Data Visualiziation with C
Session 3

Robert Palmere

2021

Topics:

- Introduce the DISLIN library
- Review of C syntax

We already saw the capabilities of Python for visualizing data using *matplotlib*, and *seaborn* libraries. In this session, we look at how we can implement similar visualization using the DISLIN library (https://www.mps.mpg.de/dislin/) using the C programming language. There are a few benefits to using the DISLIN library within the context of C which include increased speeds of computation, easy access of DISLIN routines, and integration of DISLIN visualization into larger C programs. The 'readme' file outlines the purpose of each file within the session directory and provides details on how to link the DISLIN library on Mac. The DISLIN readme has information on linking for other operating systems as well. The structure of DISLIN plots follow a set of "levels" which determines the behavior of routines. A program implementing the DISLIN library should follow a series of steps:

- 1. Setting the page format, file format, and file name
- 2. Initialization of DISLIN
- 3. Setting plot parameters
- 4. Plotting the axis system
- 5. Plotting data points

First let's look at how we can set up a DISLIN window for visualize our plots.

```
#include <stdio.h>
#include <math.h>
#include "dislin.h"
```

```
int main (){

   window(0, 0, 1200, 800); // X11 Window spawns at (0, 0) top
       left of screen with size 1200x800 pixels
   metafl ("cons");       // Metafile information goes to console
   scrmod ("revers");     // Set background color to white and
       foreground to black
   disini ();             // Initialize DISLIN with default plot
       parameters
   pagera ();             // Plots border around the page
   complx ();             // Sets font to 'Complex'
   errmod ("ALL", "OFF"); // Turn off warnings and error output
       from disfin();
   disfin ();             // Terminate DISLIN
   return 0;
}
```

To remain organized, we can place formatting, parameter setting, and plotting into separate functions. Let's try this with a random set of data.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "dislin.h"

// DISLIN formatting (global parameters)
void format(void){
   window(0, 0, 1200, 800);
   metafl("cons");
   scrmod("revers");
}

// Set page parameters for X11 display
void set_params(void){
   pagera();
   complx();
}

// Place the axis systems
void set_plot(void){

   graf(0.0, 100, 0.0, 100, 0, 26, 0, 25); // Does heavy lifting
       of placing x,y axis on the page

}

// Generate and Plot data on axis systems
void place_data(void){
```

```
   float x[100];
   float y1[100], y2[100];

   for (int i = 0; i < 100; i++){
      x[i] = i;
      y1[i] = ((float) rand() / (RAND_MAX)) + 15;
      y2[i] = ((float) rand() / (RAND_MAX)) + 15;
   }

   linwid(10); // Set the linewidth for lines
   curve(x, y1, 100); // Plot x, y data using 100 points
   color("red");
   curve(x, y2, 100);
}

int main ()
{
   format();
   disini();
   set_params();
   set_plot();
   place_data();
   errmod("ALL", "OFF");
   disfin ();
  return 0;
}
```

Just as we did in the Python session, we can also display these data as a histogram. The code is a bit longer, however, since the histograms are calculated using our own function definitions.

```
...
float min(const float* array, int size){
   float min_;
   for (int i = 0; i < size; i++){
   if (i == 0){
      min_ = array[i];
   }
   if (array[i] < min_){
      min_ = array[i];
      }
   }
   return min_;
}

float max(const float* array, int size){
   float max_ = 0;
```

```
   for (int i = 0; i < size; i++){
      if (i == 0){
         max_ = array[i];
      }
      if (array[i] > max_){
         max_ = array[i];
      }
   }
   return max_;
}

float* histogram(const float* ys, int size, int bins){

   float sum = 0;
   int bin_size;

   // Check bin size
   if (bins <= size){
      bin_size = (size)/(bins);
   }
   else{
      printf("Error: too many bins for array of size %d.", size);
      exit(1);
   }

   // Set bin values
   float xb[bin_size];
   float step = (max(ys, 100) - min(ys, 100))/bin_size;
   for (int i = 0; i < bin_size; i++){
      if (i > 0){
         xb[i] = xb[i-1] + step;
      }
      else{
         xb[i] = min(ys, 100);
      }
   }

   float* xh = (float*)malloc(sizeof(float)*bin_size); // bin
       heights

   for (int i = 0; i < bin_size; i++){
      int sum_ = 0;
      for (int j = 0; j < size; j++){
         if (i > 0){
            if (ys[j] <= (xb[i]) && ys[j] >= (xb[i-1])){
               sum_ += 1;
            }

