Миллер Дейв
Бриджес Ричард

ВИЗУАЛИЗАЦИЯ РЕШЕНИЙ ЛОГИСТИЧЕСКОГО УРАВНЕНИЯ, МНОЖЕСТВ МАНДЕЛЬБРОТА И ЖЮЛИА В EXCEL И VISUAL BASIC

В настоящей статье рассматриваются способы изображения множеств с помощью различных программных средств. Сначала мы изучаем логистическое уравнение, используя современные средства управления рабочим листом (Excel worksheet) – «движки» и макросы, управляемые кнопками. Затем те же средства применяются к множествам Мандельброта и в заключение рассматриваются образы множеств Мандельброта и Жюлиа, полученные с высоким разрешением благодаря использованию менее известных, но гораздо более эффективных алгоритмов.

ЛОГИСТИЧЕСКОЕ УРАВНЕНИЕ

В 1976 году Р. Мэй [1] настоятельно рекомендовал, чтобы знакомство с логистическим уравнением происходило на ранней стадии математического образования. Правда тогда только калькуляторы были доступны широкой публике.

Теперь же, имея изощренные программные средства, мы можем исследовать это уравнение. Логистическое уравнение (1), названное так Verhulst'ом в 40-х годах 19-го века [3] представляет собой модель изменения численности популяции, где p_n – относительная величина популяции $p_n = P_n/N$, а P_n – абсолютная величина популяции к моменту n и N – максимальная популяция. Р. Мэй использовал эквивалентную модель (2), где A – константа.

$$p_{n+1} = p_n + r \cdot p_n(1 - p_n) \qquad (1)$$

$$x_{n+1} = A \cdot x_n(1 - x_n) \qquad (2)$$

Существует обширная литература, посвященная поведению этой модели (Peitgen et al, 1992). Мы предлагаем здесь три метода, использующих рабочий лист Excel'а. Заинтересовавшиеся могут использовать эти методы для самостоятельного изучения модели.

МОДЕЛИРОВАНИЕ В EXCEL

При открытии рабочего листа Excel'а с именем logisticterms (Miller, 2001) подсчитываются 500 итераций уравнения (2) и демонстрируется график зависимости между x_n и n. Значение параметра A связано с движком и может меняться от 0 до 4 с шагом 0.001. Другой движок связан с начальным значением x_0, которое меняется от 0 до 1 с шагом 0.001. Кнопка, связанная с макросом, позволяет пользователю увидеть больше итераций, сгруппированных по 500, и тем са-

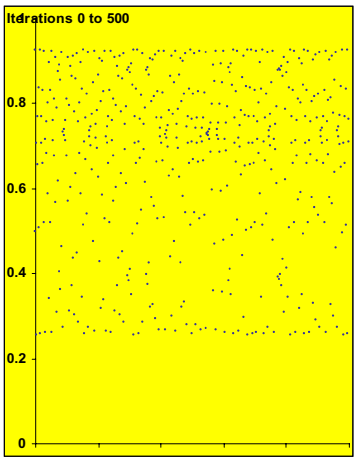

Диаграмма 1. *Хаотическое поведение при A = 3.701.*

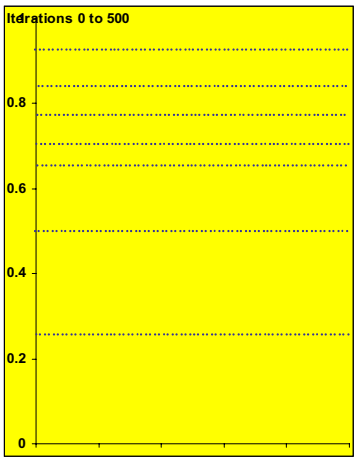

Диаграмма 2. *Цикл с периодом 7 при A = 3.702.*

мым увидеть, стабилизируются ли итерации при больших значениях n. Таким образом можно изучать поведение модели при разных значениях A и x_0. Диаграммы 1 и 2 показывают эффект изменения A от 3.701 до 3.702 (x_0 = 0.5)

Рабочий лист Excel'а с именем logisticcobweb (Miller, 2001) представляет те же результаты в другой форме – в форме паутинообразной диаграммы. Здесь строятся графики $y = Ax(1–x)$ и $y = x$, а также отмечаются и объединяются точки (x_n, x_{n+1}), (x_{n+1}, x_{n+1}), n = 0,1,2,... Это придает диаграмме вид паутины.

