Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования «Московский политехнический университет»

**факультет информационных технологий**

**Кафедра СМАРТ-технологий**

Дисциплина: Технологии визуализации данных систем управления

Отчёт по лабораторной работе №1

«Использование графических возможностей C# для визуализации данных стохастических процессов»

Работу выполнена:

<ФИО>

Научный руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

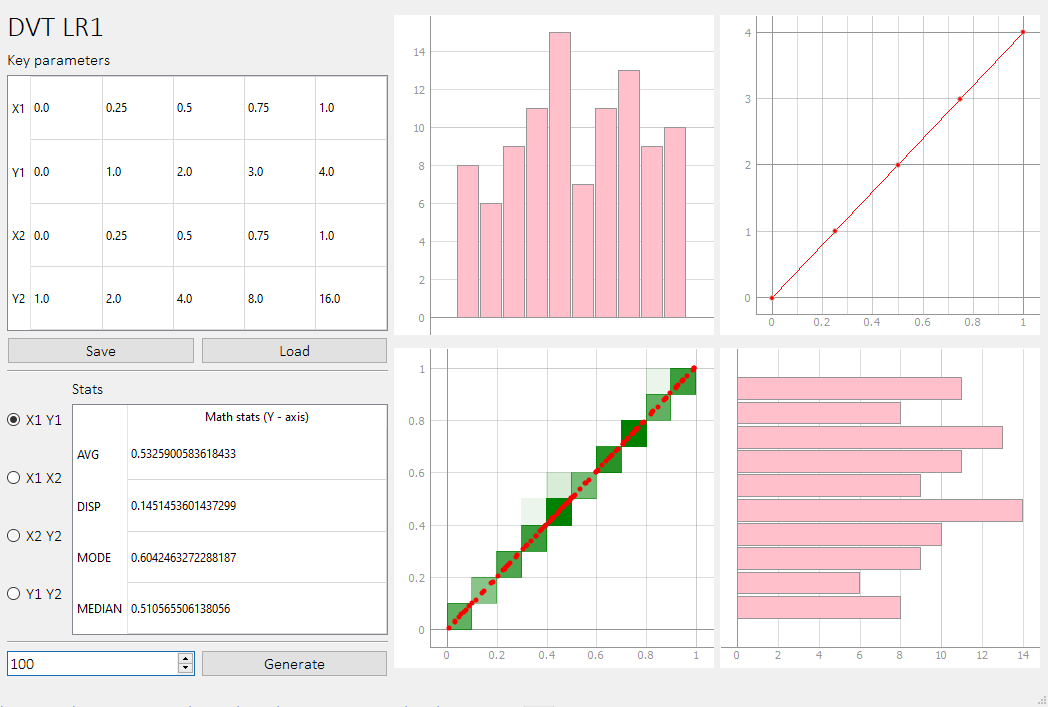
<ФИО>

**Цель работы**

Разработать приложение по генерации стохастических данных с заданным профилем распределения и визуализации распределения случайных величин.**Задачи**

* Подготовить приложение на языке C# для статистической обработки и визуализации собранных наборов данных
* Реализовать функции анализа данных

