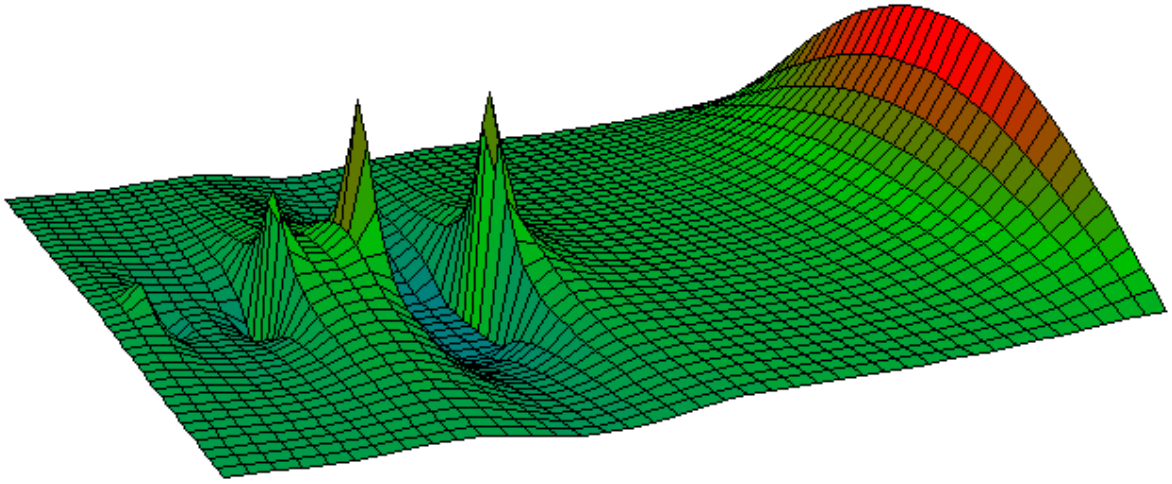
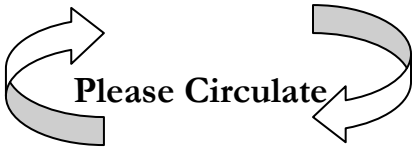
Journal of J

An interdisciplinary journal on J programming language and applications in science and technology.





MPM press

An open access Journal

ISSN: 2174-9280

Vol.1, No.2 April 2012

JoJ (J2-team)

J, a language for the science, technology and more.

Aims and scope

Journal of J is a **non professional and nonprofit** journal , involving a large research and users community in J programming language.

Journal of J is an interdisciplinary **journal** devoted to J and science and technology.

Journal of J aims to provide fast access to papers about science,technology and J.

Journal of J embodies the following principle:

Open Access: Knowledge is a public good. All readers have open access to reading and downloading papers. The simple and free access ensures maximum readership and high citation records for published papers.

Contact: journalofj at hotmail dot com

Style and Contents

Journal of J aims to cover all the main areas of science. Inevitably, articles in different areas are addressed at different audiences. Many of the articles submitted to the journal are standard technical pieces, addressed to a purely academic audience

To attract this variety of contributions ***Journal of J*** will contain the following areas:

- Mathematics: number theory, logic, calculus, algebra, arithmetic, algorithms and others…
- Physics: dynamical systems, chaos, fractals, disorder, statistical physics and others…
- Computer science, Visualization, Engineering, Computer Art, …

Visit:

http://sites.google.com/site/jforscience/

Edited by Mikel Paternain

Editorial

En este número contamos con unas interesantes contribuciones de fuerte componente matemático.

Como dialecto proveniente de APL, J es un lenguaje de marcada base matemática y especialmente útil para sintetizar ideas en esta área.

Al objeto de ir facilitando materiales de estudio, el equipo de *JoJ* ha preparado en este número una traducción del documento *J by example*.

Dado que la mayoría de las contribuciones en el lenguaje se distribuyen y dan a conocer en el foro de J (origen de JoJ) pocas contribuciones llegan a nuestra redacción todavía. No obstante esperamos que el nivel de las contribuciones siga aumentando.

Mikel Paternain (Editor)

mikelpater at Hotmail dot es

Creating Quasicrystals & Garage Doors via Canonical Projection

Cliff Reiter

Lafayette College, Department of Mathematics, Easton PA, 18042 U.S.A.

Quasicrystalline patterns contain much local symmetry, while not being periodic. In many ways they are counter-intuitive and yet intriguing [4]. As part of a project for visualizing Voronoi cells using raster techniques some students and I visualized some quasicrystalline tilings using a displacement technique [1].

The Voronoi cell around a point in a discrete collection of points is the region around the given point that at least as close to the given point than to any other point in the discrete collection. Thus, the Voronoi cell around the origin for the integer lattice in 3-space is the unit cube centered at the origin with coordinates less than or equal to one-half. I grew frustrated with the fact that references referred to canonical projection as a standard techniques for creating quasicrystalline tilings, [4], yet examples were scarce. I embarked on a project that led to an "Atlas of quasicrystalline tilings", [2], produced with J. I remain charmed by quasicrystals and when the time came to replace decaying garage doors, I built garage doors with quasicrystal patterns, see Figure 1.

The J scripts I used for [2] were complicated enough to make it difficult to share with the J community and J from more than a decade ago is now obsolete. While the J and mathematics may be a stretch, the goal of this note is to share with the J community a script to produce quasicrystals via canonical projections from the integer lattices in any dimension. Readers may enjoy exploring these fun patterns on their own.

Figure 1. Quasicrystalline Garage Doors.

1. A Brief Introduction to Canonical Projection

Canonical projection of a lattice in $\Re^n$ is based upon the idea of decomposing $\Re^n$ in terms of orthogonal spaces *E* and $E^\perp$ and projecting select portions of the integer lattice in $\Re^n$ onto *E*. In Figure 2, *E* and $E^\perp$ are perpendicular lines. In particular, some compact set, *C*, in $E^\perp$ is chosen. The "canonical" choice for *C* is the projection of Voronoi cell of the origin in $\Re^n$, relative to the lattice, as projected onto $E^\perp$.

