Curvature of Constant plaquettes on a loop

July 8, 2017

0.1 Table of contents

1 Mathematical Background

We investigate constant plaquettes of size d, with one plaquette per site. In order terms, we consider the map

$$\begin{array}{ccc} \mathbb{C}^{2^d} & \mapsto & (\mathbb{C}^2)^{\otimes N} \\ (C^{\delta_1 \dots \delta_d}) & \mapsto & \sum_\epsilon \prod_{i=1}^N C^{\epsilon_i \cdots \epsilon_{i+d-1}} |\epsilon\rangle, \end{array}$$

which to C associates $|\phi(C)>$. This map is holomorphic.

We focus on the local geometry of this map. As we will see, it is never an immersion. The null tangential vectors are explicit, and we will also give a basis of their orthogonal space.

We also give a numerical scheme to compute, up to machine precision, the curvature, which is a symmetric 2-form on the holomorphic tangent space:

$$G(\eta, \xi) = \overline{\partial}_\eta \partial_{\bar{\xi}} \log(\langle \phi(C) | \phi(C) \rangle).$$

This form is positive, as the Levi form of a plurisubharmonic function, but it is definite positive only when restricted to the orthogonal of the null space above.

1.1 The Null space

In order to understand completely the geometry of the image set above, we will make two simplifying assumptions: * N is a multiple of d(d-1); * All coefficients $C^{\delta_1, \dots, \delta_d}$ are non-zero.

The first hypothesis might seem strange, since we are mainly interested in the large N case. In fact, when N is a multiple of $d(d-1)$ the proofs are much simpler, and when it is not the case (or when we consider plaquettes on a chain, without periodicity), one must add terms to G, of order N^{-1}.

The second hypothesis is of course true in a generic configuration; it is also true along a (real) one-dimensional family of configurations. As we are interested in time evolution, one can argue that this hypothesis will remain valid along trajectories.

As all coefficients are non-zero, we make use of the projective properties of the map ϕ to consider a basis for variations of parameters which is more convenient than the natural one. If $\alpha \in (\mathbb{C}^2)^{\otimes d}$, we let

$$\alpha \cdot |\phi(C)\rangle = \sum_\delta \alpha_\delta C^\delta \partial_{C^\delta} |\phi(C)\rangle.$$

This is motivated by the fact that

$$C^\delta \partial_{C^\delta} |\phi(C)\rangle = \sum_{i=1}^N \sum_{\epsilon \in A_i^\delta} \prod_{j=1}^N C^{\epsilon_j \dots \epsilon_{j+d-1}} |\epsilon\rangle.$$

Here,

$$A_i^\delta = \{\epsilon \,|\, \epsilon_i = \delta_1, \dots, \epsilon_{i+d-1} = \delta_d\}.$$

The main result is

Theorem 1 $\alpha \cdot |\phi(C)\rangle = 0$ if and only if $\alpha = (|0\rangle + |1\rangle) \otimes v - v \otimes (|0\rangle + |1\rangle)$ for some $v \in (\mathbb{C}^2)^{\otimes d-1}$.

Proof of Theorem 1 One inclusion proceeds by direct computation. Indeed, if $v = \sum v_\delta |\delta\rangle$, then

$$(|0\rangle + |1\rangle) \otimes v \cdot |\phi(C)\rangle = \sum_\delta \sum_{i=1}^N v_\delta \sum_{\epsilon \in A_i^{0\delta} \cup A_i^{1\delta}} \prod_{j=1}^N C^{\epsilon_j \cdots \epsilon_{j+d-1}} |\epsilon\rangle,$$

and

$$A_i^{0\delta} \cup A_i^{1\delta} = A_{i+1}^{\delta 0} \cup A_{i+1}^{\delta_1},$$

so that

$$(|0\rangle + |1\rangle) \otimes v - v \otimes (|0\rangle + |1\rangle) \cdot |\phi(C)\rangle = 0.$$

The other inclusion is less direct and makes full uses of the hypotheses above.

Since all coefficients are non-zero, then $\alpha \cdot |\phi(C)\rangle = 0$ if and only if, for every ϵ, one has $\sum_{i=1}^N \alpha_{\epsilon_i \cdots \epsilon_{i+d-1}} = 0$.

In particular, let $\delta \in \{0,1\}^d$ arbitrary, and let $\epsilon = \delta\delta\delta...$ a concatenation of size N. (We assumed that N is a multiple of d). Then the equation above yields $\sum_{k=0}^{d-1} \alpha_{ls(d)} = 0$, with ls the operator of left shift.

In particular,

$$\sum_{k=0}^{d-1} ls^k(\alpha) = 0.$$

Using Lemma 1 in the Appendix, it follows that $\alpha = v - ls(v)$ for some v.

We will now prove that, for every word $w \in \{a,b\}^{d-2}$, one has

$$\alpha_{0w0} + \alpha_{1w1} - \alpha_{0w1} - \alpha 1w0 = 0.$$

Using Lemma 3, with $a = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$ and $b = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$, this will conclude the proof.

Recall that each $\epsilon \in \{0,1\}^N$ yields an equation on α. Substracting the equation given by $\epsilon = ...0w0w1w0w0...$ (recall N is a multiple of $d-1$) from the one given by $\epsilon = ...0w0w0...$ yields

$$\alpha_{0w1} + \ldots + \alpha_{1w0} = \alpha_{0w0}.$$

Now, substracting the equation given by $\epsilon = ...0w0w1w1w0w0...$ from $\epsilon = ...0w0w0...$ yields

$$\alpha_{0w1} + \ldots + \alpha_{1w0} - \alpha_{1w0} + \alpha 1w1 - \alpha 0w1 + \alpha 0w1 + \ldots + \alpha 1w0 = \alpha 0w0.$$

Hence,

$$\alpha_{1w1} + \alpha_{0w0} = \alpha_{0w1} - \alpha_{1w0}.$$

This concludes the proof.

