caused by warning in `max()`:
## ! no non-missing arguments to max; returning -Inf
## There was 1 warning in `mutate()`.
## i In argument: `max_node_risk = map_dbl(...)`.
## Caused by warning in `max()`:
## ! no non-missing arguments to max; returning -Inf
```

```
# PLOT ALL CALLGRAPHS ########################################################

all_graphs <- all_graphs[, plot_obj:= mapply(plot_top_risky_paths_fun, call_g = graph,
                                             paths_tbl = paths_tbl, model.name = model,
                                             language = language, SIMPLIFY = FALSE)]
```

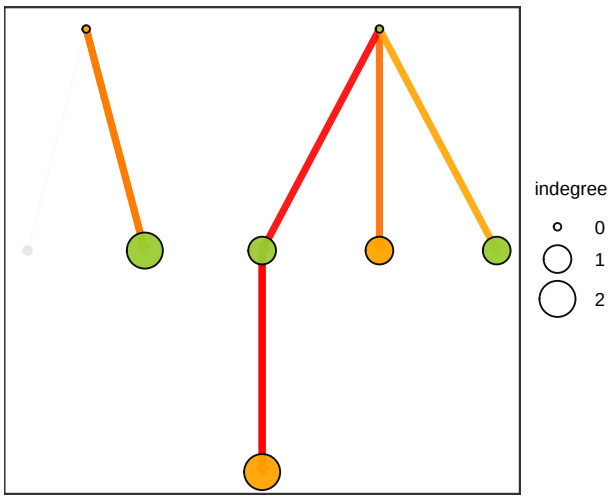
CTSM: python




CWatM: python

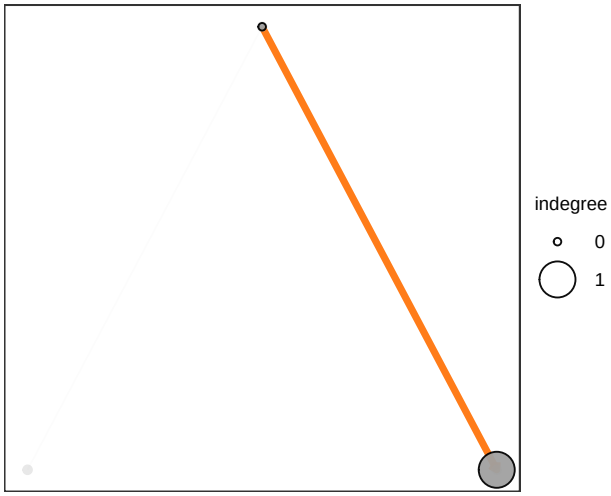


GR4J: python

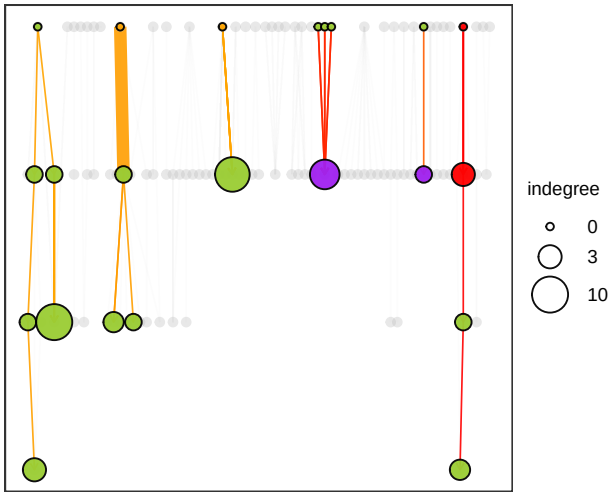



```
## No paths in paths_tbl; skipping plot for: HBV
```

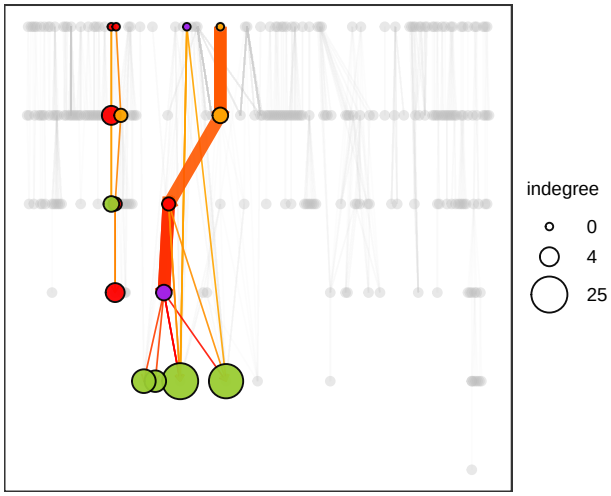
HydroPy: python



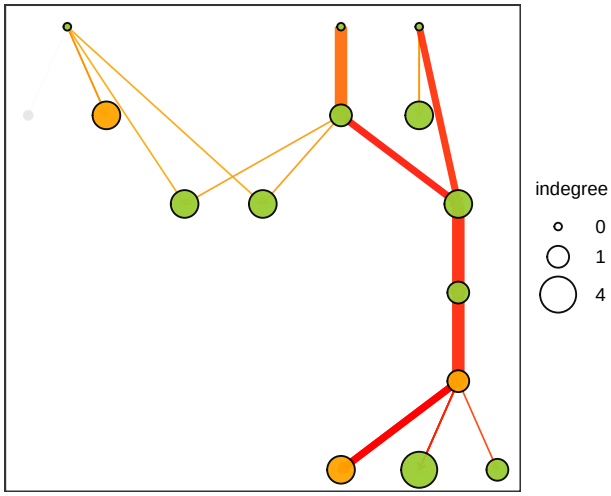

MHM: python

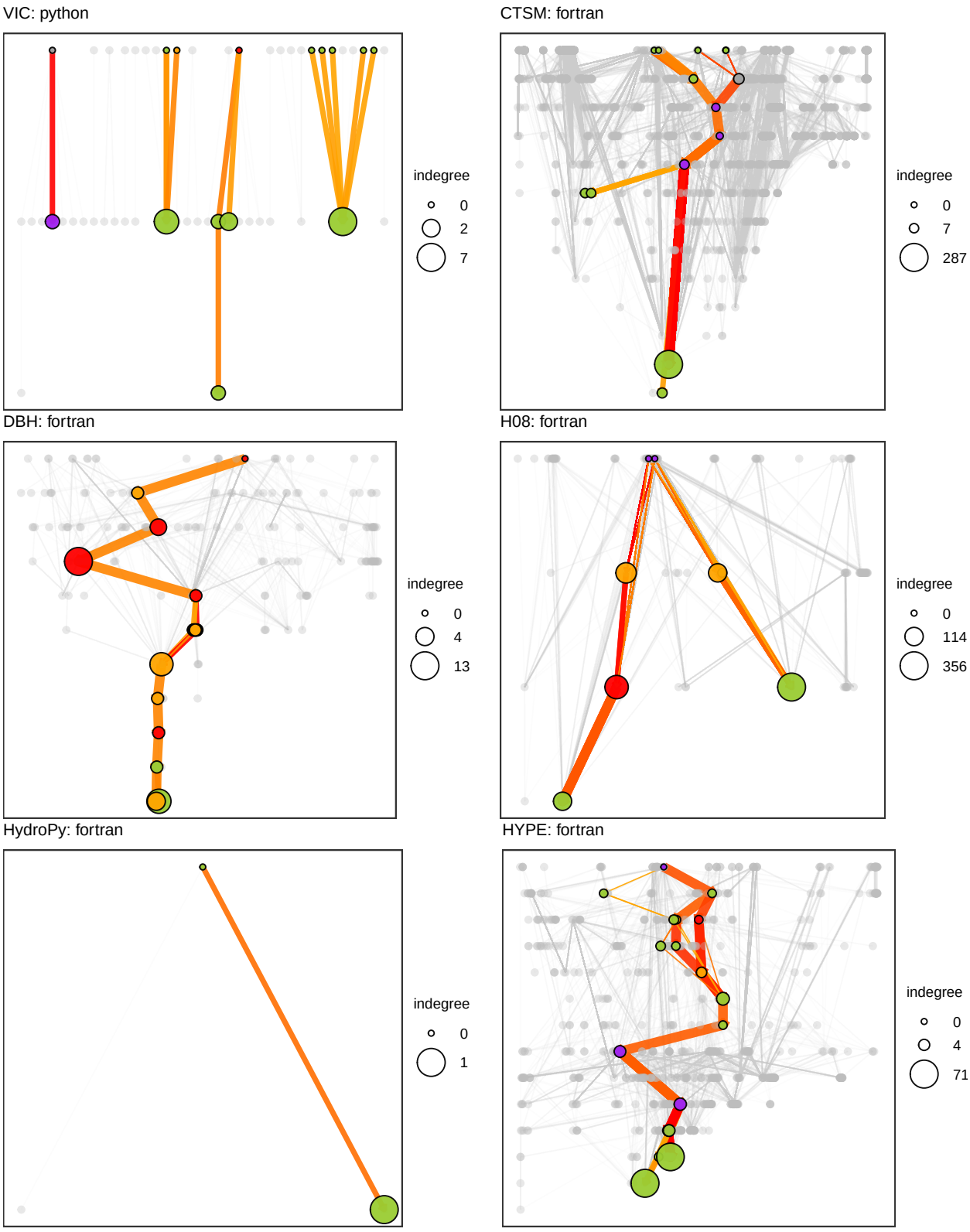