The Voronoi cell around a point in a collection of points is the set of points in space that are at least as close to the given point as any other point in the collection. We also shift (translate) the points before we project and test them; one can equivalently describe the compact set *C* as shifted, but for convenience, we do our shift in $\Re^n$. The points that are projected onto *E* are the points of the lattice which map to *C* under shift and projection to $E^\perp$. In other words, the preimage of *C* corresponds to a cylinder in $\Re^n$, compact in some directions and unbounded in others. The points of the lattice in $\Re^n$ which are in that cylinder are projected onto *E*.

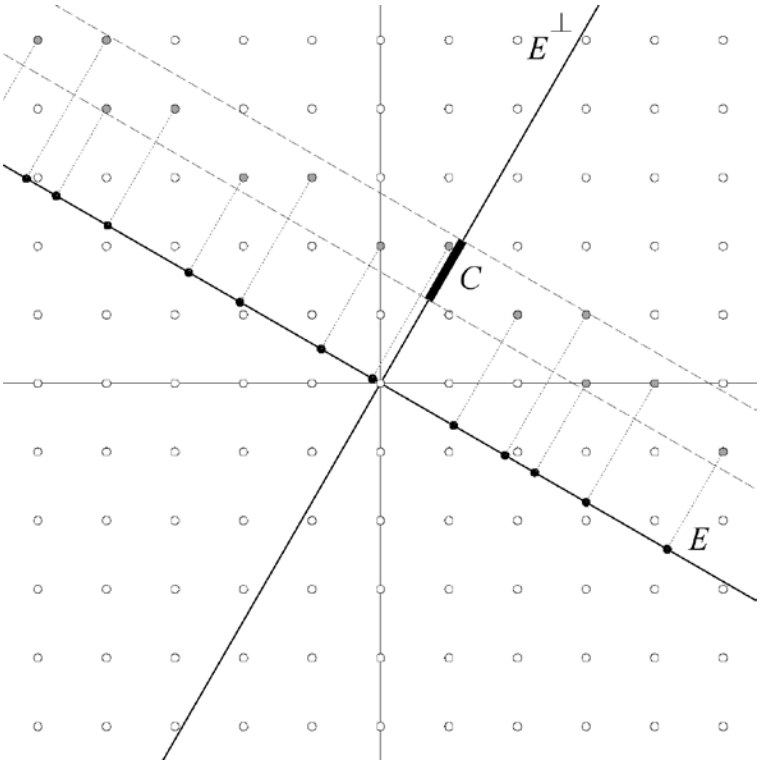


Figure 2. Canonical Projection from the Integer Lattice in 2-Space to a Line.

Figure 2 illustrates canonical projection in a simpler situation than we use, but it serves as a useful overview of the process. In the figure, the orthogonal spaces *E* and $E^{\perp}$ are the marked lines and together they generate $\Re^n$. A compact set *C* in $E^{\perp}$ is marked in black. The set *C* defines a cylinder in $\Re^n$ which lies between the dashed lines.

Some of the integer lattice points lie in the cylinder—these are marked as points with gray centers. Others lattice points lie outside the cylinder—these are marked as points with hollow centers. Those that are in the cylinder are projected onto E along dotted lines to points marked with black circles. The result of Canonical projection of the lattice is that point set in *E*.

The projections we use are not as simple. We are interested in the case when *E* is 2-dimensional and seek to connect the projected points with edges to give a tiling.

2. Initializing the Compact Set

We begin by loading scripts that will be used. The noun `p_tol` determines the tolerance for system solving. The verb `MZ` creates an identity matrix whose rows give

positive normal vectors to the faces of the Voronoi cube. The verb vor2 gives all the normals in a preferred order. There are utilities vmag for vector magnitude and unit for creating unit vectors. A verb mkU produces an orthogonal matrix used for the projections onto E and $E^{\perp}$.

```
require 'plot statfns'
require '~system/packages/math/linear.ijs'
p_tol=:1e_11
MZ=: =@i.
   MZ 4
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
mp=:+/ . *
vor2=: ,-@|.
   vor2 MZ 4
 1  0  0  0
 0  1  0  0
 0  0  1  0
 0  0  0  1
 0  0  0 _1
 0  0 _1  0
 0 _1  0  0
_1  0  0  0

vmag=:+/&.(*:"_)
unit=:% +/&.(*:"_)
```

```
mkU=: 3 : 0

unit"1&.|: (,./ +._12 o. (>:i.<:>.-:y)*/(2p1%y)*i.y),.,./2 o.
((2|y)}.1p1 0)*/i.y   NB. Line continued

)

   ]U=:mkU 3

 0.816497         0 0.57735

_0.408248  0.707107 0.57735

_0.408248 _0.707107 0.57735
```

The diagonal of a unit hypercube in n-space has length $\sqrt{n}$ and so the covering radius is half of that. The verb `cov_rad` gives the covering radius, `det` gives the determinant and `rmzrw` removes zero rows. The verb `ssgen` is a general system solver that deals with infinite solution sets. The adverb `aff_proj` does an affine projection.

```
cov_rad=: -:@%:

det=:-/ . *

rmzrw=: (+./@:|:@:(0:~:])#])


   ssgen=: 3 : 0

0 ssgen y

:

a=.p_tol gauss_jordan y,.x

a=.(1~:a *./ . = 0)#a

n=.{:$y

pp=. i.&1"1 a                          NB. pivot positions

if. pp +./ . = n do. '' return. end.  NB. inconsistent

npp=. pp-.~i.n                         NB. nonnpivot positions

ppfill=.pp}&(n#0)

x0=.ppfill {:"1 a
```

```
u=.ppfill"1&.|:-npp{"1 a
x0;u+(i.n)=/ npp
)
   1 ssgen ,:1 2
+---+--+
|1 0|_2|
|   | 1|
+---+--+
af_proj=: 1 : 0
:
'x0 U'=.x ssgen y
't  W'=.0 ssgen |:m mp U
(|:W),.(|:W) mp m mp x0
)
aff1_merge=:3 : 0"2
y=. rmzrw y
lc=.((i.>./)@:(+/@:(0:~:])){"1])(**|)>{: ssgen ,:+/}:|:y
lc mp y
)
```