1.2 Probabilistic interpretation of the curvature

The expression of the derivative of $|\phi(C)\rangle$ along a base vector was formulated as a sum with the same terms as the initial sum, but restricted on some configuration set A_i^δ. These sums are ubiquitous in the computation of the curvature, which will allow a probabilistic reformulation. Since we are dealing with one-dimensional loops, the probabilistic model will be a Markov chain (restricted to identical final and initial states). If all coefficients are non-zero, this Markov chain is mixing, and this will allow a precise numerical treatment.

We want to compute

$$G(\alpha, \beta) = \alpha \bar{\cdot} \beta \cdot \log(\langle \phi(C) | \phi(C) \rangle).$$

Here, $\bar{\cdot}$ stands for anti-holomorphic derivation. This is exactly

$$G(\alpha, \beta) = \frac{\langle \alpha \cdot \phi(C) | \beta \cdot \phi(C) \rangle}{\langle \phi(C) | \phi(C) \rangle} - \frac{\langle \alpha \cdot \phi(C) | \phi(C) \rangle \langle \phi(C) | \beta \cdot \phi(C) \rangle}{(\langle \phi(C) | \phi(C) \rangle)^2}.$$

Now

$$\langle \phi(C) | \phi(C) \rangle = \sum_\epsilon \prod_{j=1}^N |C^{\epsilon_j \ldots \epsilon_{j+d-1}}|^2.$$

Moreover, if $\alpha = C^\delta \partial_{C^\delta}$ and $\beta = C^{\delta'} \partial_{C^{\delta'}}$, then

$$\langle \alpha \cdot \phi(C) | \phi(C) \rangle = \sum_{i=1}^N \sum_{\epsilon \in A_i^\delta} \prod_j |C^{\epsilon_j \ldots \epsilon_{j+d-1}}|^2.$$

Finally,

$$\langle \alpha \cdot \phi(C) | \beta \cdot \phi(C) \rangle = \sum_{i_1=1}^N \sum_{i_2=1}^N \sum_{\epsilon \in A_{i_1}^\delta \cap A_{i_2}^{\delta'}} \prod_j |C^{\epsilon_j \ldots \epsilon_{j+d-1}}|^2.$$

Those expressions are quite clumsy. Let $(\{0,1\}^N, \mathcal{P}(\{0,1\}^N), \mathbb{P}_C)$ be a probabilized space whose elementary events are words of length N on $\{0,1\}$, with

$$\mathbb{P}_C(\epsilon) = (\prod_{j=1}^N |C^{\epsilon_j \cdots \epsilon_{j+d-1}}|^2) \langle \phi(C) | \phi(C) \rangle^{-1}.$$

Then the matrix elements of G are simply

$$G_{\delta,\delta'} = \sum_i \sum_j \mathbb{P}_C(A_i^\delta \cap A_j^{\delta'}) - \mathbb{P}_C(A_i^\delta)\mathbb{P}_C(A_j^\delta).$$

To evaluate those probabilities, we will reformulate this probabilized space with a Markov chain.

Consider the Markov chain, with $\{0,1\}^d$ as set of states, with transition matrix:

$$T(\delta_1 \dots \delta_d, \delta_2 \dots \delta_{d+1}) = \frac{|C^{\delta_2 \dots \delta_{d+1}}|^2}{|C^{\delta_2 \dots \delta_d 0}|^2 + |C^{\delta_2 \dots \delta_d 1}|^2}.$$

Consider realisations of this Markov chain, of length $N+1$, conditionned to having the same final and initial state. Since the $d-1$ last digits at one time coincide with the $d-1$ first digits at the following time, this realisation is a sequence of 0 and 1, of length $N+d$ (where the state, at step j, consists of the digits j to $j+d-1$.). Moreover, the $N+1$ digits corresponds to the first one, and so on. Then the probability of this sequence ϵ is exactly $\mathbb{P}_C(\epsilon)$, as long as the initial state is chosen according to the law of A_1.

The matrix T verifies the hypotheses of the Perron-Frobenius theorem: since all coefficients are non-zero, the matrix T^d has only positive entries. In particular T has 1 as only eigenvalue of modulus one (all other eigenvalues have strictly smaller modulus), and this eigenvalue is simple. From there we are able to identify the law of A_i. Indeed, on one hand the original probability space is rotation invariant so that A_i and A_j have the same law for any i and j; on the other hand the law of A_{i+1} (which is a 2^d-dimensional vector) is T times the law of A_i, plus an error of size e^{-cN}. Hence, if μ_C is the equilibrium measure for T, then

$$\mathbb{P}_C(A_i^\delta) = \mu_C(\delta) + O(e^{-cN}).$$

Along the same lines, if j is closer from i to the left than to the right (meaning j is between i and $i + N/2$ on $\mathbb{Z}/N\mathbb{Z}$, then

$$\mathbb{P}_C(A_j^{\delta'}|A_i^\delta) = T^{j-i}[\delta', \delta] + O(e^{-cN}).$$

In particular,

$$\mathbb{P}_C(A_j^{\delta'}|A_i^\delta) = \mathbb{P}_C(A_j^{\delta'} + O(e^{-c|i-j|}).$$

The intuition of this estimate is that, if i and j are far apart (more than the reduced spectral radius of T allows for), then A_i and A_j are independent up to an exponentially small error.