Рабочий лист Excel'а с именем logisticconsecutive (Miller, 2001) демонстрирует третий способ представления результатов. Те же результаты показаны на диаграммах 5 и 6.

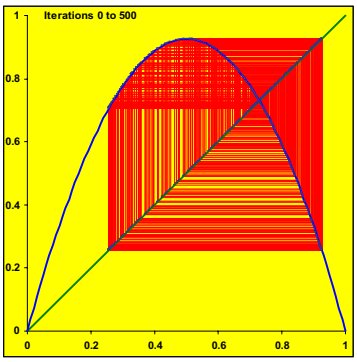

Диаграмма 3. *Хаотическое поведение при A = 3.701.*

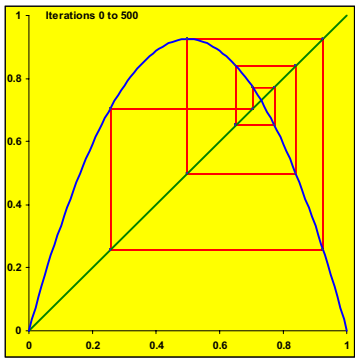

Диаграмма 4. *Цикл с периодом 7 при A = 3.702.*

МНОЖЕСТВО МАНДЕЛЬБРОТА

Множество Мандельброта было открыто только в 1982 и с тех пор изображения самого множества, его частей и связанного с ним множества Жюлиа стали весьма популярны.

Множество Мандельброта определяется с помощью функции $f(z,c) = z^2 + c$, где z и c – комплексные числа.

Если $z = z_r+iz_i$ и $c = c_r+ic_i$, то

$$f(z,c) = (z_r^2 - z_i^2 + c_r) + i(2z_r z_i + c_i) \qquad (3)$$

Последовательность (орбита) значений z получается посредством итераций

$z_1 = f(z_0, c)$, $z_2 = f(z_1, c)$, ...,

$z_n = f(z_{n-1}, c)$, ...

Очевидно, орбита зависит от начального значения z_0 и от значения константы c. При этом обнаруживаются два явления. Либо z_n стремится к бесконечности (обычно с большой скоростью), либо стремится к фиксированному числу или к предельному циклу, состоящему из конечного числа точек.

Множество Мандельброта – это множество тех значений c, для которых при $z_0 = 0$ последовательность z_n не стремится к бесконечности. Используя возможности Excel'а, можно изобразить любую часть множества Мандельброта. На диаграмме 7 показан рабочий лист «квадрат», состоящий из 441 квадратной ячейки, где каждый квадрат представляет точку комплексной плоскости. Исходная область включает множество Мандельброта, где z_r изменяется от –1.5 до 0.5, а z_i – от –1 до 1. Желтый (светло-серый) квадрат представляет число $–0.1+0.9i$, черный – $0+0i$, а красный (темно-серый) – $0.5+i$.

Применим итерации, положив $c = 0.5+i$. Таблица 1 показывает, что модуль z превосходит 5000 после 6 итераций.