**Ход работы**

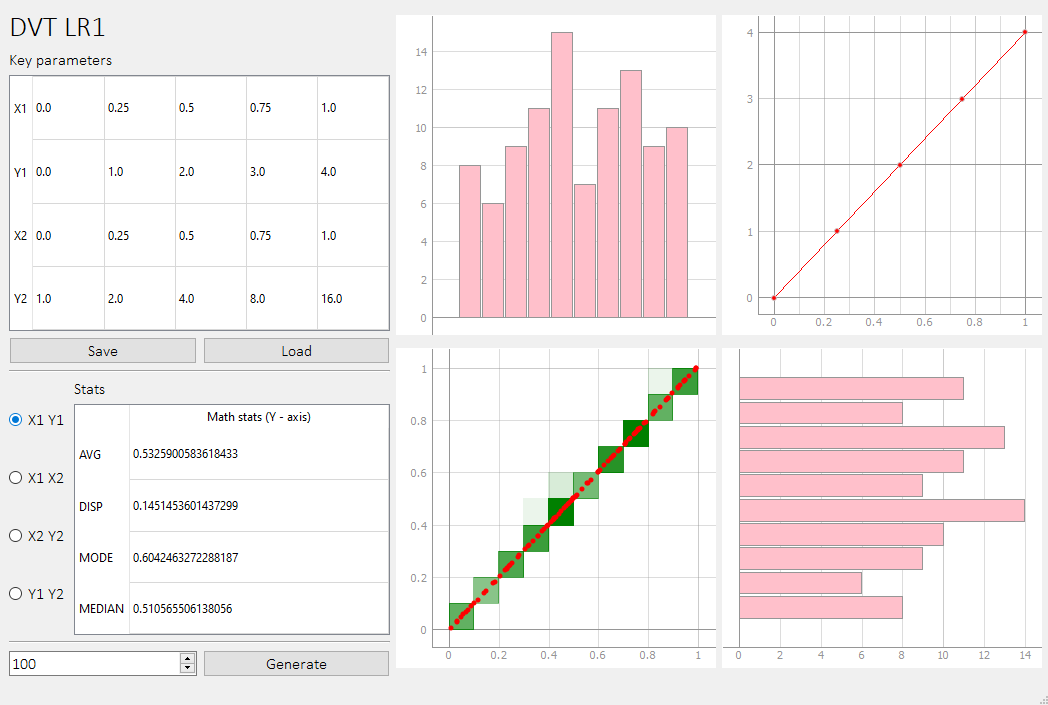