This offers a numerical scheme to compute G up to machine precision. First, we compute and store the sequence of positive powers of T, until they converge to a matrix of rank one (at step k_M). Each line of this matrix is the invariant measure μ_C. Then

$$N^{-1}G_{\delta,\delta'} = \sum_{j=0}^{k_M} \mu_C(\delta)(T^j_{\delta',\delta} - \mu_C(\delta')) + \sum_{j=1}^{k_M} \mu_C(\delta')(T^j_{\delta,\delta'} - \mu_C(\delta')) + O(\text{machine precision}) + O(Ne^{-N}).$$

2 Code documentation

First and foremost, we need to import standard numerical libraries, as well as scipy.linalg to show the spectrum of the curvature form

```
In [2]: import numpy as np, random, scipy.linalg
```

2.1 A convenient basis

The file curv_basis.py contains the decomposition along the null space and a complementary subspace, according to the discussion above.

```
In [3]: # %load ../curv_basis.py
        # a tentative basis

        def lshift(c,d):
            return (2*c)%(2**d)+c/(2**(d-1))

        def rtldigits(c,d):
            """Returns the reversed digits of c in base 2, of length d"""
            ccopy=c
            dig=[]
            for j in range(d):
                dig.append(ccopy%2)
                ccopy=ccopy/2
            return np.array(dig)

        def rotation(d=3):
            """Rotates from the base along z, to the base along x."""
            U=np.zeros((2**d,2**d))
            for c in range(2**d):
                for k in range(2**d):
                    U[c,k]=(-1)**(sum((1-rtldigits(c,d))*(1-rtldigits(k,d))))/2**(0.5*d)
            return U

        def fullbasis(d=3):
            U=np.zeros((2**d,2**d))
            line=0
            #First we look at the zero vectors
            #they look like 1c-c1 for some c
            #but we have to give an orthogonal basis !
            for j in range(d-2):
                for c in range(2**j):
                    #we consider the vectors from 1...10c0 to 0c01.....1 of size d-1
                    #here they form a Toeplitz matrix of rank d-2-j,
                    #which can easily be diagonalized
                    for k in range(d-2-j):
                        for l in range(d-2-j):
                            vect=2**(d-1)-2**(l+j+2)+c*2**(l+1)+2**l-1
                            val=np.sin(np.pi*(k+1)*(l+1)/(d-1-j))
                            U[line+k,2**(d-1)+vect]+=val
                            U[line+k,2*vect+1]-=val
                        U[line+k]/=np.sqrt(np.dot(U[line+k],U[line+k]))
                    line+=d-2-j
            #then we only forgot the vectors 1.....10 to 01......1
            for k in range(d-1):
```

```
            for l in range(d-1):
                vect=2**(d-1)-1-2**l
                val=np.sin(np.pi*(k+1)*(l+1)/d)
                U[line+k,2**(d-1)+vect]+=val
                U[line+k,2*vect+1]-=val
            U[line+k]/=np.sqrt(np.dot(U[line+k],U[line+k]))
        #The vector 1...1 is itself orthogonal
        line+=d-1
        U[line,2**d-1]=1.
        line+=1
        #There are several types of orthogonal (nonzero) vectors:
        #All 0c0 vectors
        for c in range(2**(d-2)):
            U[line+c,2*c]=1
        line+=2**(d-2)
        #All 10c0+0c01 vectors
        for c in range(2**(d-3)):
            U[line+c,2*c+2**(d-1)]=1./np.sqrt(2)
            U[line+c,4*c+1]=1./np.sqrt(2)
        line += 2**(d-3)
        #Some weird vectors
        for c in range(2**(d-3)):
            current=2*c+2**(d-1)+2**(d-2)
            shiftlist=[current]
            while current>=2**(d-1):
                current=lshift(current,d)
                shiftlist.append(current)
            for shifted in shiftlist:
                U[line+c,shifted]=1./np.sqrt(len(shiftlist))
        return U
```

The first step is the function rotation, which returns the matrix which exchanges the basis given by $(|0\rangle, |1\rangle)$ with the basis given by $(2^{-\frac{1}{2}}(|0\rangle + |1\rangle), 2^{-\frac{1}{2}}(|0\rangle + |1\rangle))$, on $(\mathbb{C}^2)^{\otimes N}$.

The second step is the function fullbasis, which computes the decomposition of the space given by Lemma 2 (with $a = |0\rangle$ and $b = |1\rangle$. Here, the non-trivial part is to give an orthonormal basis of the space E.

Indeed, if v is a base vector, $\langle a \otimes v - v \otimes a | a \otimes v' - v' \otimes a \rangle$ is equal to 2 if $v = v'$ (unless $v = aa.....a$), and otherwise is equal to -1 if $v \otimes a = a \otimes v'$ (which happens only if v begins by an a and $v' = ls(v)$) or if $v' \otimes a = a \otimes v$ (which happens only if v ends by an a and $v = ls(v')$). The exceptional case when both $v \otimes a = a \otimes v'$ and $v' \otimes a = a \otimes v$ means that $v = awa$ and $v' = aw'a$ with $v' = ls(v)$ so that $v = a...a$, a case which was excluded a priori.

Hence, the structure of the matrix given by the previous scalar product is formed of tridiagonal Toeplitz matrices, with 2 on the diagonal and -1 on the over- and underdiagonal; these matrices can be explicitly diagonalized.

