Name: Marcio Adriano de Campos Junior

No. USP: 11871625

report.doc Digital filters

Question 1.3:

Horizontal Edge Detector in africa.tif

	Time	Average	Minimum	Maximum
non-separable version	307ms	127.66	0	255
separable version	104 ms	127.66	0	255

Comparison non-separable:

Enter the part of the code written for vertical edge detector:

```
static public ImageAccess detectEdgeVertical_NonSeparable(ImageAccess input) {
    int nx = input.getWidth();
    int ny = input.getHeight();
    double arr[][] = new double[3][3];
    double pixel;
    ImageAccess out = new ImageAccess(nx, ny);
    for (int x = 0; x < nx; x++) {
        for (int y = 0; y < ny; y++) {
            input.getNeighborhood(x, y, arr);
            pixel = arr[2][0]+arr[2][1]+arr[2][2]-arr[0][0]-arr[0][1]-arr[0][2];
            pixel = pixel / 6.0;
            out.putPixel(x, y, pixel);
        }
    }
    out = rescale(out);
    return out;
}

```

Enter the part of the code written for horizontal edge detector:

```
static public ImageAccess detectEdgeHorizontal_NonSeparable(ImageAccess input) {
    int nx = input.getWidth();
    int ny = input.getHeight();
    double arr[][] = new double[3][3];
    double pixel;
    ImageAccess out = new ImageAccess(nx, ny);
    for (int x = 0; x < nx; x++) {
        for (int y = 0; y < ny; y++) {
            input.getNeighborhood(x, y, arr);
            pixel = arr[2][2]+arr[1][2]+arr[0][2]-arr[0][0]-arr[1][0]-arr[2][0];
            pixel = pixel / 6.0;
            out.putPixel(x, y, pixel);
        }
    }
    out = rescale(out);
    return out;
}

```

Comparison Separable:

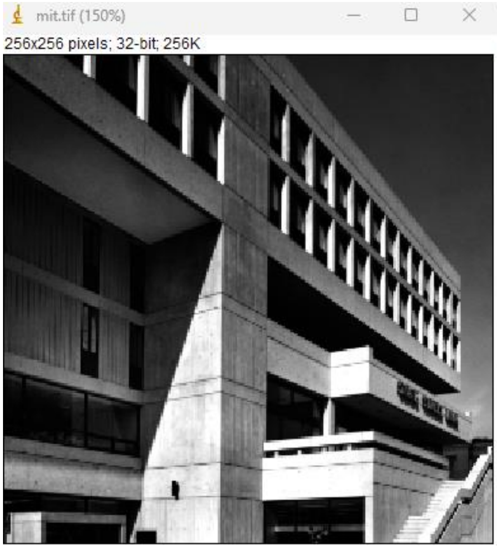
Enter the part of the code written for vertical edge detector:

```
static public ImageAccess detectEdgeVertical_Separable(ImageAccess input) {
    int nx = input.getWidth();
    int ny = input.getHeight();
    ImageAccess out = new ImageAccess(nx, ny);
    double rowin[]  = new double[nx];
    double rowout[] = new double[nx];
    for (int y = 0; y < ny; y++) {
        input.getRow(y, rowin);
        doDifference3(rowin, rowout);
        out.putRow(y, rowout);
    }

    double colin[]  = new double[ny];
    double colout[] = new double[ny];
    for (int x = 0; x < nx; x++) {
        out.getColumn(x, colin);
        doAverage3(colin, colout);
        out.putColumn(x, colout);
    }
    out = rescale(out);
    return out;
}

```

Enter the part of the code written for horizontal edge detector:

```
static public ImageAccess detectEdgeHorizontal_Separable(ImageAccess input) {
    int nx = input.getWidth();
    int ny = input.getHeight();
    ImageAccess out = new ImageAccess(nx, ny);

    double colin[]  = new double[ny];
    double colout[] = new double[ny];
    for (int x = 0; x < nx; x++) {
        input.getColumn(x, colin);
        doDifference3(colin, colout);
        out.putColumn(x, colout);
    }

    double rowin[]  = new double[nx];
    double rowout[] = new double[nx];
    for (int y = 0; y < ny; y++) {
        out.getRow(y, rowin);
        doAverage3(rowin, rowout);
        out.putRow(y, rowout);
    }
    out = rescale(out);
    return out;
}