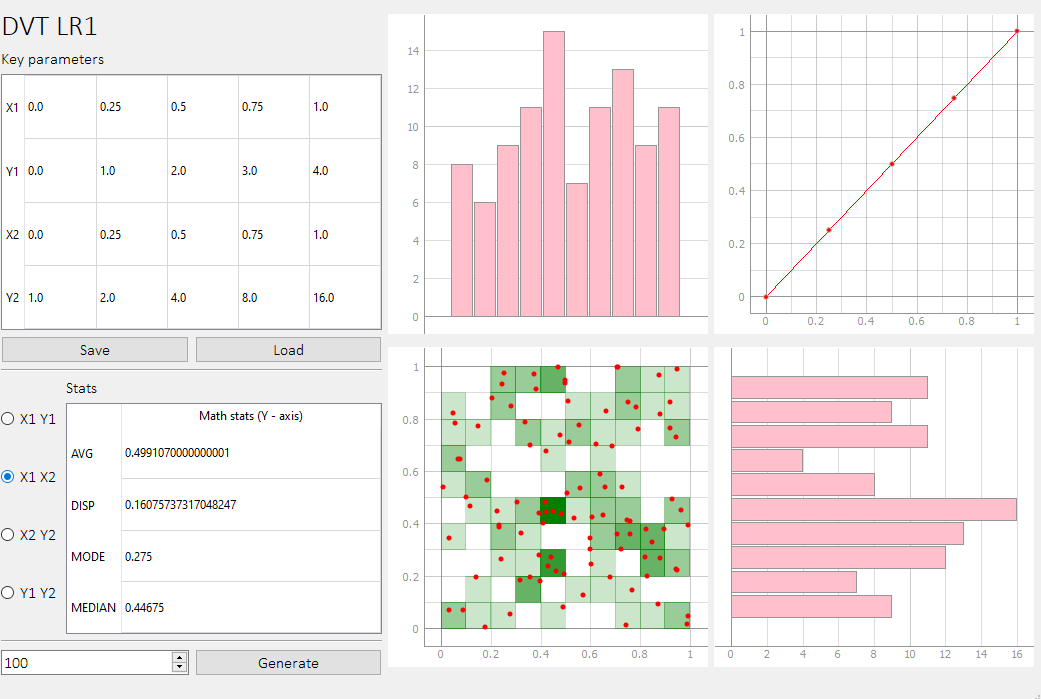
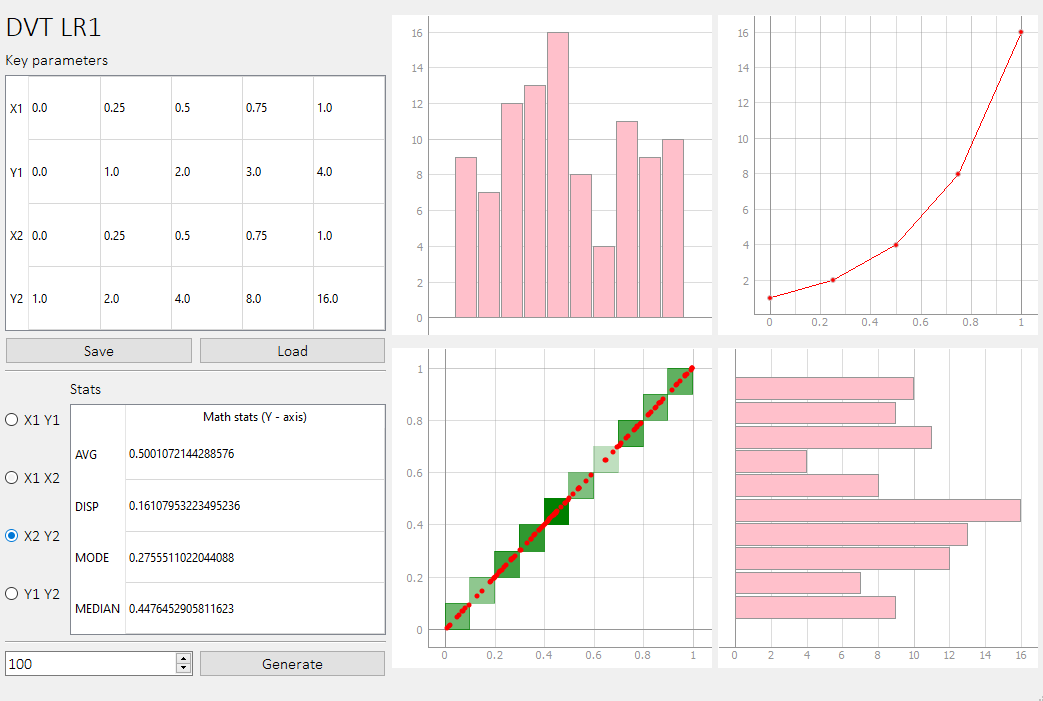