The natural basis of F and G given in Lemma 2 is orthogonal (as every word appears only once, either in F, or in G), and requires only a normalization.

```
In [4]: #We check that the change of basis U is indeed orthogonal
```

```
U=np.dot(fullbasis(4),rotation(4))
np.sum(np.abs(np.dot(U,U.T)-np.eye(2**4)))
```

```
Out[4]: 3.6332767765565026e-15
```

2.2 Computation of the curvature

The file curvature.py encodes the Markov chain to compute the curvature.

```
In [5]: # %load ../curvature.py
        import numpy as np

        def generate_C(d,m=1,M=2):
            """
            Generates a bunch of randomly chosen coefficients, far from zero
            """
            C=[]
            for k in range(2**d):
                C.append(random.random()*(M-m)+m)
            return C

        def sparse_mult(mat,C):
            """returns an efficient multiplication of mat by the transfer matrix.
            Runs in quadratic time.
            """
            prod=np.zeros((len(C),len(C)),dtype=float)
            for i in range(len(C)):
                for j in range(len(C)):
                    prod[i,j]=mat[i,(2*j)%len(C)]*C[(2*j)%len(C)]
                    prod[i,j]+=mat[i,(2*j+1)%len(C)]*C[(2*j+1)%len(C)]
                    prod[i,j]/=C[(2*j)%len(C)]+C[(2*j+1)%len(C)]
            return prod

        def Markov_powers(C,ERROR=1.0e-14):
            """Computes the powers of the Markov chain
            for constant plaquettes"""
            powerseq=[np.eye(len(C))]
            converged=False
            while not converged:
                current=powerseq[-1]
                new=sparse_mult(current,C)
                powerseq.append(new)
                dist=0.
                for k in range(len(C)/2):
                    dist+=abs(new[k,0]-new[k,1])
                if dist<ERROR:
                    converged=True
                #print dist
```

```
        print "Markov chain convergence after",len(powerseq),"steps"
        return powerseq

    def curvature(C,ERROR=1.0e-14):
        """Computes, via a Markov chain, the curvature tensor
        for constant plaquettes up to some prescribed error."""
        #compute the sequence (M**i) until convergence
        powerseq=Markov_powers(C,ERROR)
        #the lines of the last matrix are the equilibium measure
        eq_m=powerseq[-1].T[1]
        print "Equilibrium measure: ", eq_m
        G=np.zeros((len(C),len(C)),dtype=float)
        M=powerseq[1]
        for i in range(len(C)):
            for j in range(len(C)):
                for k in range(len(powerseq)):
                    current=powerseq[k]
                    G[i,j]+=eq_m[j]*(current[i,j]-eq_m[i])
                    if k !=0:
                        G[i,j]+=eq_m[i]*(current[j,i]-eq_m[j])
        return G
```

We first have to generate the square norm of the coefficients C; here they are chosen as uniform independent random variables, shifted away from zero (as we want to ensure that all coefficients are non-zero)

```
In [6]: C=generate_C(4)
```

Then we can compute the curvature (which, for the moment, is expressed in the canonical basis).

```
In [7]: G=curvature(C)

Markov chain convergence after 35 steps
Equilibrium measure:  [ 0.09658027  0.06349455  0.05311281  0.06415667  0.05001956  0.05467306
  0.08056526  0.05793416  0.06349455  0.05377493  0.05157982  0.07434275
  0.06724992  0.07124951  0.05793416  0.03983802]
```

As a control tool, we printed the step at which the Markov chain converges, and the equilibrium measure.

The matrix G is indeed quite complicated at this step.

```
In [8]: G

Out[8]: array([[ 0.27130832,  0.03375089,  0.00176749, -0.00813471,  0.00276114,
                -0.02990021, -0.03039709, -0.03130129,  0.03375089, -0.04011811,
                -0.02890656, -0.05356367, -0.00912836, -0.05257002, -0.03130129,
                -0.02801743],
```

```
[ 0.03375089,  0.03235265,  0.00966402,  0.00492188,  0.00982212,
 -0.01090545, -0.00708742, -0.01130454,  0.03235265, -0.01776675,
 -0.01074734, -0.02331384,  0.00476378, -0.02315574, -0.01130454,
 -0.01204237],
[ 0.00176749,  0.00966402,  0.04296037, -0.01791113,  0.02767104,
  0.01214251, -0.01384337, -0.01514766,  0.00966402,  0.01538522,
 -0.00314682, -0.0110799 , -0.0026218 , -0.02636923, -0.01514766,
 -0.01398708],
[-0.00813471,  0.00492188, -0.01791113,  0.03569924, -0.00274915,
 -0.02487194,  0.01167575,  0.00212325,  0.00492188,  0.01286623,
 -0.00970996, -0.02190024,  0.02053726, -0.00673826,  0.00212325,
 -0.00285337],
[ 0.00276114,  0.00982212,  0.02767104, -0.00274915,  0.04215109,
 -0.00346755, -0.01390459, -0.01488926,  0.00982212,  0.01509977,
  0.0110125 , -0.02604469, -0.0172292 , -0.01156465, -0.01488926,
 -0.01360142],
[-0.02990021, -0.01090545,  0.01214251, -0.02487194, -0.00346755,
  0.05649341, -0.00373309, -0.00802996, -0.01090545, -0.00182399,
  0.04088335,  0.0131089 , -0.00926188, -0.00250116, -0.00802996,
 -0.00919754],
[-0.03039709, -0.00708742, -0.01384337,  0.01167575, -0.01390459,
 -0.00373309,  0.06103464, -0.02192143, -0.00708742,  0.00491981,
 -0.00379431,  0.02743745,  0.01173698,  0.02737623, -0.02192143,
 -0.02049071],
[-0.03130129, -0.01130454, -0.01514766,  0.00212325, -0.01488926,
 -0.00802996, -0.02192143,  0.0365062 , -0.01130454, -0.00171988,
 -0.00777155,  0.01246151,  0.00186484,  0.01271992,  0.0365062 ,
  0.02120818],
[ 0.03375089,  0.03235265,  0.00966402,  0.00492188,  0.00982212,
 -0.01090545, -0.00708742, -0.01130454,  0.03235265, -0.01776675,
 -0.01074734, -0.02331384,  0.00476378, -0.02315574, -0.01130454,
 -0.01204237],
[-0.04011811, -0.01776675,  0.01538522,  0.01286623,  0.01509977,
 -0.00182399,  0.00491981, -0.00171988, -0.01776675,  0.0460182 ,
 -0.00210944, -0.00966629,  0.01315168, -0.00995175, -0.00171988,
 -0.00479808],
[-0.02890656, -0.01074734, -0.00314682, -0.00970996,  0.0110125 ,
  0.04088335, -0.00379431, -0.00777155, -0.01074734, -0.00210944,
  0.05504267, -0.0018559 , -0.02386928,  0.01230342, -0.00777155,
 -0.00881188],
[-0.05356367, -0.02331384, -0.0110799 , -0.02190024, -0.02604469,
  0.0131089 ,  0.02743745,  0.01246151, -0.02331384, -0.00966629,
 -0.0018559 ,  0.0617992 , -0.00693544,  0.04683441,  0.01246151,
  0.00357083],
[-0.00912836,  0.00476378, -0.0026218 ,  0.02053726, -0.0172292 ,
 -0.00926188,  0.01173698,  0.00186484,  0.00476378,  0.01315168,
 -0.02386928, -0.00693544,  0.03514466, -0.02154284,  0.00186484,
 -0.00323902],
```

```
       [-0.05257002, -0.02315574, -0.02636923, -0.00673826, -0.01156465,
        -0.00250116,  0.02737623,  0.01271992, -0.02315574, -0.00995175,
         0.01230342,  0.04683441, -0.02154284,  0.061639  ,  0.01271992,
         0.00395649],
       [-0.03130129, -0.01130454, -0.01514766,  0.00212325, -0.01488926,
        -0.00802996, -0.02192143,  0.0365062 , -0.01130454, -0.00171988,
        -0.00777155,  0.01246151,  0.00186484,  0.01271992,  0.0365062 ,
         0.02120818],
       [-0.02801743, -0.01204237, -0.01398708, -0.00285337, -0.01360142,
        -0.00919754, -0.02049071,  0.02120818, -0.01204237, -0.00479808,
        -0.00881188,  0.00357083, -0.00323902,  0.00395649,  0.02120818,
         0.0791376 ]])
```