На рабочем листе подсчитываются 50 итераций для каждой точки из заданного диапазона и определяется число ите-

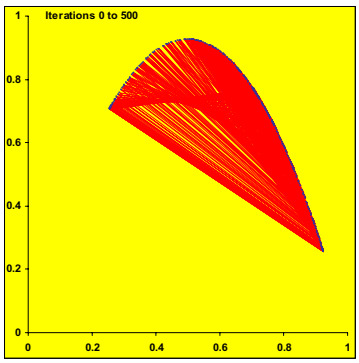

Диаграмма 5

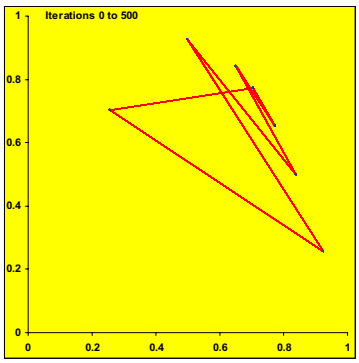

Диаграмма 6

№ итерации	z_r	z_i	$\|z\|$
0	0	0	0
1	0.5	1	1.12
2	–0.25	2	2.02
3	–3.44	0	3.44
4	12.32	1	12.36
5	151.19	25.63	153.35
6	22203.04	7752.05	23517.43

Таблица 1

раций, при котором модуль z превзойдет заданное число (здесь 5000). Предполагается, что если это не произойдет за 50 итераций, то не произойдет и при большем числе итераций. Числа 50 и 5000 выбраны произвольно. Результаты расчетов с этими параметрами приведены на диаграмме 8.

Для раскраски диаграммы 8 использованы 4 цвета (в статье – оттенки серого). Точки множества Мандельброта окрашены черным цветом. Точки, для которых расходимость ($|z_n|>5000$) обнаруживается при $n<17$, выкрашены красным (темно-серым); если же это происходит при $17\le n\le 33$ или $34\le n\le 49$, то – желтым (светло-серым) или белым, соответственно. Выбор 4-х цветов связан со свойством рабочего листа (Microsoft Excel 97), которое позволяет связывать цвет отдельной ячейки с выполнением определенных условий (таких условий может быть только 3, что дает 4 цвета).

ИЗОБРАЖЕНИЯ С ВЫСОКИМ РАЗРЕШЕНИЕМ

Наиболее распространенные «стандартные» алгоритмы построения образов множества Мандельброта плохо работают при больших увеличениях, необходимых для демонстрации самых интригующих явлений, связанных с такими множествами. Здесь мы представляем программу, реализующую менее известный, но гораздо более эффективный алгоритм.

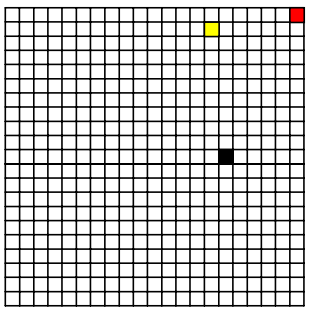

Диаграмма 7

Множество Мандельброта – это множество значений c, для которых последовательность итераций z_n не расходится при $z_0 = 0$. Множество Жюлиа для заданного параметра c – это множество тех z_0, при которых последовательность итераций z_n не расходится. Таким образом существует только одно множество Мандельброта и бесконечно много множеств Жюлиа. С этими простыми определениями связано большое количество интереснейших явлений, которых мы лишь слегка касаемся в данной статье.

Чтобы создать изображение одного из этих множеств, нужно установить соответствие между массивом пикселей на экране компьютера и множеством комплексных чисел z. Условимся, что x-координата на экране представляет вещественную часть числа z, а y-координата – мнимую часть. Соответствующий программный код для среды программирования Visual Basic 5 приведен в листинге 1. В других средах могут быть отличия в деталях, но принципы сохраняются.