```

Image results – Horizontal edge



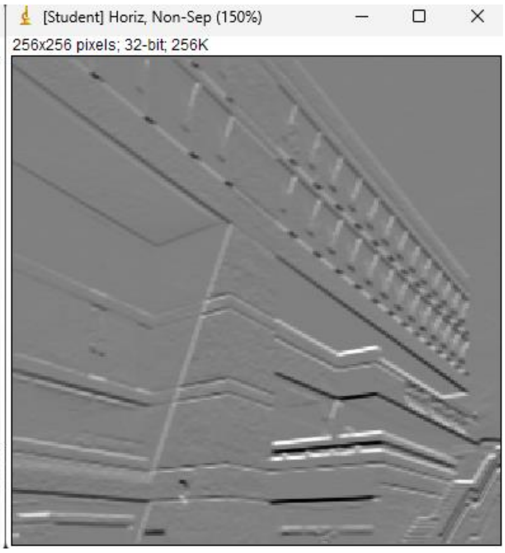
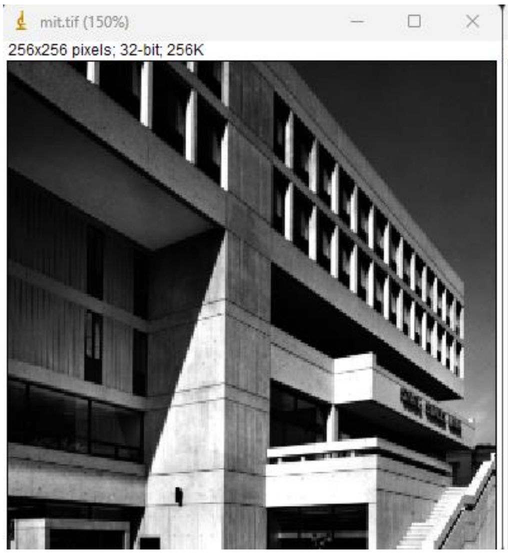
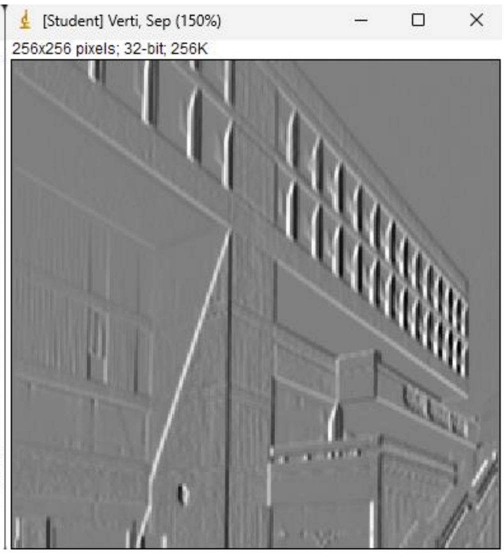


Image results – Vertical edge





Question 2:

Moving average 5*5 in africa.tif

	Time	Average	Minimum	Maximum
non-separable version	398 ms	71.44	0	255
separable version	95 ms	71.44	0	255
recursive version	111 ms	71.44	0	255

Comparison:

1. non-separable version

```
static public ImageAccess doMovingAverage5_NonSeparable(ImageAccess input) {
    int nx = input.getWidth();
    int ny = input.getHeight();
    double arr[][] = new double[5][5];
    double pixel = 0;
    ImageAccess out = new ImageAccess(nx, ny);
    for (int x = 0; x < nx; x++) {
        for (int y = 0; y < ny; y++) {
            input.getNeighborhood(x, y, arr);

            for(int i = 0 ; i < 5; i++){
                for(int j = 0 ; j < 5; j++){
                    pixel = pixel + arr[i][j];
            }}
            pixel = pixel / 25.0;
            out.putPixel(x, y, pixel);
        }
    }
    out = rescale(out);
    return out;
}
```