However, in the base given by the matrix U, the quadratic form is simpler.

```
In [9]: np.dot(U,np.dot(G,U.T))
```

```
Out[9]: array([[ -2.48393534e-17,  -1.59462515e-17,   9.53708505e-17,
                 -9.32242397e-17,   1.24503425e-16,   2.00861438e-17,
                 -3.54957426e-17,  -4.62157635e-15,  -6.24500451e-17,
                  1.23165367e-16,   2.51534904e-17,  -1.55257751e-16,
                  7.42027967e-16,   2.64653750e-16,  -1.13624388e-16,
                  1.38777878e-16],
                [ -1.40311708e-17,   1.49828117e-17,   2.85049652e-17,
                 -4.67410610e-17,  -1.91656055e-17,   2.55172553e-17,
                  1.19951018e-17,   1.90669004e-15,   1.73352334e-17,
                 -5.73378287e-17,   1.11944528e-17,   8.23593263e-17,
                 -3.54747730e-16,  -3.20835558e-17,   5.97508671e-17,
                 -1.25996182e-16],
                [  9.34542337e-17,   2.65068099e-17,  -9.50641918e-18,
                  3.60323953e-17,  -1.00277389e-16,  -6.11975735e-17,
                 -8.43311379e-18,   3.82020055e-16,  -2.08166817e-17,
                 -5.20417043e-18,   3.38271078e-17,   1.21430643e-17,
                 -7.84962373e-17,  -7.54604712e-17,  -1.42247325e-16,
                  2.18575158e-16],
                [ -9.35308984e-17,  -5.09053414e-17,   3.89456528e-17,
                 -7.35980840e-18,   9.56775092e-17,   3.14325150e-17,
                  5.08286768e-17,  -1.52907686e-15,   5.20417043e-18,
                  4.33680869e-17,  -2.16840434e-17,  -5.89805982e-17,
                  2.44596010e-16,   7.76288755e-17,   1.04083409e-16,
                 -8.67361738e-17],
                [  1.23001385e-16,  -1.93119075e-17,  -9.92597306e-17,
                  9.59978493e-17,  -5.51395683e-17,   4.93350731e-17,
                  5.64775810e-18,   6.34920131e-15,   5.03069808e-17,
                 -1.62196645e-16,   1.73472348e-18,   2.23779328e-16,
                 -9.87491339e-16,  -3.54534110e-16,   2.60208521e-18,
                 -1.73472348e-16],
                [  1.97866896e-17,   2.59666420e-17,  -5.74627151e-17,
```

```
   3.27429056e-17,   4.65664833e-17,   1.83772268e-17,
  -2.61834825e-17,  -1.39759081e-15,   1.04083409e-17,
   4.59701721e-17,   2.60208521e-18,  -6.15826834e-17,
   3.06829215e-16,   5.05238212e-17,  -2.47198095e-17,
  -7.97972799e-17],
[ -3.31111559e-17,   1.42189735e-17,  -9.58185473e-18,
   4.59042230e-17,   5.70599016e-18,  -2.97061590e-17,
   1.32099207e-17,   1.11452186e-15,   1.78049183e-17,
  -2.75716677e-17,  -2.15215439e-17,   3.56419089e-17,
  -1.63070449e-16,  -8.07907189e-17,   6.22105499e-17,
  -4.46453499e-17],
[ -4.62550462e-15,   1.90208363e-15,   3.79362340e-16,
  -1.53707342e-15,   6.34813925e-15,  -1.40153460e-15,
   1.10767515e-15,   9.46183236e-16,   1.22905158e-15,
  -3.75567633e-16,  -1.28109329e-15,   4.00721123e-16,
  -1.06772230e-15,   2.94035629e-15,   5.79744586e-15,
  -8.41687831e-15],
[ -6.10134773e-17,   2.77029167e-17,  -2.87313576e-17,
   1.62630326e-18,   5.15267082e-17,   1.07471540e-17,
   1.99191125e-17,   1.22514845e-15,   6.62717378e-02,
  -3.83933119e-03,  -3.97579948e-03,   3.43199845e-03,
  -7.53832660e-03,   1.13301646e-03,   1.56631546e-02,
  -3.82501817e-02],
[  1.23192472e-16,  -5.58877083e-17,  -4.77048956e-18,
   4.74880552e-17,  -1.53495923e-16,   4.44387365e-17,
  -2.94690215e-17,  -3.69062420e-16,  -3.83933119e-03,
   6.37928815e-02,   4.54630745e-03,  -2.87282865e-03,
   7.84932892e-04,  -1.66111248e-02,  -1.98072089e-02,
   1.07828017e-02],
[  2.89753031e-17,   2.64280817e-18,   3.46944695e-17,
  -2.25514052e-17,   2.62919027e-18,   5.38035328e-18,
  -1.50705410e-17,  -1.27979224e-15,  -3.97579948e-03,
   4.54630745e-03,   6.43772530e-02,  -2.93597108e-03,
   1.83236757e-03,  -1.64136045e-02,  -2.30386121e-02,
   1.09659479e-02],
[ -1.57941151e-16,   8.25626142e-17,   9.21571847e-18,
  -6.12574227e-17,   2.15566497e-16,  -6.05662439e-17,
   3.28771934e-17,   3.94215910e-16,   3.43199845e-03,
  -2.87282865e-03,  -2.93597108e-03,   6.49180027e-02,
  -2.19573764e-02,   6.85742349e-03,  -6.39141301e-04,
  -1.42696062e-02],
[  7.33565903e-16,  -3.50213020e-16,  -7.66646708e-17,
   2.51613450e-16,  -9.88399269e-16,   3.12370201e-16,
  -1.60671542e-16,  -1.06678889e-15,  -7.53832660e-03,
   7.84932892e-04,   1.83236757e-03,  -2.19573764e-02,
   1.29496479e-01,  -1.23635813e-02,  -1.80495711e-02,
   8.28089487e-03],
[  2.61694854e-16,  -3.23047660e-17,  -7.89646110e-17,
```

```
        8.38711499e-17,  -3.65422154e-16,   6.31525226e-17,
       -7.40385251e-17,   2.94246673e-15,   1.13301646e-03,
       -1.66111248e-02,  -1.64136045e-02,   6.85742349e-03,
       -1.23635813e-02,   1.32302542e-01,   2.48415192e-02,
       -4.82032167e-02],
      [-1.13878576e-16,   5.55634625e-17,  -1.47289427e-16,
        1.04223075e-16,   1.48666549e-18,  -1.58603734e-17,
        6.56297769e-17,   5.79818319e-15,   1.56631546e-02,
       -1.98072089e-02,  -2.30386121e-02,  -6.39141301e-04,
       -1.80495711e-02,   2.48415192e-02,   1.92880756e-01,
       -8.14388121e-02],
      [ 1.60678762e-16,  -1.31417193e-16,   2.09251019e-16,
       -9.32413868e-17,  -1.68485018e-16,  -8.35919875e-17,
       -4.80605868e-17,  -8.41584831e-15,  -3.82501817e-02,
        1.07828017e-02,   1.09659479e-02,  -1.42696062e-02,
        8.28089487e-03,  -4.82032167e-02,  -8.14388121e-02,
        2.72106443e-01]])
```

