```
/*-----------------------------------------------------------------------------*
 * File Name: Value of Pi.c                                                    *
 * Creation: ER 6/26/2001                                                      *
 * Purpose: Origin C file to compute value of Pi using Monte Carlo technique   *
 * Copyright (c) OriginLab Corp. 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007 *
 * All Rights Reserved                                                         *
 *                                                                             *
 * Modification Log:                                                           *
 *-----------------------------------------------------------------------------*/

////////////////////////////////////////////////////////////////////////////////
// you must include this header file for all Origin built-in functions and classes
#include <origin.h>
//
////////////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////////////
// start your functions here

// Monte Carlo simulation example to compute the value of Pi

void ValueOfPi()
{
    // map datasets to Origin worksheet columns
    Dataset dsInX("Data1_InX");         // holds x pos of hits inside circle
    Dataset dsInY("Data1_InY");         // holds y pos of hits inside circle
    Dataset dsOutX("Data1_OutX");           // holds x pos of hits outside circle
    Dataset dsOutY("Data1_OutY");           // holds y pos of hits outside circle
    Dataset dsNThrow("Data1_nThrow");       // total hits
    Dataset dsPiVal("Data1_PiVal");     // value of pi from hit ratio

    // zero counter for hits inside of circle
    int iCircle = 0;

    // use timer to seed random number generator
    uint wTick =GetTickCount();
    double dR = rnd(wTick);

    // display wait cursor and accept escape key to interrupt computation
    waitCursor  cur;

    // clear data display
    SetDataDisplayText("Computing...hit Esc to stop.");

    // start simulation - run for 1000 total hits
    for(int iTotal=1; iTotal<10001 && cur.CheckEsc()==false; iTotal++)
    {
        // generate x,y random pair between -1 and +1
        double dRx = 1 - 2 * rnd();
        double dRy = 1 - 2 * rnd();
        // compute distance from origin to determine if within circle
        double dDist = sqrt( dRx^2 + dRy^2 );

        if(dDist <= 1)                      // is inside circle
        {
            iCircle += 1;
            dsInX.SetSize(iCircle);         // plot point inside circle
            dsInY.SetSize(iCircle);
            dsInX[iCircle - 1] = dRx;
            dsInY[iCircle - 1] = dRy;
        }
        else                                // is not inside circle
        {

            dsOutX.SetSize(iTotal - iCircle);   // plot point outside circle
            dsOutY.SetSize(iTotal - iCircle);
            dsOutX[iTotal-iCircle - 1] = dRx;
            dsOutY[iTotal-iCircle - 1] = dRy;
        }

        // now compute pi
        double dPiValue = 4.0 * iCircle / iTotal;

        // update datasets
        dsNThrow.SetSize(iTotal);
```

```
        dsPiVal.SetSize(iTotal);
        dsNThrow[iTotal - 1] = iTotal;
        dsPiVal[iTotal - 1] = dPiValue;

        // update both graphs periodically based on the value of the number of shots
        int iPlot = log10(iTotal);
        iPlot = 10^iPlot;
        if ( (iTotal % iPlot) == 0) LT_execute("page.active=1; plot -l; page.active=2; plot -l;
    }

    // update data display
    if(cur.CheckEsc()) SetDataDisplayText("Computation interrupted!");
    else SetDataDisplayText("Done!");
}
```