Now these all get put together into a utility `init_pv` for initializing the functions `P2` and `P_2` that do the projection onto *E* and $E^{\perp}$ and a function `pvq` that determines whether an integer lattice point lies in the cylinder of interest.

```
   init_pv=: 3 : 0
U=:mkU y
UI=:%.U
UI_2=:2}.UI
```

```
P_2=:UI_2&mp"1

P2=:2&{. @(UI&mp)"1

vorn=:vor2 mgen=:MZ y

maggen=:vmag {.mgen

covrad=:cov_rad y

gcm=.((0:~:(]det@mp |:)@({&vorn))"1)pvorn=:3 comb #vorn

pvorn=:gcm#pvorn

gb=.3#-:maggen                          NB. dist to voronio

pvorn=:gb (UI_2 af_proj {&vorn)"1 pvorn

pvorn=:,/pvorn

pvorn=:pvorn*p_tol<|pvorn               NB. defuzz

pvorn=:((0: ~: {:"1)#])pvorn            NB. no 0 constants

pvorn=:(%vmag@}:)"1 pvorn               NB. scale

pvorn=:(<.@(0.5&+)@((%p_tol)&*)@:(}:"1) (}:@{.,>./@:(|@{:"1))/.
])pvorn    NB. max constant for parallel norm

pvq=:*./@:(>:&(-p_tol)@:(({:"1 pvorn)&-))@:|@:((}:"1

        pvorn)&mp)"1          NB. Line continued

$pvorn

)

   init_pv 5

10 4
```

So we see that 10 affine conditions in 3-space must be satisfied in order for a point in $E^{\perp}$ to be in the projected Voronoi cell. Before doing an example we need to mention that we offset the projected cell a bit and have found 6 useful types of offset. Offsets of zero, random numbers, the covering radius, and half the covering radius, in various coordinates all play useful roles. The verb `rand2` creates random numbers, `mkvec` creates the 6 types of offset, and `off` adds the projected offset in $E^{\perp}$. We see some lattice points are in the projected Voronoi cell and others are not.

```
   rand2=:(?. % ])@($&10000)
     mkvec=: 3 : 0
z=.,:0 0, rand2 y-2
z=.z,0 0,(rand2 y-3),covrad
z=.z,0 0,(rand2 y-3),-:covrad
z=.z,(0 $~ y-1),covrad
z=.z,(0 $~ y-1),-:covrad
z,0
)

   mkvec 5
0 0 0.6146 0.5755   0.2079
0 0 0.6146 0.5755  1.11803
0 0 0.6146 0.5755 0.559017
0 0      0      0  1.11803
0 0      0      0 0.559017
0 0      0      0        0

pars=: 4{mkvec 5

off=:(U mp pars)&+"1
   pvq P_2 off 0 0 0 0 0
1
   pvq P_2 off 0 0 0 0 1
0
```

3. Search for Good Lattice Points

The verb `sglat` does a brute force search for "good lattice points"; that is, lattice points that lie in the desired cylinder. Its right argument is the offset vector produced by `mkvec` (or other means). The first while loop finds at least one good point. Usually it would be expected to run zero times. The main while loop mitigates problems with memory.

Several potentially large nouns appear:

`tlat0` gives the initial lattice points to be tested,

`ntlat` gives the lattice points that need to be tested,

`tlat` gives the tested lattice points,

`glat` gives the good lattice points found,

`glat0` gives the good lattice points in this iteration.

Assuming `init_pv` was run as in the previous section, executing 10 `sglat 4{mkvec 5` results in 1483 good lattice points and that 39579 total lattice points were tested.

```
   sglat=: 3 : 0
6 sglat y
:
n=.#pars=:y
off=:(U mp pars)&+"1
$tlat0=:(,/^:(2:<#@$)^:_>{(#mgen)#<_1 0 1) mp mgen
$ntlat=:tlat0
mx=.0
while. (0=+/pvq@:P_2@:off tlat0)*.24>mx do.
  if. ({:pars)=-:covrad do. pars=:(-:@}:,{:)pars else. pars=:-
:pars end.
```

```
  off=:(U mp pars)&+"1
  mx=.mx+1
  end.
if. mx=24 do. 'ouch' return. end.
$glat=:tlat=:i.0 1
]bksz=:3^8
while. 0<#ntlat do.
  sz=:bksz <. # ntlat
  m=.pvq@:P_2@:off  ntlat0=:sz{.ntlat
  glat=:glat,nglat=:  m # ntlat0
  tlat=:tlat,ntlat0
  ntlat=:sz}.ntlat
  ntlat0=:i.0 1*$tlat
  for_ng. nglat do.
    nt0=: ng +"1 tlat0
    ntlat0=:ntlat0, nt0 #~ *./"1 (x+maggen) >:| P2 nt0
  end.
  ntlat0=:~.ntlat0
  ntlat0=:ntlat0 -. tlat
  ntlat0=:ntlat0 -. ntlat
  ntlat=:ntlat,ntlat0
  if. 0<+/m do. smoutput ($glat),($tlat),($ntlat) end.
end.
$glat
)
 10 sglat 4{mkvec 5
46 5 243 5 1755 5
```

```
131 5 1998 5 2835 5

286 5 4833 5 4725 5

501 5 9558 5 6345 5

726 5 15903 5 6614 5

1036 5 22464 5 8372 5

1250 5 29025 5 6400 5

1420 5 35425 5 3256 5

1479 5 38681 5 898 5

1483 5 39579 5 46 5

1483 5
```

4. Some Tilings

Now we turn to the problem of projecting those good lattice points onto the plane, connecting the appropriate edges and plotting the result.

```
   pedges=:3 : 0

k=.0

o=. vorn

z=.''

while. k<#y do.

  z=.z,<v,:"1"1 v+"1 o#~(o+"1 v=.k{y)e."1 _ y

  k=.k+1

end.

zzz=:z

P2@:off ; z

)

   $e=:pedges glat

5718 2 2
```

```
'color black' plot({."1;{:"1)e
```