Параметры профиля преобразования (Key parameters) задаются пользователем с помощью таблицы и сохраняются в .json файл по нажатию кнопки Save. Имеется и возможность загрузки их из .json файла (по нажатию кнопки Load)  


Для начала работы необходимо загрузить профиль преобразования, выбрать количество элементов и нажать Generate. Будут сгенерированы случайные числа (массивы X1 и X2) в диапазоне от 0 до 1 в указанном количестве. Из этих массивов с помощью профиля преобразования генерируются массивы Y1 и Y2 соответственно.

После генерации происходит отрисовка всех данных, включая отображение облака точек, «блоков» точек, гистограмм плотности по двум осям, а также, диаграммы профиля преобразования.

Между осями можно переключатся.

**Вывод**

В ходе данной работы было создано приложение по генерации стохастических данных с заданным профилем распределения и визуализации распределения случайных величин

**Исходный код**

import json  
import statistics  
import sys  
  
import numpy as np  
import pyqtgraph as pg  
from PyQt5 import uic, QtGui, QtWidgets, QtCore  
from PyQt5.QtCore import Qt  
from PyQt5.QtWidgets import QApplication, QMainWindow, QTableWidgetItem  
  
PROJECT\_FOLDER = **'LR1\_data'**def valmap(value, istart, istop, ostart, ostop):  
 return ostart + (ostop - ostart) \* ((value - istart) / (istop - istart))  
  
  
class Window(QMainWindow):  
 def \_\_init\_\_(self):  
 super(Window, self).\_\_init\_\_()  
  
 *# Variables* self.keys = []  
 self.data\_x = []  
 self.data\_y = []  
 self.blocks = []  
 self.max\_block\_value = 0  
 self.bars\_x = []  
 self.bars\_y = []  
  
 *# Load GUI file* self.gui = uic.loadUi(**'LR1.ui'**)  
 self.gui.show()  
  
 *# Connect GUI controls* self.gui.pushButton.clicked.connect(self.save\_keys)  
 self.gui.pushButton\_2.clicked.connect(self.load\_keys)  
 self.gui.pushButton\_3.clicked.connect(self.generate)  
 self.gui.radioButton.clicked.connect(self.update\_view)  
 self.gui.radioButton\_2.clicked.connect(self.update\_view)  
 self.gui.radioButton\_3.clicked.connect(self.update\_view)  
 self.gui.radioButton\_4.clicked.connect(self.update\_view)  
  
 *# Initialize charts and tables* self.init\_charts()  
 self.init\_tables()  
  
 **"""  
 Initializes charts  
 """** def init\_charts(self):  
 self.gui.graphWidget.setBackground(QtGui.QColor(**'white'**))  
 self.gui.graphWidget\_2.setBackground(QtGui.QColor(**'white'**))  
 self.gui.graphWidget\_3.setBackground(QtGui.QColor(**'white'**))  
 self.gui.graphWidget\_4.setBackground(QtGui.QColor(**'white'**))  
 *# self.gui.graphWidget.getAxis('left').setPen(QtGui.QColor('black'))  
 # self.gui.graphWidget.getAxis('left').setTextPen(QtGui.QColor('black'))  
 # self.gui.graphWidget.getAxis('bottom').setPen(QtGui.QColor('black'))  
 # self.gui.graphWidget.getAxis('bottom').setTextPen(QtGui.QColor('black'))* self.gui.graphWidget\_2.getPlotItem().hideAxis(**'left'**)  
 self.gui.graphWidget\_3.getPlotItem().hideAxis(**'bottom'**)  
 self.gui.graphWidget.showGrid(x=True, y=True, alpha=1.0)  
 self.gui.graphWidget\_2.showGrid(x=True, y=True, alpha=1.0)  
 self.gui.graphWidget\_3.showGrid(x=True, y=True, alpha=1.0)  
 self.gui.graphWidget\_4.showGrid(x=True, y=True, alpha=1.0)  
  
 **"""  
 Initializes tables  
 """** def init\_tables(self):  
 *# Key parameters table* self.gui.tableWidget.setColumnCount(5)  
 self.gui.tableWidget.setRowCount(4)  
 self.gui.tableWidget.horizontalHeader().setVisible(False)  
 self.gui.tableWidget.setCornerButtonEnabled(False)  
 self.gui.tableWidget.setSelectionMode(QtWidgets.QAbstractItemView.NoSelection)  
 self.gui.tableWidget.setFocusPolicy(Qt.NoFocus)  
  
 header = self.gui.tableWidget.horizontalHeader()  
 header.setSectionResizeMode(0, QtWidgets.QHeaderView.Stretch)  
 for i in range(self.gui.tableWidget.columnCount()):  
 header.setSectionResizeMode(i, QtWidgets.QHeaderView.Stretch)  
  
 header = self.gui.tableWidget.verticalHeader()  
 header.setSectionResizeMode(0, QtWidgets.QHeaderView.Stretch)  
 for i in range(self.gui.tableWidget.rowCount()):  
 header.setSectionResizeMode(i, QtWidgets.QHeaderView.Stretch)  
  
 self.gui.tableWidget.setVerticalHeaderLabels([**'X1'**, **'Y1'**, **'X2'**, **'Y2'**])  
  
