```
package com.perras.riemmanzeta;

import java.text.DateFormat;
import java.util.Date;

import android.os.AsyncTask;
import android.os.Bundle;
import android.app.Activity;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends Activity {

	private TextView consola;
	private EditText entrada;
	private Date inicio;
	private Date fin;
	private CalculoAsincrono[] calculo;
	private static final int NUMBER_THREADS = 8;
	private long zeroes = 0;
	private int hilosFinished;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);
		consola = (TextView) findViewById(R.id.salidaDeTexto);
		entrada = (EditText) findViewById(R.id.entradaX0);
	}

//	@Override
//	public boolean onCreateOptionsMenu(Menu menu) {
//		// Inflate the menu; this adds items to the action bar if it is present.
//		getMenuInflater().inflate(R.menu.main, menu);
//		return true;
//	}

	public static String fechaToString(Date fecha) {
		return DateFormat.getTimeInstance().format(fecha)+" "+DateFormat.getDateInstance
().format(fecha);
	}

	public void startCalculus(View view) {
		consola.setText("");
		double entradaX0;
		try {
			entradaX0 = Double.parseDouble(entrada.getText().toString());
			if(entradaX0 < 0)
				throw new NumberFormatException("El numero no puede ser negativo");
		}
		catch(NumberFormatException e) {
			consola.append("Error, no se puede calcular con "+entrada.getText().toString()
+"\n");
			consola.append(e.getMessage());
			return;
		}
		calculo = new CalculoAsincrono[NUMBER_THREADS];
		double incrementoHilos = entradaX0/NUMBER_THREADS;
		hilosFinished = 0;
		zeroes = 0;
		//Lanzamos las secciones
		for(int i=0; i < NUMBER_THREADS-1; i++) {
			calculo[i] = new CalculoAsincrono(i, this, (double) Math.round
(incrementoHilos*i), (double) Math.round(incrementoHilos*(i+1)));
		}
		calculo[NUMBER_THREADS-1] = new CalculoAsincrono(NUMBER_THREADS-1, this,(double)
Math.round(incrementoHilos*(NUMBER_THREADS-1)), entradaX0);
		inicio = new Date();
		consola.append("Inicio: "+fechaToString(inicio)+"\n");
```

```
		for(int i = 0; i < NUMBER_THREADS; i++) {
			//Ahora sí lanzamos los hilos
			new Thread(calculo[i]).start();
		}
	}

	private class CalculoAsincrono implements Runnable {

		private double limit;
		private double t;
		private Activity act;
		private int thread;

		CalculoAsincrono(int num, Activity activity, double init, double limit) {
			this.limit = limit;
			this.t = init;
			act = activity;
			thread = num;
		}

		private void devuelve(int result) {
			final int resultado = result;
			this.act.runOnUiThread(new Runnable() {
				@Override
				public void run() {
					hilosFinished++;
					zeroes += resultado;
					Log.d("FIN", "Soy hilo "+thread+" y he encontrado "+resultado);
					if(hilosFinished == NUMBER_THREADS) {
						fin = new Date();
						consola.append("Ceros encontrados: "+zeroes+"\n");
						consola.append("Fin:    "+fechaToString(fin)+"\n");
						consola.append("Tiempo transcurrido: "+((fin.getTime()-
inicio.getTime())/1000.0)+"s\n");
					}
				}
			});
		}

		@Override
		public void run() {
			double z, z2 = 0.0;
			int numzeros = 0;

			Log.d("InBackground", "Arranco entre "+t+" y "+limit);

			double increment = RiemmanFunctions.INCREMENTS[0];
			int increment_index = 0;
			for(int i = 0; i < RiemmanFunctions.INCREMENTS_INDEX.length; i++) {
				if(t > RiemmanFunctions.INCREMENTS_INDEX[i]) {
					increment = RiemmanFunctions.INCREMENTS[i+1];
					increment_index = i+1;
				}
			}

			while (t<limit) {
				z = RiemmanFunctions.zeta(t);
				if (((z < 0.0) && (z2 > 0.0)) || ((z > 0.0) && (z2 < 0.0))) {
					//zero found
					numzeros++;
				}
				z2 = z;
				t += increment;
				for(int i = increment_index; i <
RiemmanFunctions.INCREMENTS_INDEX.length; i++) {
					if(t > RiemmanFunctions.INCREMENTS_INDEX[i]) {
						increment = RiemmanFunctions.INCREMENTS[i+1];
						increment_index = i+1;
					} else {
						break;
					}
				}
```

```
                        }
                        z = RiemmanFunctions.zeta(limit);
                        if (((z < 0.0) && (z2 > 0.0)) || ((z > 0.0) && (z2 < 0.0))) {
                                //zero found
                                numzeros++;
                        }
                        this.devuelve(numzeros);
                }

        }

}
```

```
package com.perras.riemmanzeta;

public class RiemmanFunctions {

	public static double[] INCREMENTS = { 1, 0.05, 0.03, 0.01 };
	public static double[] INCREMENTS_INDEX = { 111, 10000, 200000 };

	public static double theta(double t) {
		double tmpThetha = ((t / 2.0 * Math.log(t / (2 * Math.PI))) - t / 2.0 - Math.PI / 8.0
+ 1 / (48 * t));
		return tmpThetha;
	}

	public static double R(int i, double t, double p) {
		double tmpR = 0.0;
		tmpR = Math.pow(-1, i - 1);
		tmpR = tmpR * Math.pow((t / (2.0 * Math.PI)), -1.0 / 4.0);
		tmpR = tmpR * (Math.cos(2 * Math.PI * (p * p - p - 1.0 / 16.0)) / Math.cos(2.0 *
Math.PI * p));

		return tmpR;
	}

	public static double zeta(double t) {
		int limit = (int) Math.floor(Math.sqrt(t / (2.0 * Math.PI)));
		double p = Math.sqrt(t / (2.0 * Math.PI)) - limit;

		double sum = 0.0;
		for (int i = 1; i <= limit; i++) {
			sum += 1.0 / Math.sqrt(i) * Math.cos(theta(t) - t * Math.log(i));
		}
		sum = 2.0 * sum;
		sum += R(limit, t, p);

		return sum;
	}
}
```