As proved in Theorem 1, only the lower right quadrant of this matrix is non-zero; moreover this quadrant is positive.

```
In [10]: Gred=np.dot(U,np.dot(G,U.T))[8:15].T[8:15]
         scipy.linalg.eigvalsh(Gred)

Out[10]: array([ 0.05383961,  0.05894793,  0.05949194,  0.0657521 ,  0.12245141,
                 0.13478487,  0.2187718 ])
```

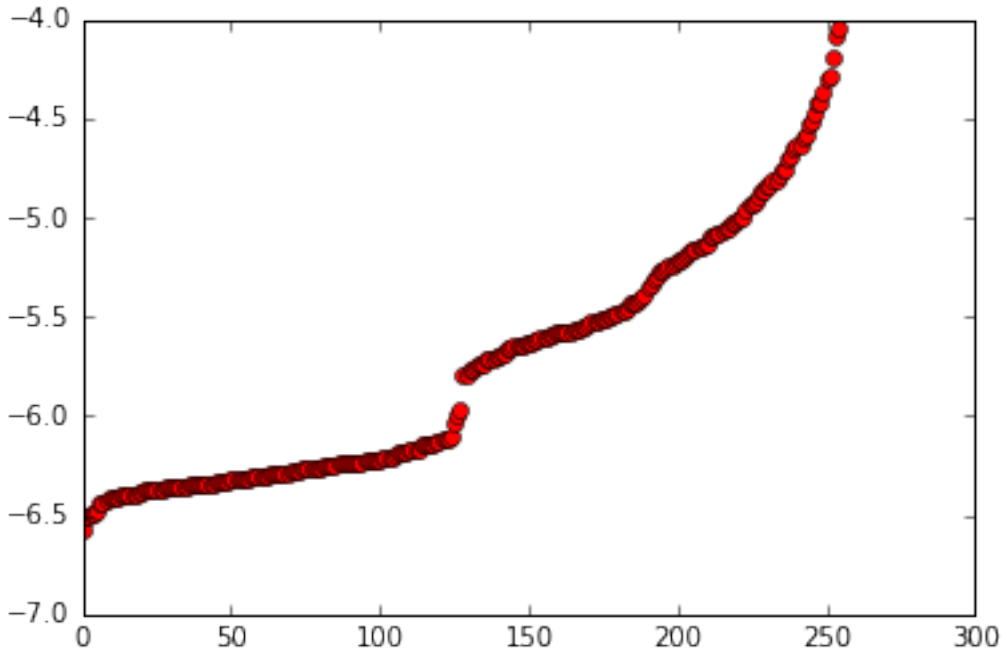
This code has exponential complexity in d (both in time and in memory). There is numerical evidence that the spectral gap of the Markov chain is uniform in d (as long as the coefficients C are uniformly away from zero and infinity), and the code works on a laptop up to $d = 9$ or 10.

```
In [40]: import matplotlib.pyplot as plt
         C=generate_C(9)
         G=curvature(C)
         U=np.dot(fullbasis(9),rotation(9))
         Gred=np.dot(U,np.dot(G,U.T))[2**8:2**9-1].T[2**8:2**9-1]
         vals=scipy.linalg.eigvalsh(Gred)
         plt.plot(np.log(vals),'ro')
         plt.show()

Markov chain convergence after 76 steps
Equilibrium measure:  [ 0.00182074  0.00171958  0.00226584  0.0014632   0.00231165  0.00163143
  0.00166615  0.00146926  0.00212249  0.00191125  0.00193759  0.00141323
  0.00211753  0.00120598  0.0017097   0.00156215  0.00174398  0.00248309
  0.00171972  0.00231592  0.00166662  0.00229871  0.00197373  0.00120172
  0.00191208  0.00222087  0.00141419  0.0018427   0.00203398  0.0015287
  0.00166264  0.00152818  0.00205464  0.0018762   0.00244509  0.00284443
  0.00185977  0.00205537  0.00221001  0.00179184  0.0021826   0.00123072
  0.00205751  0.00210923  0.00202925  0.00203106  0.00150642  0.00158941
```