Первая секция листинга содержит объявления переменных. Те, что участвуют в итерациях, объявлены как **Double** для сохранения точности, иначе изображения могут потерять детали при большом увеличении. Подпрограмма **Form_Load** игра-

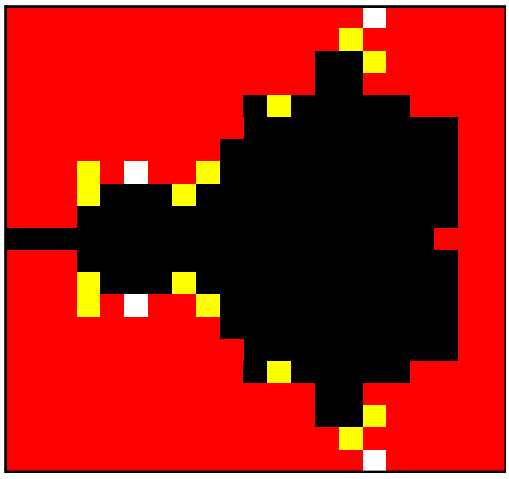

Диаграмма 8

ет роль головной программы. Подпрограмма **SetScreen** инициализирует экран Visual Basic'а с началом координат в левом нижнем углу и координатами пикселей в диапазоне от **0** до **srange**. Подпрограмма **SetFractal** инициализирует переменные, связанные с создаваемым изображением: **zrmid**, **zimid** и **zrange** описывают прямоугольник на комплексной плоскости, соответствующий экрану, а **palno** определяет используемую раскраску. Остальные переменные будут использованы позже. Подпрограмма **SetColours** сохраняет в массиве **palcol()** цвета, которые будут использованы для окраски разных частей образа.

Остальные короткие функции и подпрограммы:

– **NotPlottedYet** проверяет, был ли изменен цвет пикселя по сравнению с начальным; такая проверка упрощает дальнейшую работу с массивом пикселей;

– **PlotPixel** окрашивает пиксель;

– **CheckForStop** необходима в многозадачной среде Windows, чтобы иметь возможность прервать работу программы, когда приходится достаточно долго ждать завершения работы; подпрограмма **StopButton** реализует этот стоп-сигнал.

Подпрограмма **Form_Load** просматривает все пиксели, преобразуя экранные координаты в координаты комплексной плоскости, и затем вызывает подпрограмму **ComputeColour**, которая выполняет математические преобразования. Таким образом **Form_Load** можно рассматривать как скелет любой программы, рисующей фракталы различных типов. Нужно только внести соответствующие изменения в **ComputeColour**. Пользователь может увидеть почти с первых шагов, как будет выглядеть изображение. Оно вычисляется с двойным разрешением, определяемым переменной *s*.

Математическая часть задачи выполняется в **ComputeColour**. Здесь экран представляет часть комплексной плоскости параметра *c*. Числа $z_n = f(z_{n-1}, c)$, $n = 1, 2, 3, \ldots$, $z_0 = 0$, полученные из уравнения (3) хранятся в массивах zrorb() и ziorb() (для стандартного алгоритма это не является необходимым). Цикл **Do loop** работает, пока либо число итераций, либо модуль z_n не превысят заданных границ. В первом случае *c* принадлежит множеству Мандельброта, и точка окрашивается одним цветом (обычно черным), а во втором – не принадлежит и окрашивается другим цветом. В простейшем варианте (palno = 1) все точки, не принадлежащие множеству Мандельброта, окрашены одним цветом. Однако, как будет показано далее, целесообразней сделать цвет зависящим от числа итераций, проделанных до того, как достигнуто предельное значение модуля z_n (**maxmodzsq**). Это приводит к появлению множества цветных контуров вокруг множества Мандельброта. Простейший выбор – чередование цветов контуров (рисунок 1).

Выбор предельного числа итераций (**maxiter**) зависит от степени увеличения. Если **maxiter** мало, то при большом увеличении граница изображаемого множества будет искажаться. Значение maxiter = 256 годится только для небольшого увеличения, а maxiter = 1024 подходит для любого увеличения, хотя и замедляет работу программы.

В листинге 3 содержатся различные варианты работы подпрограммы **SetFractal**, позволяющие изучать эффекты различных увеличений и цветовых схем.