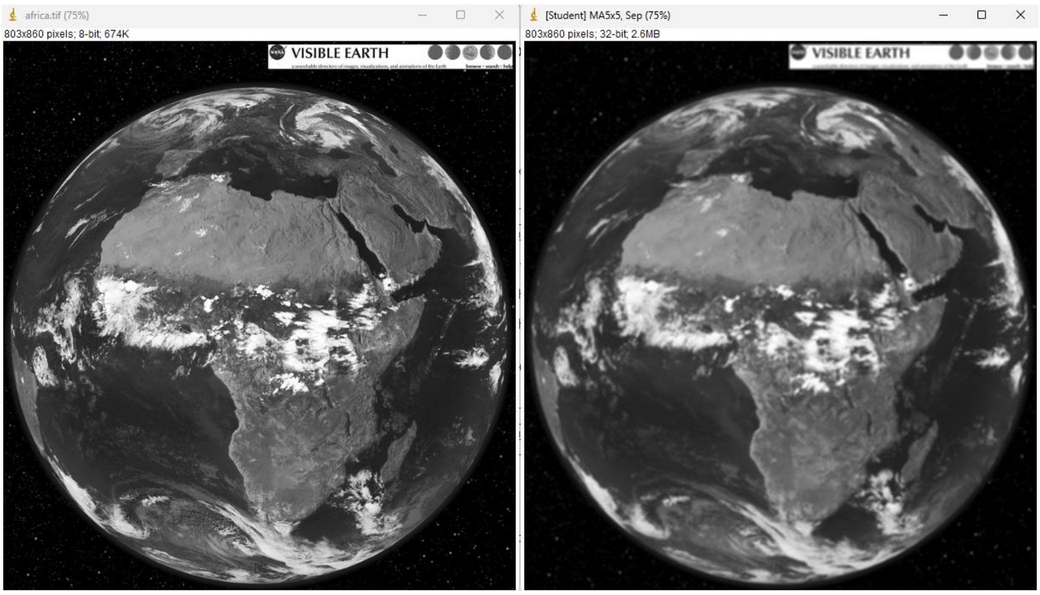
2. separable version

```
static public ImageAccess doMovingAverage5_Separable(ImageAccess input) {
    int nx = input.getWidth();
    int ny = input.getHeight();
    ImageAccess out = new ImageAccess(nx, ny);

    double colin[]  = new double[ny];
    double colout[] = new double[ny];

    for (int x = 0; x < nx; x++) {
        input.getColumn(x, colin);

        Average5(colin, colout);

        out.putColumn(x, colout);
    }

    double rowin[]  = new double[nx];
    double rowout[] = new double[nx];
    for (int y = 0; y < ny; y++) {
        out.getRow(y, rowin);
        Average5(rowin, rowout);
        out.putRow(y, rowout);
    }
    out = rescale(out);
    return out;
}

static private void Average5(double vin[], double vout[]){

    int n = vin.length;

    vout[0] = (vin[0] + 2.0 * vin[1] + 2.0 * vin[2]) / 5.0;
    vout[1] = (vin[0] + vin[1] + vin[2] + 2.0 *vin[3]) / 5.0;

    for (int k = 2; k < n-2; k++) {
        vout[k] = (vin[k-2] + vin[k-1] + vin[k] + vin[k+1] + vin[k+2]) / 5.0;
    }
    vout[n-2] = (vin[n-1] + vin[n-2] + vin[n-3] + 2 * vin[n-4]) / 5.0;
    vout[n-1] = (vin[n-1] + 2.0 * vin[n-2] + 2.0 * vin[n-3]) / 5.0;

}
```

3. recursive version

```java
static public ImageAccess doMovingAverage5_Recursive(ImageAccess input) {
    int nx = input.getWidth();
    int ny = input.getHeight();
    ImageAccess out = new ImageAccess(nx, ny);

    double colin[]  = new double[ny];
    double colout[] = new double[ny];

    for (int x = 0; x < nx; x++) {
        input.getColumn(x, colin);
        recursiveAverage5(colin, colout, 0);
        out.putColumn(x, colout);
    }

    double rowin[]  = new double[nx];
    double rowout[] = new double[nx];
    for (int y = 0; y < ny; y++) {
        out.getRow(y, rowin);
        recursiveAverage5(rowin, rowout, 0);
        out.putRow(y, rowout);
    }
    out = rescale(out);
    return out;
}

static private void recursiveAverage5(double vin[], double vout[], int k) {
    int n = vin.length;

    if (k == 0) {
        vout[0] = (vin[0] + 2.0 * vin[1] + 2.0 * vin[2]) / 5.0;
        recursiveAverage5(vin, vout, k + 1);
    } else if (k == 1) {
        vout[1] = (vin[0] + vin[1] + vin[2] + 2.0 * vin[3]) / 5.0;
        recursiveAverage5(vin, vout, k + 1);
    } else if (k >= 2 && k < n - 2) {
        vout[k] = (vin[k - 2] + vin[k - 1] + vin[k] + vin[k + 1] + vin[k + 2]) / 5.0;
        recursiveAverage5(vin, vout, k + 1);
    } else if (k == n - 2) {
        vout[n - 2] = (vin[n - 1] + vin[n - 2] + vin[n - 3] + 2.0 * vin[n - 4]) / 5.0;
        recursiveAverage5(vin, vout, k + 1);
    } else if (k == n - 1) {
        vout[n - 1] = (vin[n - 1] + 2.0 * vin[n - 2] + 2.0 * vin[n - 3]) / 5.0;
    }
}
```