PCR−GLOBWB: python

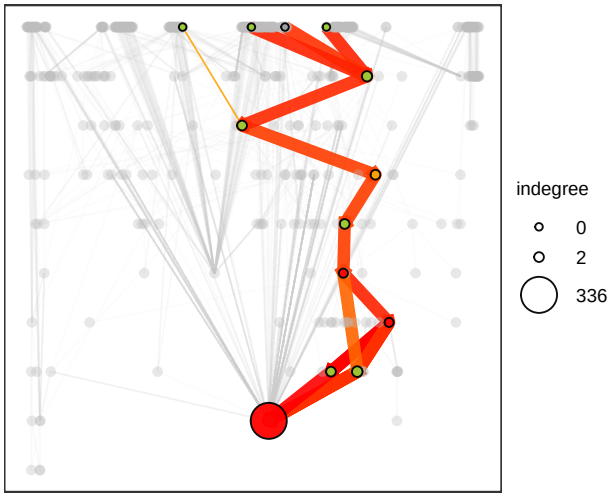



SWAT: python

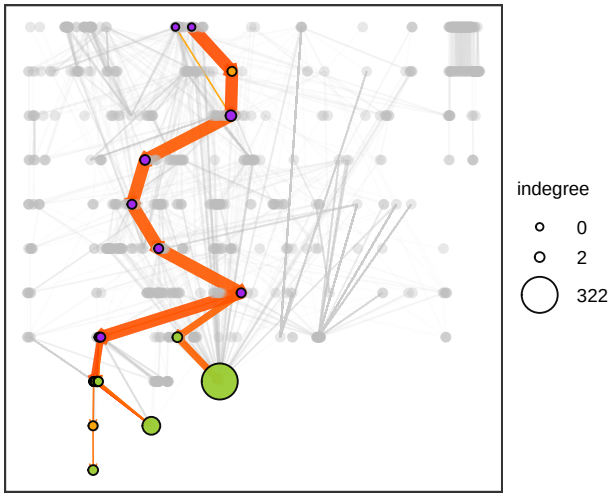



VIC: python
indegree
0
2
7
CTSM: fortran
indegree
0
7
287
DBH: fortran
indegree
0
4
13
H08: fortran
indegree
0
114
356
HydroPy: fortran
indegree
0
1
HYPE: fortran
indegree
0
4
71

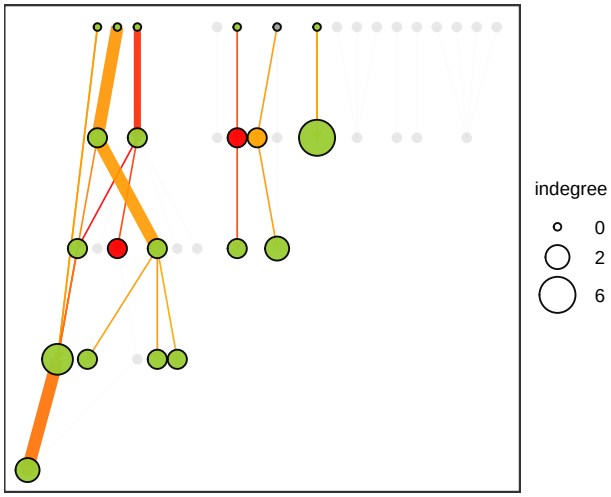
MHM: fortran



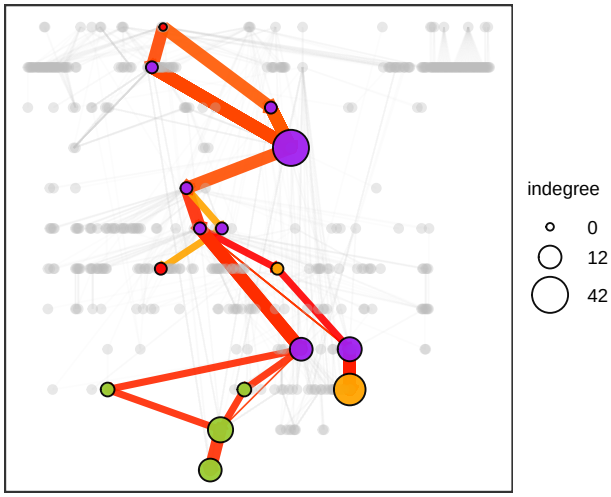

ORCHIDEE: fortran




SACRAMENTO: fortran




SWAT: fortran




VIC: fortran