Al-Jabr

GUI application for piecewise linear and parabolic spline interpolation

Fazliddin Juraev

M. V. Lomonosov Moscow State University

fazliddin.uzbek@yahoo.com

© 2009 Fazliddin Juraev

All rights reserved. This document is redistributable under the license given in the file "COPYRIGHT"

distributed in the Al-Jabr archive.

Content

Introduction

Project is symbolically called ***Al-Jabr*** after the name of the popular Uzbek mathematician's algebra book. Project's main purpose is to develop cross-platform software that approximately computes certain functions values on given interval which is very expensive to evaluate by analytic manner, and draw 2D and 3D graphics of the function.

Al-Jabr is a user-friendly and customizable program that enables you build graphics of functions.

Problem statement

- Piecewise linear interpolation

if values of function $f(x)$ is known at a set of points $x_1<x_2<\ldots<x_n$, but not known analytically for all x , then it is often desirable for the intermediate values to be approximated. This estimate of $f(x)$ for arbitrary x is called interpolation. We speak of extrapolation if estimates for values outside the given domain are required, i.e. $x<x_1$ or $x>x_n$.

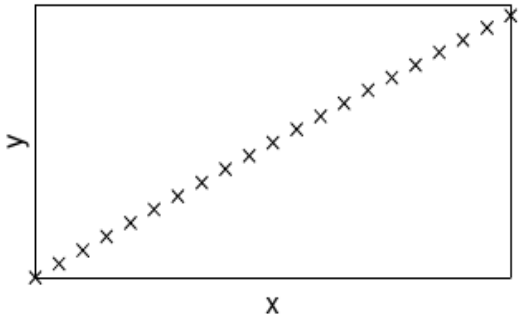


Figure 1.

In practice the interpolation of a set of measured data $\{x_i, y_i\}$ can be difficult when the general behavior of the function is unknown. it is therefore necessary to find an appropriate family of functions for every interpolation problem. For example the data given in figure 1 suggests a linear interpolation.

Interpolation is closely related to approximation. This is problem of finding a replacement function for a given one. If a function is expensive to evaluate then it may be preferable if it can be replaced by an approximation which shows similar behavior but is easier to compute.

To solve the problem we should find a set of linear functions L_1 that are equal to f(x) in given points: $L_1(x_i)=f(x_i), x_i \subset [a,b], i=1..n$. To find L_1 we use so called piecewise linear interpolation method. Connecting adjacent given points we get a set cuts that best interpolate f(x) in the interval (Figure 2).

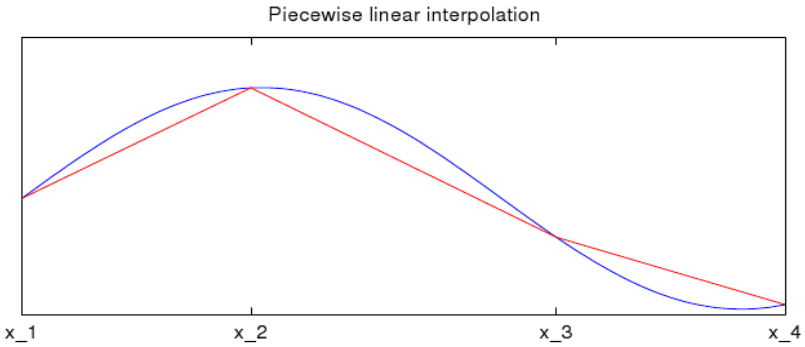


Figure 2.

As shown there appeared three cuts connecting x_1 with x_2, x_2 with x_3, x_3 with x_4. If two points $(x_{i-1}, f_{i-1}), (x_i, f_i)$ are given, it is possible find linear function that connects them by the following formula:

$$y = a_i x + b_i \qquad (1)$$

$$a_i = \frac{(f_i - f_{i-1})}{(x_i - x_{i-1})},\ b_i = f_{i-1} - a_i x_{i-1},\ i = 1..n$$

So for arbitrary point x in the interval, $f(x)$'s approximate value can be computed by (1), if we find i index where $x_{i-1} < x < x_i$ is relevant.