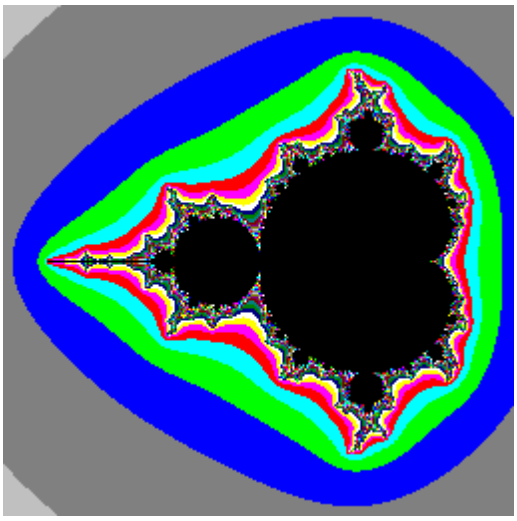

Рисунок 1

Пусть ваш код правильно изображает множество Мандельброта и окружающие его контуры. Теперь измените код, чтобы был реализован **Case 2** (двухцветная схема – palno = 1). В этом случае все еще используется стандартный алгоритм, и можно увидеть, насколько он неэффективен (рисунок 2) по сравнению с более мощным алгоритмом, реализованным в **Case 3**. В **Case 2** теряется тонкая структура границы изображаемого множества (рисунок 3).

В чем же заключается трудность и как ее преодолевает этот более мощный алгоритм? Дело в том, что множество Мандельброта *M* состоит из чрезвычайно тонких нитевидных фрагментов. Такие фрагменты исчезают в любом двухцветном изображении, если разрешение не очень велико. Решение состоит в том, чтобы оценить расстояние от любой точки *c*, не принадлежащей *M* до ближайшей точки этого множества. Если это расстояние меньше одного пикселя, то точку с экрана следует окрашивать цветом границы *M*. Метод оценки расстояния был опубликован в [5]. Он основан на глубоком анализе последовательности итераций в [2]. Они доказали, что расстояние от *c* до *M* меньше, чем $2|z_n|\log|z_n|/|z_n'|$, где *n* – номер первой итерации, для которой $|z_n|$ превосходит некоторое заданное значение (в программе это **maxmodzsq** должно быть весьма большим), а z_n' – производная z_n по *c*. Дифференцируя уравнение $z_{k+1}=z_k^2+c$ по *c*, получаем рекуррентное соотношение для z_n'

$$z_{k+1}'=2z_kz_k'+1,\ \ k=0,1,2,...,\ \ z_0'=0 \quad (4)$$

Упомянутый выше метод основан на существовании потенциала в области комплексной *c*-плоскости вне множества *M*. Добавим, что контуры, рисуемые данной программой, являются эквипотенциальными линиями для электростатического поля заряда, распределенного на множестве *M*. Подробности, а также многие интересные следствия можно найти в [4].

Возвращаясь к листингу 2, заметим, что для использования соотношения (4) необходимо хранить в памяти все значения z_k, $0<k<n$; они сохраняются в массивах **zrorb()** и **ziorb()**. Если оценка расстояния предусмотрена (переменная **DEMflag**), то нужно вычислять z_n', помещая вещественную и мнимую части в **zdr** и **zdi**. Так как величина z_n' может быть очень большой вблизи множества *M*, нужно быть уверенным в отсутствии переполнения (в этом причина сравнения с **maxmodzsq**). Величина **delta** (ширина пикселя) подсчитывается в конце подпрограммы **SetFractal**, так как она зависит от степени увеличения. Дополнительная переменная **dfactor** (поправочный множитель) обеспечивает тонкую настройку, так как результат очень чувствителен к величине **delta**. Если **Mdist** (оценка расстояния между *c* и *M*) не превышает эту величину, то переменной **colour** присваивается значение 1 (черный), что обозначает точку, удаленную от *M* не более, чем на ширину пикселя, но не принадлежащую *M*. Если