The result is shown in Figure 3. Another example is generated by the following lines. The resulting projection from eight dimensions looks periodic at a glance but look again. The result is shown in Figure 4.

```
init_pv dim=:8

10 sglat 4{mkvec dim

$e=:pedges glat

'color black' plot({."1;{:"1)e
```

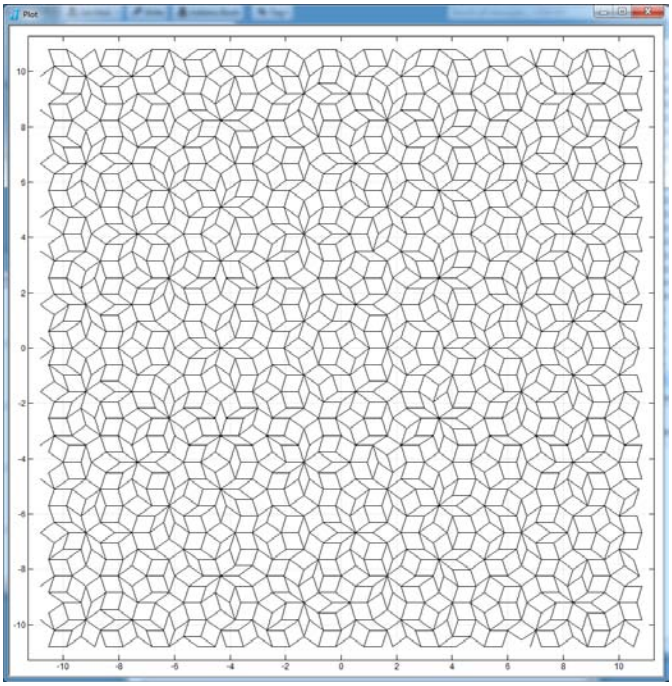


Figure 3. A Tiling via Canonical Projection from 5-Space.

Try other dimensions and offsets. Computations become difficult for high dimensions. Using 64-bit J gives more memory and reducing the window size (left argument of sglat) mitigate those problems.

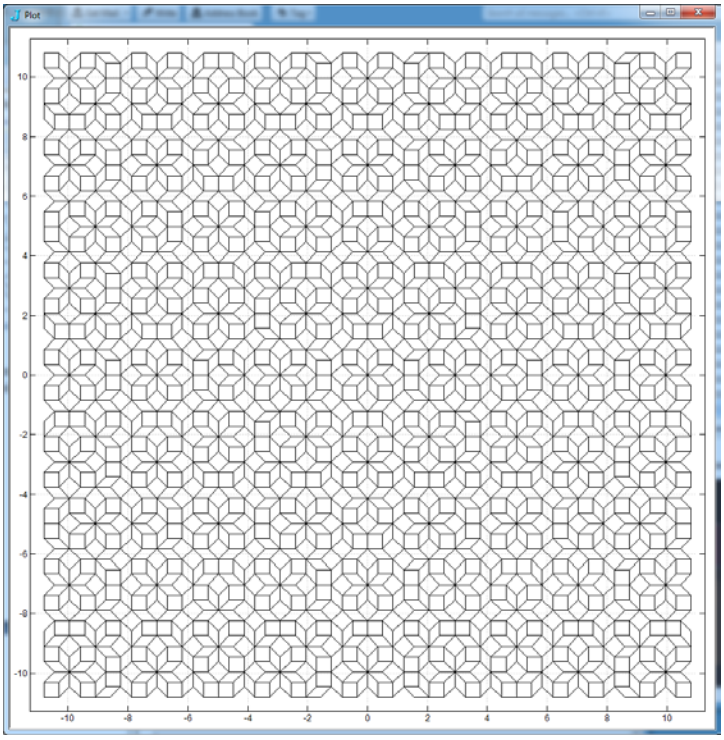


Figure 4. A Tiling via Canonical Projection from 8-Space.

References

[1] Ned W. Allis, Jeffrey P. Dumont and Clifford A. Reiter, Visualizing point sets, fractals, and quasicrystals using raster techniques, *Computers & Graphics*, 25 (2001) 519-527.

[2] Clifford A. Reiter, Atlas of Quasicrystalline Tilings, *Chaos, Solitons & Fractals*, 14 (2002) 937-963. See also: http://webbox.lafayette.edu/~reiterc/aqct/index.html.

[3] Cliff Reiter, Quasicrystals & Garage Doors via Canonical Projection Script:

http://webbox.lafayette.edu/~reiterc/j/JoJ/qcgdcp.html.