- Parabolic spline interpolation

Problem is same. But we use an another method which interpolates more precisely. To build interpolating function F(x), we look for parabolas in segments which get closer to our function. For given

$$a = x_1 < x_2 < \ldots < x_n = b$$

sequence of numbers it is calculated new

$$\xi_1 < a = x_1 < \xi_2 < x_2 < \xi_3 < x_3 < \ldots < x_{n-1} < \xi_n < x_n = b < \xi_{n+1}.$$

where

$\xi_1 = x_0$,
$\xi_{n+1} = x_{n+1}$,
$\xi_i = \frac{(x_i + x_{n+1})}{2}, i = 1 \ldots n$

x_0, x_{n+1} and $f(x_0)$, $f(x_{n+1})$ are additionally given by user.
Next step is to find parabolas in following manner:

$$\left.\begin{array}{lll} P_i(x_i) & = f(x_i), & \\ P_i(\xi_i) & = v_i, \qquad & P_i(\xi_{i+1}) \quad = v_{i+1} \end{array}\right\} \quad i = 1, \ldots, n$$

Where P_i is a parabola that is determined in $[\xi_i, \xi_{i+1}]$, v_i are some parameters. These parameters are found by solving tree-diagnal matrix using Gauss method. After v_i have been found, we can evaluate proper P_i coefficients in following manner:

$$\begin{aligned} a_{1,i} &= v_i \\ a_{2,i} &= \frac{f(x_i) - v_i}{x_i - \xi_i} \\ a_{3,i} &= \frac{1}{\xi_{i+1} - \xi_i} \left(\frac{v_{i+1} - f(x_i)}{\xi_{i+1} - x_i} - \frac{f(x_i) - v_i}{x_i - \xi_i} \right) \end{aligned}$$

Our parabolas have following view:

$$P_i(x) = a_{1,i} + a_{2,i}(x - \xi_i) + a_{3,i}(x - \xi_i)(x - x_i)$$

Application manual

This program is open source and available for running under Linux. To run it, you should have Qt4 installed on you machine.

File that contains function points are given by command line. The first line in the file must be number of points. Pair of values x and y must be given in single line (Figure 3). If no file is given program uses one of its default functions.

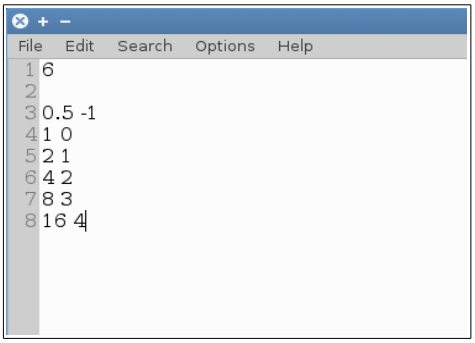


Figure 3

Al-Jabr supports some command line options.

SYNOPSIS

```
aljabr [Qt-options] [options] [filename]
```

Qt options:

`-style [windows | motif | plastique | cde]`

Specifies main window style. By default your system's style is used.

Basic options:

`-h, --help`

Show help about options.

`--help-qt`

Show Qt specific options.

`-v` Prints version

`--no-gui`

Run in comman line mode . Don't load GUI.

`-n, --number-of-points point_num`

Number of the points in the interval. This option has less affect, if you give input file. Because in this case `point_num` argument will be changed to the number of really read points.

`-a, --from interval_a`

Beginning a value of the interval [a,b]. If this option is not

given this argument is set to x_1.

`-b, --to` interval_b
Ending b value of the interval [a,b]. If this option is not given this argument is set to x_n.

`-t, --func-type` [cos | sin | quad | log]
Specify function name to be interpolated. it is useful If you don't want interpolation have been built from input file. Al-Jabr has four built-in functions that can be specified by this option. By default, it is quad: $f(x) = x^2 - 2x + 1$. Except default functions you can specify your own functions for interpolation. See `-d/--plugin` option.
Option `-t` conflicts with option `-f`.

`-d, --plugin` dynamically loadable library path
Load external shared object. If this option is given function will be used from plugin. Al-Jabr is extensible, that is, it supports plugins. You can download or create your own plugins. Plugin is shared object file in which any function can be implemented and given to the program by command line. The function name which is called from the program must be given by option `-t/--func-type`.
Option `-d` conflicts with option `-f`.