0.00204368 0.00212084 0.00132315 0.00249355 0.00247656 0.00182442
0.00254107 0.00181021 0.00195086 0.00183609 0.00228872 0.00191343
0.00282291 0.0018159 0.00142802 0.0026794 0.00159307 0.00199645
0.00196948 0.00162603 0.00244129 0.00229253 0.0025905 0.0027909
0.00185204 0.00229168 0.00292287 0.00184314 0.00179025 0.00167287
0.00197188 0.00149402 0.00158403 0.00260014 0.00117483 0.00158126
0.00227614 0.00240011 0.00228841 0.00199589 0.00214255 0.00131948
0.00159705 0.0021868 0.00177656 0.00180375 0.00160163 0.00151739
0.00144288 0.00287124 0.00212351 0.00159015 0.00135988 0.00189132
0.00264445 0.00266209 0.00238811 0.00182233 0.00141831 0.00197444
0.00193331 0.00218848 0.00174873 0.00233014 0.00148464 0.00213652
0.0019828 0.00181789 0.00162677 0.0023608 0.00184469 0.00223353
0.00220672 0.00214329 0.00179007 0.00237229 0.0021406 0.00165441
0.00214387 0.00184124 0.00172811 0.00181065 0.00190829 0.00159596
0.00200196 0.00157249 0.00199023 0.00150183 0.00283258 0.00225076
0.00199373 0.00176332 0.00230382 0.00265721 0.00253662 0.00278824
0.0012759 0.00179376 0.00274975 0.00161681 0.00192273 0.00275499
0.00162367 0.00157848 0.00170333 0.0020718 0.00192003 0.00204529
0.00219366 0.00179578 0.0017681 0.0015112 0.00154827 0.00155966
0.00246325 0.00268183 0.00152769 0.00123326 0.00189443 0.00241045
0.00207869 0.00215239 0.00220179 0.00179722 0.00201572 0.00193141
0.00157251 0.00182535 0.00231192 0.00165787 0.00179112 0.00130437
0.00222131 0.00125325 0.00190884 0.00192446 0.00186941 0.00188139
0.00169698 0.00159641 0.00171535 0.00178478 0.00204674 0.00175496
0.00169964 0.00191544 0.00217447 0.00228292 0.00171036 0.00196821
0.00179875 0.00212426 0.00210776 0.00162577 0.00153627 0.00214856
0.00297075 0.00171897 0.00171668 0.00230192 0.00262747 0.00231652
0.00199065 0.00204184 0.00174804 0.00096097 0.00167941 0.00179839
0.00183129 0.00214661 0.0020734 0.00313493 0.00187916 0.00211229
0.00220283 0.00198168 0.00178339 0.00164796 0.00170018 0.00272871
0.00198408 0.00178107 0.00174924 0.00184431 0.00143218 0.00155266
0.00229809 0.001869 0.0023847 0.00183263 0.00227553 0.00155863
0.00219081 0.00187927 0.00219049 0.00161242 0.00165426 0.00234739
0.00209173 0.00187538 0.00249652 0.00202148 0.00162906 0.00138553
0.00194221 0.00199895 0.0016554 0.0023544 0.00171958 0.00200946
0.00167724 0.00167221 0.00172209 0.0017194 0.00165736 0.0018026
0.00210458 0.00212439 0.00202774 0.00176221 0.00201542 0.0020509
0.00185297 0.00162867 0.00218686 0.00280644 0.00219542 0.00168593
0.00174669 0.00186802 0.00208658 0.00189411 0.00225244 0.00159583
0.00288679 0.00250858 0.00175297 0.00267346 0.00297617 0.00257924
0.00153488 0.00171931 0.00228872 0.00253697 0.00228394 0.00271065
0.00125311 0.00167405 0.00200157 0.00152537 0.00261875 0.00217506
0.00143278 0.00175279 0.00207389 0.00152961 0.00227044 0.00159282
0.00192805 0.00281299 0.00173388 0.00156833 0.00158073 0.00226866
0.0016703 0.0019646 0.00169885 0.00216479 0.0015271 0.00234646
0.00236698 0.00130571 0.00194569 0.0015078 0.00160496 0.00186602
0.00264205 0.00146453 0.00237053 0.00253395 0.00121763 0.00207489
0.00175485 0.001359 0.00198488 0.00229245 0.00201757 0.00178528

```
0.0015239   0.00254494  0.00158612  0.00272362  0.00195493  0.0015989
0.00165872  0.00140197  0.00182723  0.001776    0.00187751  0.00164651