 *# Stats table* self.gui.tableWidget\_2.setColumnCount(1)  
 self.gui.tableWidget\_2.setRowCount(4)  
 self.gui.tableWidget\_2.setCornerButtonEnabled(False)  
 self.gui.tableWidget\_2.setSelectionMode(QtWidgets.QAbstractItemView.NoSelection)  
 self.gui.tableWidget\_2.setFocusPolicy(Qt.NoFocus)  
 self.gui.tableWidget\_2.setEditTriggers(QtWidgets.QAbstractItemView.NoEditTriggers)  
  
 header = self.gui.tableWidget\_2.horizontalHeader()  
 header.setSectionResizeMode(0, QtWidgets.QHeaderView.Stretch)  
 for i in range(self.gui.tableWidget\_2.columnCount()):  
 header.setSectionResizeMode(i, QtWidgets.QHeaderView.Stretch)  
  
 header = self.gui.tableWidget\_2.verticalHeader()  
 header.setSectionResizeMode(0, QtWidgets.QHeaderView.Stretch)  
 for i in range(self.gui.tableWidget\_2.rowCount()):  
 header.setSectionResizeMode(i, QtWidgets.QHeaderView.Stretch)  
  
 self.gui.tableWidget\_2.setVerticalHeaderLabels([**'AVG'**, **'DISP'**, **'MODE'**, **'MEDIAN'**])  
 self.gui.tableWidget\_2.setHorizontalHeaderLabels([**'Math stats (Y - axis)'**])  
  
 **"""  
 Generates random X numbers and calcultaes Y values  
 """** def generate(self):  
 *# Check if keys are loaded* assert len(self.keys) > 0  
  
 self.data\_x = [[round(num, 4) for num in np.random.sample(self.gui.spinBox.value())],  
 [round(num, 4) for num in np.random.sample(self.gui.spinBox.value())]]  
 self.data\_y = []  
  
 for i in range(2):  
 array\_y = []  
 for data\_x in self.data\_x[i]:  
 *# Calculate Y* if self.keys[0][0] <= data\_x <= self.keys[0][1]:  
 *# if X >= keys\_X1[0] && X <= keys\_X1[1]  
 # Y = keys\_Y1[1] / keys\_X1[1] \* X* array\_y.append(self.keys[1][1] / self.keys[0][1] \* data\_x)  
 if self.keys[0][1] < data\_x <= self.keys[0][2]:  
 *# if X > keys\_X1[1] && X <= keys\_X1[2]  
 # Y = keys\_Y1[2] / keys\_X1[2] \* X* array\_y.append(self.keys[1][2] / self.keys[0][2] \* data\_x)  
 if self.keys[0][2] < data\_x <= self.keys[0][3]:  
 *# if X > keys\_X1[2] && X <= keys\_X1[3]  
 # Y = keys\_Y1[3] / keys\_X1[3] \* X* array\_y.append(self.keys[1][3] / self.keys[0][3] \* data\_x)  
 if self.keys[0][3] < data\_x <= self.keys[0][4]:  
 *# if X > keys\_X1[3] && X <= keys\_X1[4]  
 # Y = keys\_Y1[4] / keys\_X1[4] \* X* array\_y.append(self.keys[1][4] / self.keys[0][4] \* data\_x)  
  
 *# Normalize data* array\_y /= max(array\_y)  
  
 self.data\_y.append(array\_y)  
  
 self.update\_view()  
  