`-f` filename
Input file name from where points are read. If this option is not given default function will be used. This option conflicts with options `-t,--func-type` and `-d,--plugin`.

`-p, --parabolic`
Use parabolic spline method. Conflicts with option `-l,--linear`

`-l, --linear`
Use linear spline method. Conflicts with option `-p,--parabolic`.

GUI reference:

File → Take photo

Saves current image of graph in PNG format

View → In
Zooms in current image of graph

View → Out
Zooms out current image of graph

View → Show Function
Shows/Hides f(x) function graph

View → Show Error
Shows/Hides |F(x) – f(x)| error function

View → Show Points
Shows/Hides points of interpolation function

View → Effects
Enables/Disables 2D graph widget's effects

Go → Up
Moves up current image of graph

Go → Down
Moves down current image of graph

Go → Left
Moves left current image of graph

Go → Right
Moves right current image of graph

Options → Function color
Selects custom color for drawing f(x) function

Options → Interpolation color
Selects custom color for drawing F(x) interpolation function

Options → Error color
Selects custom color for drawing |F(x) – f(x)| error function

Options → Pen Width

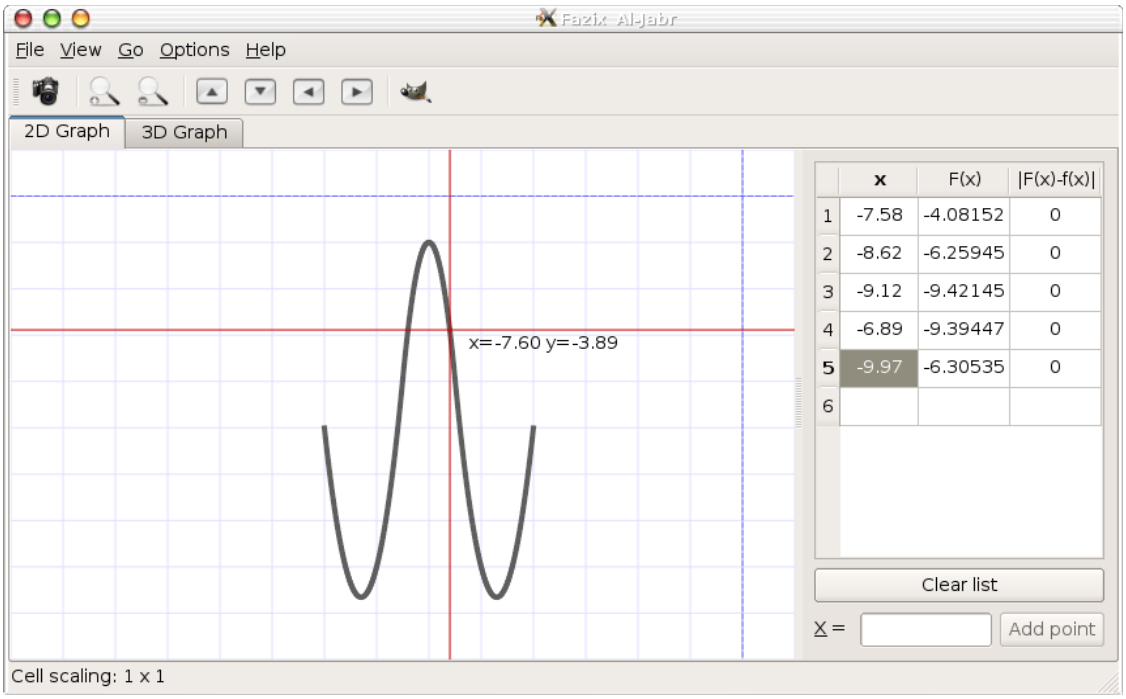
Selects custom line width for drawing functions