Image results


africa.tif (75%)
803x860 pixels; 8-bit; 674K
VISIBLE EARTH
[Student] MA5x5, Sep (75%)
803x860 pixels; 32-bit; 2.6MB
VISIBLE EARTH

Question 3.1:

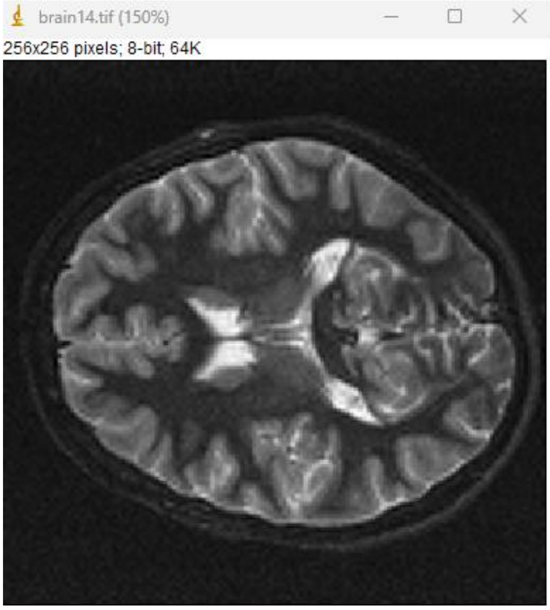
Segmenting with the Smoothing Operator
L = 15
T = 35

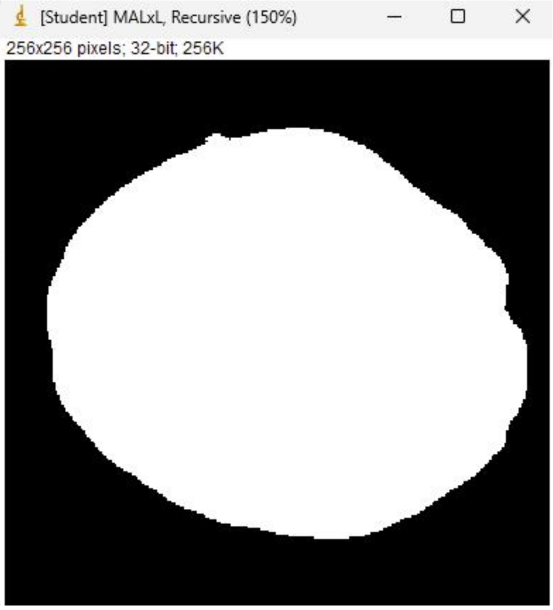
Enter the part of the code you wrote.

Note: imagej presented an error when trying to implement the recursive mode due to memory issues, so the mode without recursion was maintained.

```java
    static public ImageAccess doMovingAverageL_Recursive(ImageAccess input, int length) {
        int nx = input.getWidth();
        int ny = input.getHeight();
        ImageAccess out = new ImageAccess(nx, ny);

        int halfWindow = length / 2;

        for (int row = 0; row < nx; row++) {
            for (int col = 0; col < ny; col++) {
                double sum = 0.0;
                int count = 0;

                for (int i = -halfWindow; i <= halfWindow; i++) {
                    for (int j = -halfWindow; j <= halfWindow; j++) {
                        int rowIndex = row + i;
                        int colIndex = col + j;

                        // Espelhamento nas bordas
                        if (rowIndex < 0) rowIndex = -rowIndex;
                        if (rowIndex >= nx) rowIndex = 2 * nx - rowIndex - 2;
                        if (colIndex < 0) colIndex = -colIndex;
                        if (colIndex >= ny) colIndex = 2 * ny - colIndex - 2;

                        sum += input.getPixel(rowIndex, colIndex);
                        count++;
                    }
                }
                out.putPixel(row, col, sum / count);
            }
        }

        return rescale(out);}
```