 **"""  
 Draws keys chart, main chart and bar charts  
 """** def update\_view(self):  
 *# Keys chart* if self.gui.radioButton.isChecked():  
 *# X1 Y1* keys\_x = self.keys[0]  
 keys\_y = self.keys[1]  
 data\_x = self.data\_x[0]  
 data\_y = self.data\_y[0]  
 elif self.gui.radioButton\_2.isChecked():  
 *# X1 X2* keys\_x = self.keys[0]  
 keys\_y = self.keys[2]  
 data\_x = self.data\_x[0]  
 data\_y = self.data\_x[1]  
 elif self.gui.radioButton\_3.isChecked():  
 *# X2 Y2* keys\_x = self.keys[2]  
 keys\_y = self.keys[3]  
 data\_x = self.data\_x[1]  
 data\_y = self.data\_y[1]  
 else:  
 *# X1 Y2* keys\_x = self.keys[1]  
 keys\_y = self.keys[3]  
 data\_x = self.data\_x[0]  
 data\_y = self.data\_y[1]  
  
 self.gui.graphWidget\_4.clear()  
 self.gui.graphWidget\_4.plot(keys\_x, keys\_y, pen=pg.mkPen(QtGui.QColor(**'red'**)),  
 symbolBrush=(255, 0, 0),  
 symbolSize=5)  
  
 *# Count how many points in each block* self.count\_blocks(data\_x, data\_y)  
  
 *# Draw blocks (background)* self.gui.graphWidget.clear()  
 for x in range(10):  
 for y in range(10):  
 self.gui.graphWidget.addItem(  
 RectItem(QtCore.QRectF(x / 10, y / 10, 0.1, 0.1),  
 alpha=valmap(self.blocks[x][y], 0, self.max\_block\_value, 0, 1)))  
  
 *# Draw main chart* self.gui.graphWidget.plot(data\_x, data\_y, pen=None,  
 symbolBrush=(255, 0, 0), symbolSize=5, symbolPen=None)  
  
 *# Bar charts* bar\_x = list(range(10))  
 self.gui.graphWidget\_2.clear()  
 self.gui.graphWidget\_3.clear()  
 bar\_top = pg.BarGraphItem(x=bar\_x, height=self.bars\_x, width=0.9, brush=QtGui.QColor(**'pink'**))  
 bar\_right = pg.BarGraphItem(x=bar\_x, height=self.bars\_y, width=0.9, brush=QtGui.QColor(**'pink'**))  
 bar\_right.rotate(-90)  
 self.gui.graphWidget\_3.addItem(bar\_top)  
 self.gui.graphWidget\_2.addItem(bar\_right)  
  
 *# Math stats* self.count\_stats(data\_y)  
  
 **"""  
 Counts how many points in each block  
 """** def count\_blocks(self, data\_x, data\_y):  
 self.max\_block\_value = 0  
 self.blocks = []  
 self.bars\_x = [0] \* 10  
 self.bars\_y = [0] \* 10  
 for x in range(10):  
 blocks\_column = []  
 for y in range(10):  
 points\_in\_block = 0  
 for i in range(len(data\_x)):  
 if x / 10 <= data\_x[i] < (x / 10 + 0.1) \  
 and y / 10 <= data\_y[i] < (y / 10 + 0.1):  
 points\_in\_block += 1  
 if points\_in\_block > self.max\_block\_value:  
 self.max\_block\_value = points\_in\_block  
  
 self.bars\_x[x] += points\_in\_block  
 self.bars\_y[y] += points\_in\_block  
  
 blocks\_column.append(points\_in\_block)  
  
 self.blocks.append(blocks\_column)  
 self.bars\_y.reverse()  
  
 **"""  
 Calculates math stats (average, dispertion, mode, median)  
 """** *# noinspection PyBroadException* def count\_stats(self, data\_y):  
 *# Average* average = sum(data\_y) / len(data\_y)  
  
 *# Dispertion* disp\_sum = 0  
 for point in data\_y:  
 disp\_sum += (point - average) \* (point - average)  
 dispertion = disp\_sum / sum(data\_y)  
  
 *# Median* try:  
 mode = statistics.mode(data\_y)  
 except:  
 mode = None  
  
 *# Median* median = statistics.median(data\_y)  
  