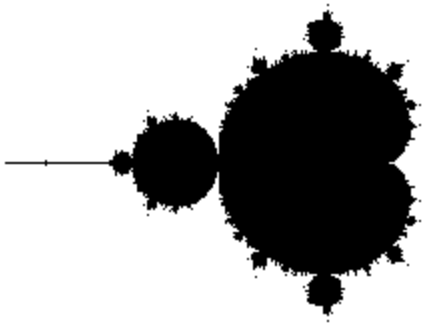

Рисунок 2

Рисунок 3

Рисунок 4

colour = 0 соответствует серому цвету, а colour = 1 – черному (как в **SetColours**), то при достаточно большом увеличении в правильно выбранном месте видны малые копии *M*, окрашенные серым цветом, сквозь которые просвечивают черные нитевидные структуры (**Cases** 2-15 в **SetFractal** и рисунок 4). Пиксели, не принадлежащие *M* или окрестности границы *M*, можно окрашивать стандартным алгоритмом.

Множества Жюлиа вычисляются аналогичным образом, так что небольшие поправки, зависящие от значения флага *j*, достаточны, чтобы вместо множества Мандельброта вычислялось множество Жюлиа. Основное изменение состоит в том, что значение *c* остается фиксированным в ходе вычислений, и на экране показано изменение *z*, а не *c*. Стандартный алгоритм не нуждается в изменении, а в методе оценки расстояния теперь нужно для получения рекуррентного соотношения дифференцировать по z_0, а не по *c*.

$$z'_{k+1} = 2z_k z'_k \ , k = 0,1,2,..., \quad z'_0 = 1 \qquad (5)$$

Case 18 в **SetFractal** дает с большим увеличением изображение части множества Жюлиа вокруг точки $z = c$, где *c* взято из малой копии множества *M*, построенной в **Case** 15. Изменяя увеличение, можно увидеть, что множество Жюлиа для *c*, взятого из окрестности границы множества *M*, само выглядит как эта часть *M*.

В заключение нужно предупредить о некоторых ограничениях, присущих данной программе. Наиболее серьезное из них связано с невозможностью изменять степень увеличения интересующего нас участка простым щелчком по экрану. Написать соответствующий код несложно, однако он будет существенно зависеть от используемой среды. Менее серьезным является неэффективность вычисления множества Жюлиа – значения *z* быстро сходятся к предельному значению или предельному циклу, однако вычисления продолжаются пока не проделано **maxiter** итераций. Это может быть легко исправлено. И, наконец, отсутствуют какие-либо средства сохранения результатов, но если вы работаете в Visual Basic, то можете связать управляющую кнопку с подпрограммой, помещенной в конце листинга 3. Нажатие кнопки посылает изображение в **ClipBoard** в виде **bitmap**. Имеется также большой простор для экспериментирования с цветовыми схемами; другая возможность появляется при установке значения palno = 3.

Литература.

1. May R.(1976) Simple mathematical models with very complicated dynamics, Nature, 261, pp.459-467.
2. Milnor J., Thurston W.(1989) Selfsimilarity and hairiness in the Mandelbrot set, Lecture Notes, Pure and Appl. Math.,114.
3. Peitgen H-O., Jurgens H. & Saupe D.(1992) Fractals for the Classroom: Part One, Introduction to Fractals and Chaos, (Springer-Verlag).
4. Peitgen H-O., Richter P.H.(1986) The Beauty of Fractals, (Springer-Verlag).
5. Peitgen H-O., Saupe D. (1988) The Science of Fractal Images, (Springer-Verlag).

Приложение.