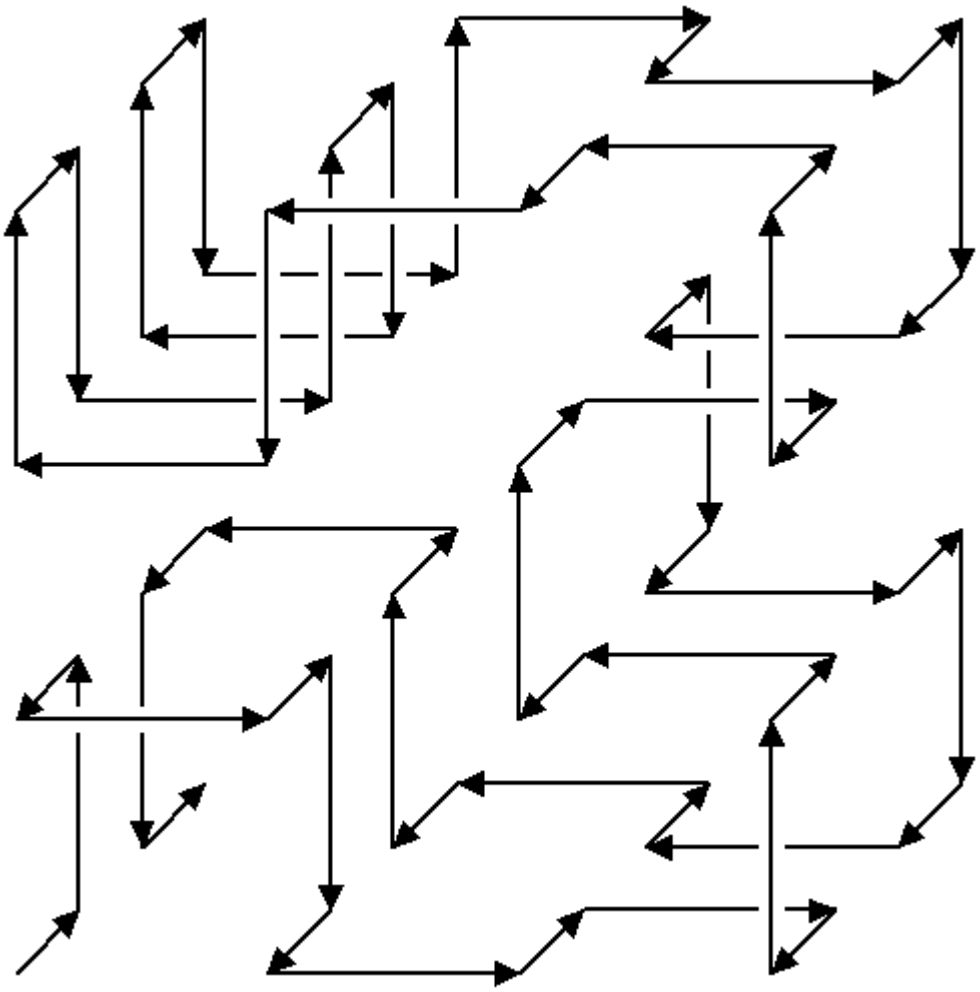
[4] M. Senechal, *Quasicrystals and Geometry*, Cambridge University Press, 1995.

Plotting depth with J

R.E. Boss

Introduction

Suppose you want a picture of a Hilbert curve like this one, a second order HC in 3D, how do you achieve that in J?



The idea being you see some depth in the figure as you walk the curve according to the arrows. It is rather obvious that a line which is behind another line is kind of interrupted by that line in the front.

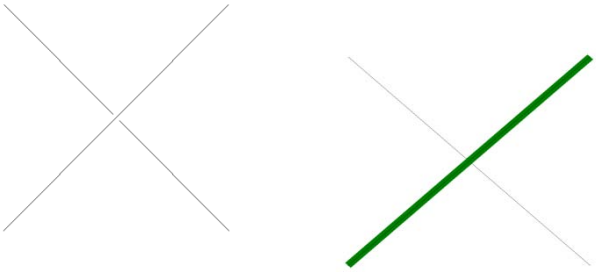
I did not find any solution in the existing features of J's plot facility, neither for depth, nor for arrows, so I designed them myself.

Line behind line

The simplest example is one line behind another, see the first figure below.

In the second picture one can see how this is achieved: by drawing a small rectangle around the line in front with a white color. In the second that color was green to make the rectangle visible.

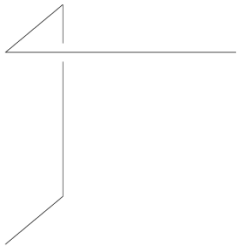
This is done by the code:



```
A=: 1 0,:0 1  NB. Let A and
B represent the lines and
pA, pB the surrounding
rectangles

B=: 0 0,: 1 1

t=: 0 1 * 0.05 % 3

'pA pB'=:(<2 3) C."_1 ,/"3(A,:B) (+,:-)"1"_1 +.*.^:_1"1  t ({."1@[ ,.
+&({:"1))"_ _1 &([:*.j./"1) -~/"_1 A,:B

    pd 'reset'

    pd 'labels 0;frame 0; tics 0; grids 0; color black'

    pd 'type line; pensize 1; color black'

    pd <"1|:A

    pd 'type poly; itemcolor white; edgecolor white'    NB.'
    itemcolor green' for the 2nd plot

    pd <"1|: pB

    pd 'type line; pensize 1; color black'

    pd <"1|:B

    pd 'show'
```

And converting this to a verb wasn't that difficult.

```
lbl=: 3 : 0         NB. lines behind lines

 NB. y is is the set of consecutive points which form a curve;
2={:$y

 Y=. y        NB.=: 1 0,1 2,2 3,_1 3,: 0

 lay=. 2-~/\Y NB. these are the separate points

 tw=. 0.1 % 3 NB. width of  'gap', i.e. rectangle

 hds=. (,:~ _1 1 |.@:(*"1) ])(,:1 _1 * ])(0-3*tw),0+1.4*tw

 pY=:. (2[\y)([:,/"3 +"1"1 2)1 0 1 #^:_1"2 +.r./"1 hds ({.@[ ,.
+&{:)"1"_ _1 &([:*.j./"1) lay

 pd 'pensize 1; type poly;itemcolor white; edgecolor white'

 pd <"(_1) 0 1 |: pY

 pd 'type line; pensize 2; color black'

 pd <"1|: Y

 NB. pd 'show'

)
```



If we choose the first 5 point of the Hilbert curve of page 1 and want get the picture on the right we do

```
pd 'reset'

pd 'tics 0;labels 0;grids 0;frame 0;boxed 0'

lbl&>"0 |. (3 4;0 1;2 3;1 2) {&.> <pNB. p contains all 64 points
of the Hilbert curve

pd 'show'
```

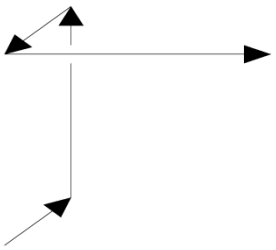
Notice that we draw the lines from the back to the front. So first the vertical one, then the diagonal ones and finally the horizontal line.