Help → About

Shows message about Al-Jabr

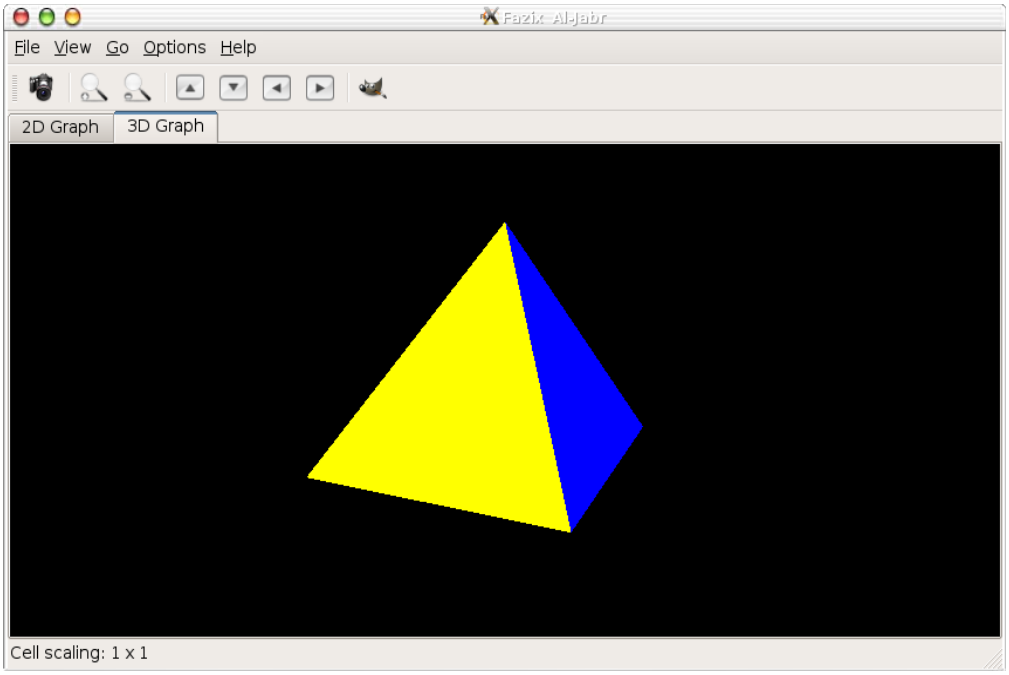
Help → About Qt

Shows message about Qt Gui library



You can navigate through 2D and 3D graph tabs. On the right of the window there is provided the table where you can check real values of the interpolating function.

Warning: 3D tab is yet demo. It just paints 3D pyramid. The user can rotate it by pressing a mouse button and dragging. The user can set the color of a face by double-clicking it and choosing a color from the Dialog that pops up.



Structure of the project

The first concept that is used in programming is cross-platform development. That is, once written, the program compiled and run without any changes in code. This platform independent software can be compiled and run on most modern operating systems such as Windows, Linux and Mac X OS. This is done due to keeping ANSI requirements and using libraries that support cross-platform concept.

GUI part of the project is implemented in Qt cross-platform library. To be independent from compiler, the project has its own built-in libraries. For example, in order to parse command line options and arguments, project has its own `libCmdArgParse.c` library. It means that to parse command line arguments neither Visual Studio's MFC, nor GNU's `getop.h`, nor Mac's xCode parsers are used. Project also has its own `libGauss.cpp` library.

Project is logically divided into three part:

- `Al-Jabr-Core` part contains mathematical functions that build piecewise linear interpolation and find interpolating function's value in an arbitrary point in the interval.
- `libAl-Jabr` part contains set of library functions such as parser, input file reader, sorting, etc. that are used by the project.
- `Al-Jabr-Qt4` part contains main GUI classes.

Implementation of the program

Now let's get to know source code closely. Aforementioned project parts do not depend on each other anyhow. For example, `Al-Jabr-Qt4` part does not know any thing about core or libraries. It just paints functions that are given by pointers.