Listing 1

```
Dim ssize, sxmin, symin, srange, scol, sbackcol As Double
Dim sx, sy, s, stopflag
Dim palno, mandelcol, nearmandelcol, notmandelcol, palcol(1024)
Dim iter, maxiter, maxmodsq, maxdmodsq, delta, pi, DEMflag, j, fractalno
Dim zrmin, zimin, zrange, cr As Double, ci As Double, zr As Double
Dim zi As Double, t As Double
Dim zrorb(1024) As Double, ziorb(1024) As Double, zdr As Double
Dim zdi As Double

Private Sub Form_Load()
    SetScreen
    SetFractal 1
    SetColours
    s = 256: stopflag = False
    While s > smin And Not CheckForStop
    s = s / 2
        For sy = symin To symin + srange Step s
            For sx = sxmin To sxmin + srange Step s
                If NotPlottedyet(sx, sy) Then
                    zr = zrmin + (sx - sxmin) * zrange / srange
                    zi = zimin + (sy - symin) * zrange / srange
                    scol = ComputeColour(zr, zi)
                    PlotPixel sx, sy, scol
                End If
            Next sx
            If CheckForStop Then Exit For
        Next sy
    Wend
End Sub

Public Sub SetScreen()
    ssize = 3900:  sxmin = 0: symin = 0: srange = 256: smin = 1
    screen.Height = ssize: screen.ScaleHeight = -srange
    screen.ScaleTop = srange
    screen.Width = ssize: screen.ScaleWidth = srange: MainForm.Show
End Sub

Public Sub SetFractal(fractalno)
    maxmodzsq = 1E+21: maxdmodzsq = 1E+60: pi = 4 * Atn(1)
    Select Case fractalno
    Case 1: j = 0: zrmid = -0.75: zimid = 0: zrange = 3: maxiter = 64
        palno = 0: DEMflag = False: dfactor = 1
    End Select
    delta = dfactor * 0.71 * zrange / srange: zrmin = zrmid - zrange / 2
    zimin = zimid - zrange / 2
End Sub

Public Sub SetColours()
    sbackcol = vbCyan: screen.BackColor = sbackcol
    mandelcol = RGB(128, 128, 128): nearmandelcol = vbBlack
    notmandelcol = vbWhite
    palcol(0) = mandelcol: palcol(1) = nearmandelcol
    palcol(2) = notmandelcol
    Select Case palno
        Case 0
        For c = 3 To maxiter
            palcol(c) = QBColor(1 + (c Mod 15))
        Next c
    Case 1
        For c = 2 To maxiter
```

```
            palcol(c) = notmandelcol
        Next c
    Case 2
        m = 12
        For c = 2 To m
            s = Log(c) / Log(m)
            palcol(c) = RGB(256 * s, 256 * s, 256 * s)
        Next c
        For c = m + 1 To maxiter
            palcol(c) = notmandelcol
        Next c
    End Select
End Sub

Public Function NotPlottedyet(sx, sy)
    If screen.Point(sx, sy) = sbackcol Then NotPlottedyet = True _
    Else NotPlottedyet = False
End Function

Public Sub PlotPixel(sx, sy, scol)
    screen.PSet (sx, sy), palcol(scol)
End Sub

Public Function CheckForStop()
    DoEvents
    If stopflag Then CheckForStop = True Else CheckForStop = False
End Function

Private Sub StopButton_Click()
    stopflag = True
End Sub
```

Listing 2

```
Public Function ComputeColour(zr As Double, zi As Double)
    If j = 0 Then cr = zr: ci = zi: zr = 0: zi = 0
    iter = 0: zrorb(0) = zr: ziorb(0) = zi
    Do
        iter = iter + 1
        t = zr * zr - zi * zi + cr: zi = 2 * zr * zi + ci: zr = t
        zrorb(iter) = zr: ziorb(iter) = zi
        'Comment: not needed for simple algorithm
    Loop Until iter >= maxiter Or zr * zr + zi * zi > maxmodzsq
    If iter >= maxiter Then ComputeColour = 0
    If zr * zr + zi * zi > maxmodzsq Then
        ComputeColour = iter
    'Comment: standard simple algorithm stops here
        If DEMflag Then
            zdr = 0: zdi = 0: i = 0: If j = 1 Then zdr = 1
            While zdr * zdr + zdi * zdi < maxdmodzsq And i <= iter - 1
                t = 2 * (zdr * ziorb(i) + zdi * zrorb(i))
                zdr = 2 * (zdr * zrorb(i) - zdi * ziorb(i))
                If j = 0 Then zdr = zdr + 1
                zdi = t: i = i + 1
            Wend
        If zdr * zdr + zdi * zdi > maxdmodzsq Then
            ComputeColour = 1
        Else
            modz = Sqr(zr * zr + zi * zi)
            modzd = Sqr(zdr * zdr + zdi * zdi)
            Mdist = 2 * modz * Log(modz) / modzd
            If Mdist < delta Then ComputeColour = 1
        End If
```

```
        End If
    End If
End Function
```