         }
      }
```

```
      xh[i] = sum_;
   }

   return xh;
}

void place_data(void){

   float x[100];
   float y1[100], y2[100];

   for (int i = 0; i < 100; i++){
      x[i] = i;
      y1[i] = ((float) rand() / (RAND_MAX)) + 15;
      y2[i] = ((float) rand() / (RAND_MAX)) + 15;
   }

   // Calculate histogram for plotting
   int bins = 10;
   float* h1 = histogram(y1, 100, bins);
   float* h2 = histogram(y2, 100, bins);

   float hx[bins];
   for (int i = 0; i < bins; i++){
      hx[i] = i+1;
   }

   // Bar plot of the data
   float xhb[bins];
   for (int i = 0; i < bins; i++){
      xhb[i] = 0;
   }

   bars(hx, xhb, h1, bins);
   bars(hx, xhb, h2, bins);

   free(h1);
   free(h2);

}
```

We generate a line histogram in 5-histline.c. DISLIN has its own convience though in that interpolation is easier to implement than even Python. For instance, to implement a spline interpolation of the data we only need to call a single routine *polcrv()* which will apply this interpolation to the plotted data sets for us as shown in 6-interp.c. Animation of plots was also one of the less intuitive features of Python where *FuncAnimation* did its own iterations if a generator was not used. If we want to animate the display of multiple line plots

we can do so easily in DISLIN by calling *curve()* in a loop. The important part here is that, although DISLIN will place the curve, the display will not be update unless we call *sendbf()* to send the buffer to the X11 display window.

```
int main ()
{
   format();
   disini();
   set_params();
   set_plot();

   int frames = 0;
   float x[100];
   float y1[100], y2[100];

   while (frames != 100){
      frames++;

      for (int i = 0; i < 100; i++){
         x[i] = i;
         y1[i] = rand() % 50;
         y2[i] = rand() % 50;
      }

      curve(x, y1, 100);
      sendbf();
   }
   endgrf();
   set_params();
   set_plot();
   setrgb(0, 0, 0);
   curve(x, y1, 100);
   errmod("ALL", "OFF");
   disfin ();
  return 0;
}
```

In the above animation, we continuously add curves to the plot without removing the previous line. One solution for this in DISLIN is to set the previous curve color to that of the background color before plotting the next curve.

```
void set_data(float* x, float* y){
   for (int i = 0; i < 100; i++){
      float itx = i;
      float ity = rand() % 50;
      *x = itx;
      *y = ity;
```

```
      x++;
      y++;
   }
}

void place_data(float* x, float* y, char* onoff){

   if (strcmp(onoff, "ON") == 0){
      color("WHITE");
   }

   if (strcmp(onoff, "OFF") == 0){
      color("BLACK");
   }

   curve(x, y, 100);
}

int main ()
{
   format();
   disini();

   int frames = 0;
   float x[100];
   float y1[100];
   set_plot();
   set_params();
   while (frames != 100){
      frames++;
      set_data(&x[0], &y1[0]);
      place_data(x, y1, "ON");
      sendbf();      // Send updated buffer to the screen
      place_data(x, y1, "OFF");
   }
   place_data(x, y1, "ON");
   sendbf();
   errmod("ALL", "OFF");
   disfin ();
  return 0;
}
```

We have adjusted the function for placing data to adjust the color of the line while iterating through the frames of the simulation. Additionally, we modify the setting of data so that each time set_data is called the pointers to x and y data values are also iterated. As we can see, the code is rather lengthy for simple plots. Fortunately, quick plots are also available in DISLIN in the case that we want to visualize data without setup and configuration.

```
int main(){

   int size = 100;
   float x[size];
   float y[size];
   float z[size];

   for (int i = 0; i < size; i++){
      x[i] = rand() % 50;
      y[i] = rand() % 50;
   }

   // Scatter plot
   window(0, 0, 800, 800);
   scrmod("revers");  // White foreground
   qplsca(x, y, size); // Quick scatter plot

   return 0;
}
```