Firstly we allocate memory for the file name, array `x`, array `f` and a coefficient array:

```
filename = (char * ) malloc (sizeof(char[20]));

x = (float *) malloc ( sizeof ( float [n+2] ));
f = (float *) malloc ( sizeof ( float [n+2] ));

a = (float **) malloc (sizeof (float * [n+1]));
for(i=0;i<n+1;++i)
     a[i] = (float *) malloc (sizeof (float [3]));
```

Then we call `cmd_args_support` which extracts data from the string passed by command line

```
int cmd_args_support(int argc, char **argv, char *filename,
float &interval_a , float &interval_b , int &n )
```

This function, in turn, uses `Process_Cmd_Switch_List` function to parse options.

```
OptionPattern  optpattern[]={
          {1, "-f","",           REQUIRE_ARGUMENT},
          {2,  "", "--from",     REQUIRE_ARGUMENT},
          {3,  "", "--to",       REQUIRE_ARGUMENT},
          {4,  "-n","--number-of-points",REQUIRE_ARGUMENT},
          {5,  "", "--no-gui",   NO_ARGUMENT},
          {6,  "-h","--help",        NO_ARGUMENT},
          {7,  "", "--qt-help",  NO_ARGUMENT},
          {8,  "-v","",          NO_ARGUMENT},
          {9,  "", "",           FREE_ARGUMENT},
          {10, "-t","--func-type",    REQUIRE_ARGUMENT},
          {11, "-l",    "--linear",   ALLOW_ARGUMENT},
          {12, "-p","--parabolic",    ALLOW_ARGUMENT} ,
          {13, "", "-style",     ALLOW_ARGUMENT}
                    };

OptionList *list,error;
list=Process_Cmd_Switch_List(argc,argv,myoptin,error);
```

`Process_Cmd_Switch_List` function returns linked list of arguments.

If any error occurs, `cmd_args_support` returns -1. As its first two arguments, it takes `argc` and `argv` which were sent by stack to the main program and analyzes command line options. After the analysis it assigns properly converted arguments to the provided parameters as result:

1. number of points
2. beginning of the interval
3. ending of the interval
4. input file name

if some arguments are not sent by command line then -1 is assigned to respecting parameter and if no input file name is given, then returns 0. After that we call `read_from_file` function:

```
int read_from_file(float * x, float * f, char * filename,
float &interval_a ,float &interval_b ,int &n)
```

Function reads data (x1,x2... y1,y2...) from the file and sorts them in increasing order. 1- and 2-arguments are pointers to the memory where data will be written to. 3-argument donates file name as string. if interval values are illogical then `x1` and `xn` will be used as interval limits. Function returns number of points read. if the file is empty, 0 is returned and if the file doesn‘t exist, -1 is returned. Then we build linear functions

```
int build_piecewise_linear_interpolation(int n, float *x,
float *f , float **a)
```

Function builds set of linear functions which are determined in $[x_i , x_{i+1}]$ intervals. `find_piecewise_linear_interpolation_value` function evaluates interpolating function value:

```
int find_piecewise_linear_interpolation_value(float &x0,
float interval_a, float interval_b, int n, float **a, float
*x)
```

There are also `build_parabolic_spline_interpolation` and `find_parabolic_spline_interpolation_value` functions for parabolic splines method.

Additional functions:

```
void sort(float * x, float * f , int n)
```

Sorts array `x` and applies all replacements to array `f` too.
In order to load external shared object files, `dlopen()` function is used.

Qt4 part of the program consists of two classes: Main Window Class and Drawer Widget Class. Necessary data are passed to GUI object through structure defined in `data.h`. So, we've got:

```
main()
{
  int (*interpolate)(float &,float,float,int,float **,float *);
     mem = malloc(...);
     ...
     cmd_args_support(...);
     ...
     if(bitmap & FJ::LinearMethod)
     {
     build_piecewise_linear_interpolation(...);
     interpolate = find_piecewise_linear_interpolation_value;
     }
     else
     {
     build_parabolic_spline_interpolation(...);
     interpolate = find_parabolic_spline_interpolation_value;
     }
     ...
     if(!(bitmap & FJ::NoGui))
     {
          data = new Data;
          data->begin=interval_a;
          data->end=interval_b;
          data->x=x;
          data->f=f;
          data->N=n;
          data->a=a;
          data->ksi=ksi;
          data->virtx = virx;
          data->bitmap=bitmap;
          data->func=func[ftype];
          data->interpolate = interpolate;

          window = new Aljabr(data);
          window->show();
          return app.exec();
     }
```