0.00197424  0.00148964  0.0018985   0.00228431  0.0021722   0.00158615
0.00155506  0.00233286  0.00237364  0.00179351  0.00204526  0.00135651
0.00255588  0.00221015  0.0012907   0.00150336  0.00204459  0.00301984
0.00224272  0.00185438  0.00194671  0.00229237  0.00178235  0.00177566
0.00135808  0.0018063   0.00237263  0.00160063  0.00186336  0.00165962
0.00221159  0.00159482  0.00237741  0.00136019  0.00179729  0.00216856
0.00200093  0.0015388   0.0015332   0.001864    0.00222701  0.00221747
0.0020761   0.00197981  0.00216071  0.00163059  0.00162099  0.00221737
0.00154445  0.00273816  0.00188981  0.00276717  0.00197828  0.00303194
0.00224484  0.00131035  0.00160421  0.00203882  0.00156191  0.00202502
0.00215993  0.00266924  0.00138218  0.0018041   0.00144124  0.00206786
0.00210547  0.00216149  0.00190521  0.00191133  0.00164333  0.00222266
0.00176483  0.00188059  0.0023829   0.00139239  0.00199016  0.00215735
0.00135204  0.00126348  0.00158752  0.0015926   0.00189138  0.00235746
0.00144643  0.00223005  0.00237603  0.0020974   0.00254472  0.00154081
0.00315559  0.00217264  0.00236967  0.00167661  0.00146739  0.00237685
0.00180764  0.00202162  0.00169085  0.0022109   0.00184009  0.00148177
0.00227     0.00177299  0.00208094  0.00187014  0.00248385  0.00253272
0.00290246  0.00192941  0.00210676  0.00143837  0.00185842  0.00146731
0.00179242  0.00196504  0.00118907  0.00154947  0.00165477  0.00173345
0.00239947  0.00165454  0.00150071  0.00245045  0.00184519  0.00232682
0.00201213  0.0021933   0.00216712  0.00173196  0.00162643  0.00192007
0.00153847  0.00239503  0.0022509   0.00228787  0.0028478   0.00176406
0.00157649  0.00191316  0.00130635  0.00183556  0.00175592  0.00208216
0.00178732  0.00237281  0.00162355  0.00198786  0.00174269  0.0026595
0.00242137  0.00187723  0.00148765  0.00149069  0.00206841  0.00173603
0.00190567  0.00227712  0.00134928  0.0024189   0.00224058  0.00176922
0.0023544   0.00228699]
```



3 Appendix

3.1 Lemmas: the structure of the null space

Lemma 1 Let ls be the operator of left shift on $(\mathbb{C}^2)^{\otimes d}$, which acts on product states as

$$ls(v_1 \otimes \ldots \otimes v_d) = v_2 \otimes \ldots \otimes v_d \otimes v_1.$$

Then

$$Ker(\sum_{k=0}^{d-1}(ls)^k) = Im(Id - ls).$$

Proof of lemma 1

Lemma 2 Let (a, b) an orthonormal basis of $\mathbb{C}^2$. Let E be the following subspace of $(\mathbb{C}^2)^{\otimes d}$:

$$E = \{a \otimes v - v \otimes a \,|v \in (\mathbb{C}^2)^{\otimes d-1}\}.$$

Let

$$F = \{b \otimes v \otimes b \,|v \in (\mathbb{C}^2)^{\otimes d-2}\}.$$

Moreover, for a word $w \in \{a, b\}^d$, to which we naturally associate an element of $(\mathbb{C}^2)^{\otimes d}$, we let $k(w)$ be the minimal k such that $ls(w)$ begins with an a. We then define

$$G = Span\{\sum_{i=0}^{k(awb)}(ls)^i(awb) \,|w \text{ of length d-2}\}.$$

Then the four following spaces are in direct orthogonal sum:

$$(\mathbb{C}^2)^{\otimes d} = E \oplus F \oplus G \oplus \mathbb{C}(b \otimes \ldots \otimes b).$$

Proof of lemma 2
Lemma 3 If (a, b) is an orthonormal basis of $\mathbb{C}^2$, then

$$\{a \otimes v - v \otimes a | v \in (\mathbb{C}^2)^{\otimes d-1}\} = Im(Id - ls) \cap \{b \otimes w \otimes b | w \in (\mathbb{C}^2)^{\otimes d-1}\}^{\perp}.$$

Proof of lemma 3