Image results

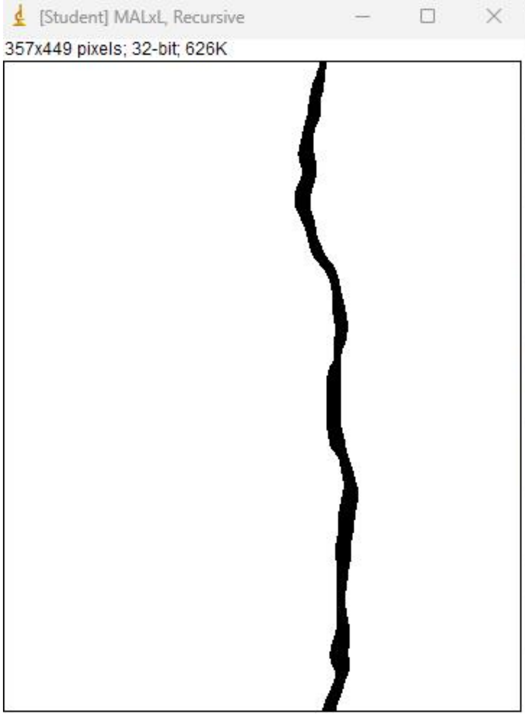


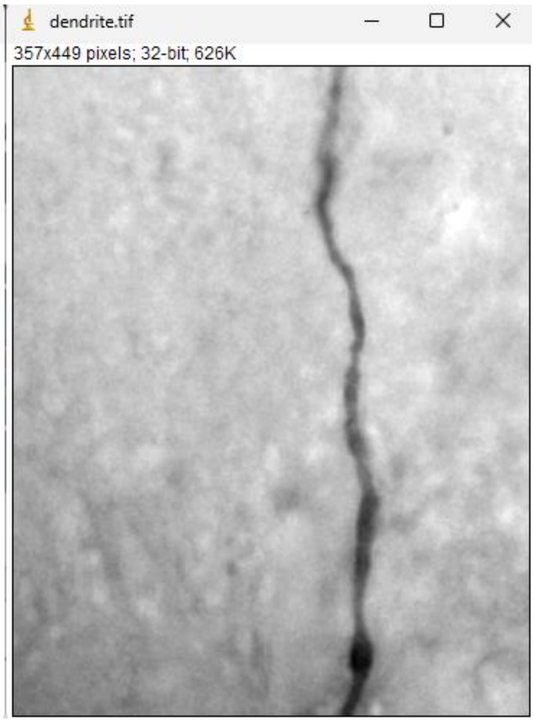

[Student] MALxL, Recursive (150%)
256x256 pixels; 32-bit; 256K

Question 3.2:

Para segmentação da imagem “dentrite.tif” foi necessário utilizar a detecção de borda vertical juntamente com o filtro de média móvel com tamanho de janela de 11 pixels, logo com um limiar de escala de 83 foi obtido a máscara.

Image results





Question 4:

Sobel Operator

Enter the part of the code written for this exercise

```java
    static public ImageAccess doSobel(ImageAccess input) {
        int nx = input.getWidth();
        int ny = input.getHeight();
        double arr[][] = new double[3][3];
        double pixel;
        double pixel_x;
        double pixel_y;
        ImageAccess out = new ImageAccess(nx, ny);
        for (int x = 0; x < nx; x++) {
            for (int y = 0; y < ny; y++) {
                input.getNeighborhood(x, y, arr);
                pixel_x = arr[2][1] + 2*arr[2][1] + arr[2][1] - arr[0][0]- 2*arr[0][1]- arr[0][2];
                pixel_x = Math.pow(pixel_x, 2);
                pixel_y = - arr[2][0] - 2*arr[1][0] - arr[0][0] + arr[0][2] + 2*arr[1][2] + arr[2][2];
                pixel_x = Math.pow(pixel_y, 2);
                pixel = Math.sqrt(pixel_x + pixel_y);

                out.putPixel(x, y, pixel);
            }
        }
        out = rescale(out);
        return out;
    }
```