First we create pointer to the interpolating function, so that we could pass it to GUI object later. Main Window Class is called Aljabr and it has

following declaration:

```
class Aljabr : public QMainWindow
{
    Q_OBJECT

public:
    Aljabr(Data * d = 0);

protected:

private slots:
    void save();
    void about();
    ...

private:
    void createActions();
    void createMenus();
    void createToolBar();
    void createStatusBar();

    FGraph2d * graph2d;
    QTableWidget * point_list;
    QTabWidget *tab;
    QLineEdit *text2dX;
    QLabel *label;
    QPushButton *flush_list;
    QPushButton *add;
    QLabel *scale;
    int row;

    QMenu *fileMenu;
    ...

    QAction *openAct;
    ...

    QToolBar * toolbar;
};
```

For more information see `Al-Jabr-Qt4.h` and `Al-Jabr-Qt4.cpp` files. If you spotted, Main Window Class has `FGraph2d * graph2d` member. It is Drawer Widget Class and we are mostly interested in this class:

```
class FGraph2d :public QWidget
{
    Q_OBJECT
```

```
public:
     FGraph2d(Data * , QWidget *parent = 0);
     QSize minimumSizeHint() const;
     QSize sizeHint() const;
     int calc(float,float &,float&);
     void refreshPixmap();

     QColor funcPenColor;
     ...

     int penWidth;
     int margin;

public slots:
     void zoomIn();
     void zoomOut();
     ...

protected:
     void paintEvent(QPaintEvent *event);
     void mouseMoveEvent(QMouseEvent *event);
     void resizeEvent(QResizeEvent *event);
     void keyPressEvent(QKeyEvent *event);

private:
     void drawGrids(QPainter *);
     void drawFunctions(QPainter *);
     void initData();
     void initPainter(QPainter *);

     QPointF fromWidget(QPointF);
     QPointF toWidget(QPointF);
     QPointF inFunc(QPointF);

     Data * data;
     QRect * viewport;
     QRect * window;
     QPixmap pixmap;
     QPointF mark;
     QPoint centre;

     int Axis;
     bool effects;
     bool error;
     bool func;
     enum {
          axisNo = 0x0,
          axisX  = 0x1,
          axisY  = 0x2,
```

```
            axisXY = 0x4
            };

    };
```

`drawFunctions` method fulfills all main drawings:

```
void FGraph2d::drawFunctions(QPainter * painter)
{
    ...

    // Drawing interpolating function
    x0 = begin;
    p1.setX(x0);
    data->interpolate(x0,begin,end,n,a,x);
    p1.setY(x0);

    while(p1.x()<=end-step)
        {
        x0 = p1.x()+step;
        p2.setX(x0);
        data->interpolate(x0,begin,end,n,a,x);
        p2.setY(x0);
        painter->drawLine(p1,p2);
        p1 = p2;
        }

    //// Drawing default function if no input file is given
    ...
    p1.setX(begin);
    p1.setY(data->func(begin));

    while(p1.x()<end)
        {
        p2.setX(p1.x()+step);
        p2.setY(data->func(p2.x()));
        painter->drawLine(p1,p2);
        p1 = p2;
        }

    //// Drawing error function if no input file
    //// is given and Show error is triggered
    ...
    x1 = x0 = begin+step;
    p1.setX(x0);

    x0 = data->func(x0);
    data->interpolate(x1,begin,end,n,a,x);
```

```
x0 = FJ::abs(x0 - x1);
x0+=(float) pad;
p1.setY(x0);

while(p1.x()<end-step)
     {
     x1 = x0 = p1.x()+step;
     p2.setX(x0);
     x0 = data->func(x0);
     data->interpolate(x1,begin,end,n,a,x);

     x0 = FJ::abs(x1 - x0);
     x0+=(float) pad;
     p2.setY(x0);
     painter->drawLine(p1,p2);
     p1 = p2;
     }
```

For more information see and `Fgraph2D-Qt4.h` and `Fgraph2D-Qt4.cpp` files.

Warning: Code extracts used in this text are simplified enough and may differ from real code.

Bibliography

1. “C++ GUI Programming with Qt 4”, Jasmin Blanchette and Mark Summerfield , Massachusetts 2006.
2. “Approaches to Knot Generation for Best Piecewise Linear Interpolation”, Robert Ketzscher, Cranfield University 2000.
3. “Методы приближения функций”, К.Ю. Богачев, Москва 1998.