Arrows

Drawing an arrow is more or less equivalent. As can be seen in the next verb, first a standard arrowhead is constructed, this is added to each point (of the curve) and then turn in the right direction

```
arrow=: 3 : 0

 NB. y is is the set of consecutive points which form a curve;
2={:$y

 lay=: 2-~/\y       NB. these are the separate points

 ta=: 0.1 % 3       NB. size of the standard arrowhead: base is
3*ta, width is 1.4*ta

 hd=: (,:1 _1 * ])(0-3*ta),0+1.4*ta

   NB. turn head in right direction and translate

hY=. (}.y) (+"1"_1) 0,"2 +.*.^:_1"1 hd ({."1@[ ,. +&({:"1))"_ _1
&([:*.j./"1) lay

pd 'pensize 1; type poly;itemcolor black; edgecolor black'

pd <"(_1)0 1 |: hY

)
```

So we get the following figure after



```
pd 'reset'

pd 'tics 0;labels 0;grids 0;frame 0;boxed 0'

lbl&>"0|. (3 4;0 1;2 3;1 2) {&.> <p

arrow 5{.p

pd 'show'
```

Conclusion

The methods proposed here are neither sophisticated nor high browed.

They serve me in trying to express some clarity in my pictures, both 3D en 2D, of (space filling) curves.

I leave it to the reader to determine the lines from back to front. As in the HC32 on page 1, first the lines in depth 3, than those from depth 3 to depth 2, than the ones in depth 2, and so on, until the lines in depth 0, which is the front.

More examples can be seen on my wiki-pages:

http://www.jsoftware.com/jwiki/RE%20Boss/J-blog/plotdepth which is the Hilbert curve

http://www.jsoftware.com/jwiki/RE%20Boss/J-blog/Arrow

http://www.jsoftware.com/jwiki/RE%20Boss/J-blog/betaomega

PRIMER CORNER

****

****

This is a new section in *Journal of J* devoted to materials for teaching J.

In this issue we present a traduction of **J by example.**

NB. J con ejemplos-JoJ Team
*NB. Basado en el documento **J BY EXAMPLE de Oleg Kobchenko***
http://olegykj.sourceforge.net

NB. ARITMETICA

```
==============================================
   2 + 2            NB. Comentario es 'NB.'
4

   2 - 3            NB. Los números negativos usan '_'
_1

  2 * 3 + 4         NB. No precedencia, evaluación derecha a
izquierda
14
```

```
   (2 * 3) + 4      NB. Los paréntesis cambian el orden
10

   3 % 4            NB. La división se representa por '%'
0.75

  *: 4              NB. Cuadrado
16

   %: 4             NB. Raíz cuadrada
2

   % 4              NB. 1/x
0.25
```

NB. OPERACIONES CON LISTAS

```
==========================================
   2 * 1 2 3   NB.Lista de números separados por espacios
2 4 6

   1 2 3 % 2 4 6    NB.Operaciones lista a lista,igual tamaño
0.5 0.5 0.5
   #1 2 3           NB. Tamaño del vector
3

   3$1              NB. Generar secuencia de números iguales
1 1 1

   5$1 2            NB. O de una lista de números dados
1 2 1 2 1
```

NB. ELEMENTOS DE UNA LISTA

```
================================================
   {.1 2 3          NB. Primer elemento
1

   {:1 2 3          NB. Último elemento
3

   }.1 2 3          NB. Resto sin primer elemento
2 3


   }:1 2 3          NB. Resto sin último elemento
1 2
```

```
   |.1 2 3          NB. Reverso
3 2 1
```

NB. INDEXACIÓN Y CLASIFICACIÓN

```
==========================================
   1{1 2 3          NB. Indexación basada en cero
2

   1 0{1 2 3        NB. Índice también puede ser cero
2 1

   i.3              NB. Generar secuencia basada en cero
0 1 2

   2 4 6 i. 4       NB. Índice de un elemento dado
1

   /:2 1 6          NB. índices de orden
1 0 2

   /:~2 1 6         NB. Vector ordenado
1 2 6               NB. F~y <=> y F y
```

NB. AGREGACIÓN DE LISTAS

```
================================================
   1 2 3,10 20      NB. Union de vectores
1 2 3 10 20

   1 + 2 + 3        NB. sum of elements
6

   +/1 2 3 NB. inserta '+' entre elementos
6

   +/\1 2 3         NB. Sumas sucesivas de elementos
1 3 6

   1,(1+2),(1+2+3)  NB. Igual que esto
1 3 6

   2+/\1 2 3 4 5    NB. Sumas sucesivas de “pares”
3 5 7 9


   _2+/\1 2 3 4 5   NB. non-intersecting pairs
3 7 5
```

```
   (<1 2),3 4 6;7 6 NB. < encajado, ; es encajado y unión
+---+-----+---+
|1 2|3 4 6|7 6|
+---+-----+---+

   >{. 3 4 6;7 6    NB. > desencajado
3 4 6
```

NB. COMBINACIÓN DE FUNCIONES

```
==========================================
   (+ *:) 4          NB. hook (F G) y <=> y F (G y)
20 NB. a + a^2

   (%: , *:) 4       NB. fork (F G H) y <=> (F y) G (H y)
2 16 NB. [sqrt(a), a^2]

   *:@(+/) 2 3       NB. composición (F o G) y <=> F G y
25 NB. (a + b)^2

   2 +&*: 3          NB. x F & G y <=> (G x) F (G y)
13                   NB. a^2 + b^2

   2 (+&*: + 2: * *) 3 NB. (a + b)^2 = a^2 + b^2 + 2ab
25                   NB. 0: 1: 2: ... son funciones Constantes

   3 +&.*: 4         NB. F&.G y <=> (G^:_1) F G y
5                    NB. sqrt(a^2 + b^2)
```

NB. FUNCIONES DEFINIDAS POR EL USUARIO Y ARGUMENTOS

```
============================
m1=: -               NB. Declaración tácita ambivalente

m2=: 3 : '-y.'       NB. Declaración monádica explicita

m3=: 4 : 'x.-y.'     NB. Diádica explcita

   (m1 , m2 , 0&m3) 4    NB. Uso monádico, 0& es unión
_4 _4 _4

   3 (m1 , (+ m2) , m3) 4 NB.Uso diádico, hook para
dyadization
_1 _1 _1


   (m1 , m3) / 3 4  NB. Distribución de argumentos:
dyadization
_1 _1
```

```
   3 (m1 , m4) @ , 4          NB. Recopilar argumentos:
monadization
_3 _4 _3 _4
```