 *# Fill table* self.gui.tableWidget\_2.setItem(0, 0, QTableWidgetItem(str(average)))  
 self.gui.tableWidget\_2.setItem(0, 1, QTableWidgetItem(str(dispertion)))  
 if mode is not None:  
 self.gui.tableWidget\_2.setItem(0, 2, QTableWidgetItem(str(mode)))  
 else:  
 self.gui.tableWidget\_2.setItem(0, 2, QTableWidgetItem(**'No Mode'**))  
 self.gui.tableWidget\_2.setItem(0, 3, QTableWidgetItem(str(median)))  
  
 **"""  
 Saves keys to JSON file  
 """** def save\_keys(self):  
 self.keys = []  
 for i in range(self.gui.tableWidget.rowCount()):  
 keys\_row = []  
 for k in range(self.gui.tableWidget.columnCount()):  
 keys\_row.append(float(self.gui.tableWidget.item(i, k).text()))  
 self.keys.append(keys\_row)  
  
 with open(PROJECT\_FOLDER + **'/keys.json'**, **'w'**) as f:  
 json.dump(self.keys, f)  
  
 **"""  
 Loads keys from JSON file  
 """** def load\_keys(self):  
 self.keys = np.array(json.load(open(PROJECT\_FOLDER + **'/keys.json'**)))  
  
 for i in range(self.gui.tableWidget.rowCount()):  
 for k in range(self.gui.tableWidget.columnCount()):  
 self.gui.tableWidget.setItem(i, k, QTableWidgetItem(str(self.keys[i][k])))  
  
  
**"""  
This class draws rectangles on the chart  
"""**class RectItem(pg.GraphicsObject):  
 def \_\_init\_\_(self, rect, alpha=1.0, parent=None):  
 super().\_\_init\_\_(parent)  
 self.\_rect = rect  
 self.\_alpha = alpha  
 self.picture = QtGui.QPicture()  
 self.\_generate\_picture()  
  
 @property  
 def rect(self):  
 return self.\_rect  
  
 @property  
 def alpha(self):  
 return self.\_alpha  
  
 def \_generate\_picture(self):  
 painter = QtGui.QPainter(self.picture)  
 color = QtGui.QColor(**'green'**)  
 color.setAlphaF(self.alpha)  
 painter.setPen(pg.mkPen(color))  
 painter.setBrush(pg.mkBrush(color))  
 painter.drawRect(self.rect)  
 painter.end()  
  
 def paint(self, painter, option, widget=None):  
 painter.drawPicture(0, 0, self.picture)  
  
 def boundingRect(self):  
 return QtCore.QRectF(self.picture.boundingRect())  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 app = QApplication(sys.argv)  
 win = Window()  
 sys.exit(app.exec\_())

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>MainWindow</class>

<widget class="QMainWindow" name="MainWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>1051</width>

<height>707</height>

</rect>

</property>

<property name="font">

<font>

<family>Calibri Light</family>

<pointsize>12</pointsize>

</font>

</property>

<property name="windowTitle">

<string>DVT LR1</string>

</property>

<widget class="QWidget" name="centralwidget">

<layout class="QGridLayout" name="gridLayout\_2">

<item row="1" column="0">

<layout class="QVBoxLayout" name="verticalLayout\_3">

<item>

<widget class="QLabel" name="label">

<property name="font">

<font>

<pointsize>22</pointsize>

</font>

</property>

<property name="text">

<string>DVT LR1</string>

</property>

</widget>

</item>

<item>

<widget class="QLabel" name="label\_2">

<property name="text">

<string>Key parameters</string>

</property>

</widget>

</item>

<item>

<widget class="QTableWidget" name="tableWidget"/>

</item>

<item>

<layout class="QHBoxLayout" name="horizontalLayout\_2">

<item>

<widget class="QPushButton" name="pushButton">

<property name="text">

<string>Save</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="pushButton\_2">