Listing 3

```
Case 2: j = 0: zrmid = -0.75: zimid = 0: zrange = 3: maxiter = 64
    palno = 1: DEMflag = False: dfactor = 1
Case 3: j = 0: zrmid = -0.75: zimid = 0: zrange = 3: maxiter = 64
    palno = 1: DEMflag = True: dfactor = 1
Case 4: j = 0: zrmid = -0.75: zimid = 0: zrange = 3: maxiter = 64
    palno = 0: DEMflag = True: dfactor = 1
Case 5: j = 0: zrmid = -0.75: zimid = 0: zrange = 3: maxiter = 64
    palno = 2: DEMflag = True: dfactor = 1
Case 6: j = 0: zrmid = 0: zimid = -1: zrange = 0.1: maxiter = 64
    palno = 1: DEMflag = True: dfactor = 1
Case 7: j = 0: zrmid = -0.05: zimid = -1: zrange = 0.05: maxiter = 256
    palno = 1: DEMflag = True: dfactor = 1
Case 8: j = 0: zrmid = 0.2849758: zimid = -0.0112716: zrange = 1
    maxiter = 64: palno = 1: DEMflag = True: dfactor = 0.5
Case 9: j = 0: zrmid = 0.2849758: zimid = -0.0112716: zrange = 0.1
    maxiter = 128: palno = 1: DEMflag = True: dfactor = 0.5
Case 10: j = 0: zrmid = 0.2849758: zimid = -0.0112716: zrange = 0.01
    maxiter = 256: palno = 1: DEMflag = True: dfactor = 0.5
Case 11: j = 0: zrmid = 0.2849758: zimid = -0.0112716: zrange = 0.001
    maxiter = 1024: palno = 1: DEMflag = True: dfactor = 0.5
Case 12: j = 0: zrmid = 0.2849758: zimid = -0.0112716: zrange = 0.0001
    maxiter = 1024: palno = 1: DEMflag = True: dfactor = 0.5
Case 13: j = 0: zrmid = 0.2849758: zimid = -0.0112716: zrange = 0.00001
    maxiter = 1024: DEMflag = True: palno = 1: dfactor = 0.5
Case 14: j = 0: zrmid = 0.2849758: zimid = -0.0112716: zrange = 0.000001
    maxiter = 1024: DEMflag = True: palno = 1: dfactor = 0.5
Case 15: j = 0: zrmid = 0.2849757: zimid = -0.0112718: zrange = 0.0000005
    maxiter = 1024: DEMflag = True: palno = 1: dfactor = 0.5
Case 16: j = 1: zrmid = 0: zimid = 0: zrange = 3: cr = 0: ci = 1
    maxiter = 256: DEMflag = True: palno = 1: dfactor = 0.5
Case 17: j = 1: zrmid = 0: zimid = 0: zrange = 3: cr = 0.2849757
    ci = -0.0112718: maxiter = 1024: palno = 1: DEMflag = True
    dfactor = 0.5
Case 18: j = 1: zrmid = 0.2849757: zimid = -0.0112718: zrange = 0.0001
    cr = 0.2849757: ci = -0.0112718: maxiter = 1024: palno = 1
    DEMflag = True: dfactor = 0.5

Private Sub copybutton_Click()
    ClipBoard.Clear
     Clipboard.SetData screen.Image
End Sub
```

Richard Bridges,
King Edwards School, Birmingham, UK.
Dave Miller,
Department of Education,
Keele University, UK.