NB. EXPONENCIAL Y LOGARITMO

```
==========================================
   1x1 2x1 1x2 NB. e, 2e, e al cuadrado
2.71828 5.43656 7.38906

   ^2 NB. exponent, e^2
7.38906

   2^16 NB. exponent base 2, 2^16
65536

   ^. 1x2 NB. logaritmo, ln e^2
2

   2^.65536 NB. logaritmo base 2, log2 65536
16
```

NB. TRIGONOMETRIA

```
====================================================
   1p1 2p1 1p2 NB. pi, 2 pi, pi al cuadrado
3.14159 6.28319 9.8696

   load'trig' NB. Carga libreria trigonometrica
   cos 1p1 NB. cosine of pi
_1

   (*:cos 1p1) + *:sin 1p1 NB. Teorema de la trigonometria
1

   (cos +&*: sin) 1 2p1 1p2 NB. Igual usando fork and &
1 1 1
```

NB. MATRICES

```
 ==========================================================
   1 2 3 */ 1 2 3 NB. Producto exterior
1 2 3             NB. Igual que */~ 1 2 3
2 4 6
3 6 9

   =/~i.3     NB. Matriz identidad, also =@i. (auto
              clasificación)
```

```
1 0 0          NB. F~y <=> y F y
0 1 0
0 0 1

   ]M=. i.2 3 NB. Genera matriz
0 1 2
3 4 5

   2 2 $ 0 1 1 1 NB.transforma un vector dado a matriz
0 1
1 1


NB. TRANSFORMACIONES ESTRUCTURALES
==========================================
   ,N=: i.2 2 3                NB. “enmarañar”: lista de
átomos
0 1 2 3 4 5 6 7 8 9 10 11

   ,"2 N                       NB. enmarañar cada submatriz
0 1 2 3 4 5
6 7 8 9 10 11

   (]; |:; |.;|."1;1&|.) M=. 3 3$'ABC123!@#'
                               NB. Matriz de caracteres
+---+---+---+---+---+          NB. ] devuelve argumento
|ABC|A1!|!@#|CBA|123|          NB. |: traspuesta
|123|B2@|123|321|!@#|          NB. |. reverses outer list
|!@#|C3#|ABC|#@!|ABC|          NB. |."1 reverses inner list
+---+---+---+---+---+          NB. 1|. rotates outer list

   ;:^:_1 </.M                 NB. oblique: secondary
diagonals
A B1 C2! 3@ #                  NB. same as (</.~&, +"0/~@i.@#)
M
                               NB. ;:^:_1 is inverse of boxing
                           tokens

   i.@# } M                    NB. diagonal principal
A2#

NB. SELECCIÓN
=====================================================
   1{1{1{N                     NB. Selección repetitiva de
elementos de Lista
10
```

```
   1{^:3 N                    NB. applicar selección 3 veces
10

   (<1 1 1){N                 NB. scatter select
10
   1 1 1 ({~ <)~ N            NB. using unboxed list
10
```

NB. FACTORIAL Y BINOMIAL

```
=========================================
   ! 1+i.5                    NB. factorial
1 2 6 24 120

   */\ 1+i.5                  NB. Productos sucesivos
1 2 6 24 120

   !/~ i.5                    NB. Coeficientes binomiales
1 1 1 1 1
0 1 2 3 4
0 0 1 3 6
0 0 0 1 4
0 0 0 0 1

   +/@(! |.)\i. 15            NB. fibonacci: suma de la
segunda diagonal de la matriz binomial
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610
```

NB. PRODUCTO ESCALAR

```
=====================================================
   1 2 3(+/ . *)1 2 3         NB. Producto escalar
14

M=: 2 2$0 1 1 1               NB. signación
dot=: +/ . *                  NB. Dar un nombre a una función

   dot~ M                     NB. Matriz cuadrada
1 1
1 2

   dot^:(15)~ M               NB. Matriz a la potencia de 15,
610 987
987 1597

   {:@{."2 dot^:(<15)~ M      NB. F^:n se aplica F n veces
acumulativamente
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610
```

```
NB. ALEATORIEDAD YPROBABILIDAD
======================================
   ]A=: 5 ?@$ 0                NB. 5 números de 0..1
0.57708 0.542732 0.488337 0.26004 0.0101683
   ]B=: 10 ?@$ 2               NB. Lanzamiento de moneda
1 1 1 0 0 1 1 0 1 0

   ]C=: 3 ? 3                  NB. deal 3 out of 3 cards in
certain order
1 2 0

   (<./ , >./) A               NB. min and max over the list
0.0101683 0.57708

   B i. 0                  NB. first zero
3

   (+/ % #) C-:"1 (?~"0) 10000#3 NB. method monte carlo
0.1637                         NB. -: is list equality, F"n is
rank modifier

   %!3                         NB. exact probability of 3
cards in given order
0.166667                       NB. unique elements
================================================
   ]D=.~. S=. 'mississippi' NB. distinct (nub)
misp

   ]K=. D i. S                 NB. key (index)
0 1 2 2 1 2 2 1 3 3 1

   K </. S                 NB. group by key
+-+----+----+--+
|m|iiii|ssss|pp|
+-+----+----+--+

   K #/. S                 NB. frequencies
1 4 4 2

   ]I=. ~: S               NB. sieve of nub
1 1 1 0 0 0 0 0 1 0 0    NB. where D is in S

   I # S                   NB. filter by sieve to get D
misp
```

```
   +/"1 =S                    NB. = is classify, bool matrix of S
vs D
1 4 4 2                       NB. where items of D are in S
```

POUTPOURRI

Levenshtein distance, string metrics, J and bioinformatics. String distance and patters from strings…to numbers!

