```
private void OtsuBtn_Click(object sender, EventArgs e)
        {
            double [] prob = new double [GRAYLEVEL]; // prob of graylevels
            int [] histogram = new int [GRAYLEVEL]; // histogram
            double [] myu = new double [GRAYLEVEL]; // mean value for separation
            double[] omega = new double[GRAYLEVEL]; // prob of graylevels
            double[] sigma = new double[GRAYLEVEL]; // inter-class variance

            int i, j, k; // Loop variable
            int m_Nopixels; // No. of pixels N
            int threshold = 0; // threshold for binarization
            double max_sigma = 0.0;

            // Convert and smooth an image
            gray_image = My_Image.Convert<Gray, Byte>();
            binary_image = gray_image.CopyBlank(); // create an image of the same size

            // Calculation of a gray image's histogram
            m_Nopixels = gray_image.Width * gray_image.Height;
            for (i = 0; i < gray_image.Height; ++i) //rows
            {
                for (j = 0; j < gray_image.Width; ++j) //columns
                {
                    k = gray_image.Data[i,j,0];
                    histogram[k]++;
                }
            }

            // calculation of probability density
            for (i = 0; i < GRAYLEVEL; ++i)
            {
                prob[i] = (double) ((double)histogram[i] / (double)m_Nopixels);
            }

            // Otsu thresholding for binarization
            // omega & myu generation
            omega[0] = prob[0];
            myu[0] = 0.0;
            for (i = 1; i < GRAYLEVEL; ++i)
            {
                omega[i] = omega[i - 1] + prob[i];
                myu[i] = myu[i - 1] + (i * prob[i]);
            }

            // sigma maximization
            // sigma stands for inter-class variance
            // and determines optimal threshold value
            for (i = 0; i < GRAYLEVEL - 1; i++) // 2 is the class of thresholding
            {
                if (omega[i] != 0.0 && omega[i] != 1.0)
                {
                    sigma[i] = ((myu[GRAYLEVEL-1]*omega[i]-myu[i]) * (myu[GRAYLEVEL-
1]*omega[i]-myu[i])) / (omega[i]*(1.0-omega[i]));
                }
                else
                {
                    sigma[i] = 0.0;
                }
```

```
                if (sigma[i] > max_sigma)
                {
                    threshold = i;
                    max_sigma = sigma[i];
                }
            }

            // binarization output into thresholded_image
            binary_image = gray_image.ThresholdBinaryInv(new Gray(threshold), new
Gray(MAX_BRIGHTNESS));

            image_PCBX.Image = binary_image.ToBitmap();
        }
```