<property name="text">

<string>Load</string>

</property>

</widget>

</item>

</layout>

</item>

<item>

<widget class="Line" name="line">

<property name="orientation">

<enum>Qt::Horizontal</enum>

</property>

</widget>

</item>

<item>

<layout class="QHBoxLayout" name="horizontalLayout\_3">

<item>

<layout class="QVBoxLayout" name="verticalLayout\_2">

<item>

<widget class="QRadioButton" name="radioButton">

<property name="text">

<string>X1 Y1</string>

</property>

<property name="checked">

<bool>true</bool>

</property>

</widget>

</item>

<item>

<widget class="QRadioButton" name="radioButton\_2">

<property name="text">

<string>X1 X2</string>

</property>

</widget>

</item>

<item>

<widget class="QRadioButton" name="radioButton\_3">

<property name="text">

<string>X2 Y2</string>

</property>

</widget>

</item>

<item>

<widget class="QRadioButton" name="radioButton\_4">

<property name="text">

<string>Y1 Y2</string>

</property>

</widget>

</item>

</layout>

</item>

<item>

<layout class="QVBoxLayout" name="verticalLayout">

<item>

<widget class="QLabel" name="label\_3">

<property name="text">

<string>Stats</string>

</property>

</widget>

</item>

<item>

<widget class="QTableWidget" name="tableWidget\_2"/>

</item>

</layout>

</item>

</layout>

</item>

<item>

<widget class="Line" name="line\_2">

<property name="orientation">

<enum>Qt::Horizontal</enum>

</property>

</widget>

</item>

<item>

<layout class="QHBoxLayout" name="horizontalLayout">

<item>

<widget class="QSpinBox" name="spinBox">

<property name="minimum">

<number>1</number>

</property>

<property name="maximum">

<number>99999</number>

</property>

<property name="value">

<number>20</number>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="pushButton\_3">

<property name="text">

<string>Generate</string>

</property>

</widget>

</item>

</layout>

</item>

</layout>

</item>

<item row="1" column="1">

<layout class="QGridLayout" name="gridLayout" rowstretch="0,0" rowminimumheight="0,0" columnminimumwidth="0,0">

<property name="sizeConstraint">

<enum>QLayout::SetDefaultConstraint</enum>

</property>

<item row="0" column="1">

<widget class="PlotWidget" name="graphWidget\_4" native="true">

<property name="enabled">

<bool>true</bool>

</property>

<property name="minimumSize">

<size>

<width>320</width>

<height>320</height>

</size>

</property>

<property name="maximumSize">

<size>

<width>320</width>

<height>320</height>

</size>

</property>

</widget>

</item>

<item row="1" column="0">

<widget class="PlotWidget" name="graphWidget" native="true">

<property name="minimumSize">

<size>

<width>320</width>

<height>320</height>

</size>

</property>

<property name="maximumSize">

<size>

<width>320</width>

<height>320</height>

</size>

</property>

</widget>

</item>

<item row="1" column="1">

<widget class="PlotWidget" name="graphWidget\_2" native="true">

<property name="minimumSize">

<size>

<width>320</width>

<height>320</height>

</size>

</property>

<property name="maximumSize">

<size>

<width>320</width>

<height>320</height>

</size>

</property>

</widget>

</item>

<item row="0" column="0">

<widget class="PlotWidget" name="graphWidget\_3" native="true">

<property name="minimumSize">

<size>

<width>320</width>

<height>320</height>

</size>

</property>

<property name="maximumSize">

<size>

<width>320</width>

<height>320</height>

</size>

</property>

<property name="sizeIncrement">

<size>

<width>0</width>

<height>0</height>

</size>

</property>

</widget>

</item>

</layout>

</item>

</layout>

</widget>

<widget class="QMenuBar" name="menubar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>1051</width>

<height>21</height>

</rect>

</property>

</widget>

<widget class="QStatusBar" name="statusbar"/>

</widget>

<customwidgets>

<customwidget>

<class>PlotWidget</class>

<extends>QWidget</extends>

<header>pyqtgraph</header>

<container>1</container>

</customwidget>

</customwidgets>

<resources/>

<connections/>

</ui>