matxinlekim at hotmail dot com

In Vol.0, No.1 of *JoJ* the paper *Correlation in simbolic sequences* shows how use J language in bioinformatics. Several metrics are used between sequences. The most basical metric is the *Hamming distance* between two strings of equal length. This distance is the number of positions at which the corresponding symbols are different.

```
seq1=.'ASDERFTESXDR'

seq2=.'ASFRRFTTXXDR'

   +/seq1~:seq2  NB.Hamming

4
```

Another metric in information theory and computer science, is the *Levenshtein distance.* This is a string metric for measuring the amount of difference between two sequences. The term *edit distance* is often used to refer specifically to Levenshtein distance. In JWiki is a traslation of the wikipedia code:

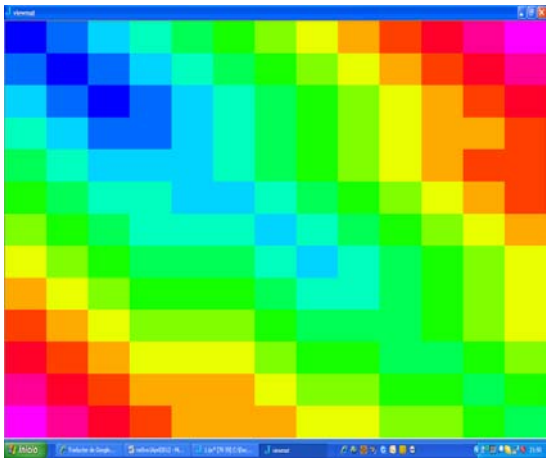
```
LevenshteinMatrix=: 4 : 0
 d=. (1+(#x),#y)$0
 for_i. i.1+#x do. d=. i (<i,0)}d
end.  NB. deletion
 for_j. i.1+#y do. d=. j (<0,j)}d
end.  NB. insertion
 for_j. i.#y do.
  for_i. i.#x do.
   if. (i{x) = j{y do.
    d=. d (<1+i,j)}~ (<i,j){d
   else.
    m=.      1 + (<i,1+j){d
NB. deletion
    m=. m <. 1 + (<(1+i),j){d
NB. insertion
    m=. m <. 1 + (<i,j){d
NB. substitution
    d=. m (<1+i,j)}d
 end. end. end.
```

In our example (seq1 and seq2) the result is:

```
   seq1 LevenshteinMatrix seq2

0  1  2 3 4 5 6 7 8 9 10 11 12
 1  0  1 2 3 4 5 6 7 8  9 10 11
 2  1  0 1 2 3 4 5 6 7  8  9 10
 3  2  1 1 2 3 4 5 6 7  8  8  9
 4  3  2 2 2 3 4 5 6 7  8  9  9
 5  4  3 3 2 2 3 4 5 6  7  8  9
 6  5  4 3 3 3 2 3 4 5  6  7  8
 7  6  5 4 4 4 3 2 3 4  5  6  7
 8  7  6 5 5 5 4 3 3 4  5  6  7
 9  8  7 6 6 6 5 4 4 4  5  6  7
```

```
10  9  8 7 7 7 6 5 5 4  4  5  6
11 10  9 8 8 8 7 6 6 5  5  4  5
12 11 10 9 8 8 8 7 7 6  6  5  4
```

With `viewmat` the tool result is



For calculating distance we can use the next code (HenryRich in JWiki)

```
levdist=: 4 : 0 " 1
 'a b'=. (/: #&>)x;y
 z=. >: iz =. i.#b
 for_j. a do.
  z=. <./\&.(-&iz) (>: <. (j ~:
b) + |.!.j_index) z
 end.
 {:z
)
```

Now, an interesting experiment. First, we use the verb `<.@o.10x^n` to obtain the first n decimal numbers of pi.

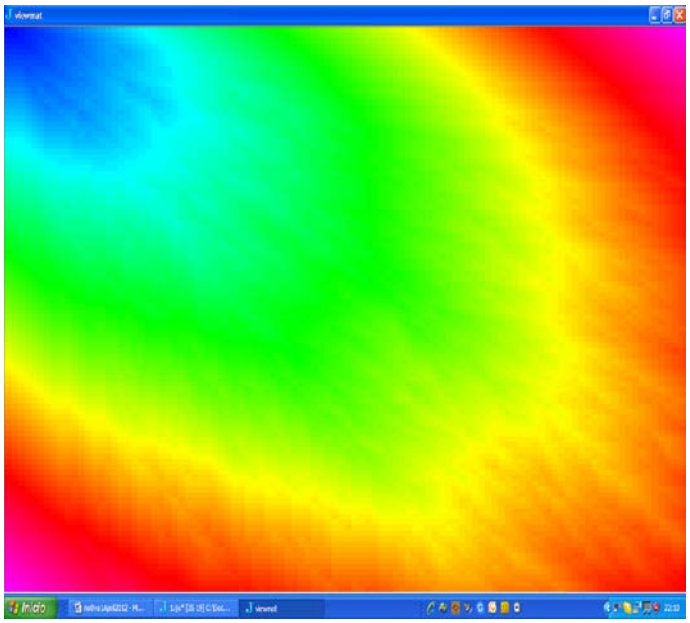
```
]a=.<.@o.10x^20

31415926535897932384
```

With this verb we can create large sequences of digital numbers (number in the decimal representation of pi) Now we can compare the first n/2 numbers with the rest in the digital expansion.

For example (n=200)

The next code (thanks to J forum) extract digits and assign to variables e and f

```
'e f'=. |:_2[\10#.^:_1 a

load 'viewmat

viewmat e LevenshteinMatrix f
```



A different view of Pi !!

Collaborators in this issue:

Cliff Reiter	Creating Quasicrystals & Garage Doors via Canonical
R.E. Boss	Plotting depth with J
J2-Team	J con ejemplos, edition, etc.
Matxin Lekim	Levenshtein distance in Poutpourri section

****

****

****

****

****

****

*

MPM Press **AN OPEN JOURNAL**

ISSN: 2174-9280

Vol. 1, No.2, April 2012

http://sites.google.com/site/jforscience/ **journalofj@hotmail.com**