A Class of Real Expander Codes Based on Projective-Geometrically Constructed Ramanujan Graphs

B.S. Adiga, M. Girish Chandra and Swanand Kadhe,

Innovation Labs, Tata Consultancy Services, Bangalore, INDIA

Summary
Quite recently, codes based on real field are gaining momentum in terms of research and applications. In high-performance computing, these codes are being explored to provide fault tolerance under node failures. In this paper, we propose novel real cycle codes based on expander graphs. The requisite graphs are the Ramanujan graphs constructed using incidence matrices of the appropriate projective-geometric objects. The proposed codes are elegant in terms of reduced complexity encoding and very simple erasure correction. Further, the codes are guaranteed to correct three erasures. Apart from building the codes from the sound existing principles, necessary simulation results and justification of the useful properties are also presented in the paper.

Key words:
Fault Tolerant Computing, Real Number Codes, Ramanujan Graphs, Expander Codes, Cycle Codes

1. Introduction

The ability to detect and correct errors in communications, storage and computing is of both fundamental and practical importance. Error control codes providing this capability are well researched and also many related practical issues are addressed from last sixty years. Even though, the error control codes are most often defined over finite fields (or Galois fields), one can see codes defined over real-number or complex-number fields in the literature. These real-number or complex-number codes can be advantageous in certain situations and some aspects; the advantages in general are captured in [1]. These codes were initially approached from a signal processing perspective [2], and some of the applications include coding for Orthogonal Frequency Division Multiplexing (OFDM) transmission over fading wireless channels [3] and in High Performance Computing (HPC) ([4],[5],[6],[7]). In high-performance computing, these codes are explored to provide fault tolerance for the smooth execution of sophisticated scientific applications under node failures. Both diskless checkpointing and check-point free scenarios are considered while examining these codes in [6], [7]. The concepts and issues related to diskless check pointing are well known in the computing community; see [8], [9] and references therein for details. Cursorily, checkpointing is one of the techniques used in fault-tolerant computing, where the "checkpoints" are stored in stable storage (i.e. disk), since the stable storage typically survives processor failures [8]. The goal of diskless checkpointing is to remove stable storage and replace it with memory and processor redundancy [8]. The redundant information in the spare processors is obtained through coding (parity or Reed-Solomon codes) and in this sense diskless checkpointing is akin to software-implemented Redundant Array of Independent Disks (RAID) technology [9], [10]. In linear algebra computations involving a large dense matrix, the application involved modify a large amount of memory between consecutive checkpoints, introducing considerable overhead (see [9] and the references therein). In these cases, checkpoint-free fault tolerance is useful [9]. The benefits of using real or complex number codes in HPC are covered in [8]. Further, as mentioned in [9], the real (or complex) number codes can be used in other fields like fault tolerant combinatorial and dynamic systems, Compressive Sensing (CS) (see [11],[12],[13],[14]), and Network Coding ([15]).

It is well known that the error correction involves finding the location of errors and their values to reconstruct the original data. When the positions of errors are known, we have a simpler case of erasure correction, where the values (magnitudes) of the "erased" digits are estimated. The error correction for real (or complex) codes may involve l_1-norm minimization (also called as basis pursuit) and greedy algorithms such as various matching pursuits, based on the recent CS framework ([11],[12],[13],[14]). Depending on the encoding or the choice of generator matrix, reduced complexity decoding is also possible (see [14]), including very simple majority-logic decoding [1]. In this paper, we restrict to erasure correction which involves less complex decoding algorithms compared to error correction.

Manuscript received January 5, 2011
Manuscript revised January 20, 2011

One of the main problems of real (or complex) number codes is the contamination of the codeword due to computations involving the digital representation of: (a) the data (b) the coefficients of generator or parity check matrices and (c) the results of encoding and decoding operations. This contamination can lead to numerical instability as elaborated in [4], [6] in the context of HPC. The argument is based on the following facts [6], [7]: (1) most of the error correction procedures involve solving a system of linear equations (2) in computer floating-point arithmetic no computation is exact due to round-off errors (3) in solving a system of linear equations using computer, a condition number of 10^p for the coefficient matrix leads to a loss of accuracy of about *p* digits (4) the generator matrices of many of the real (or complex) number codes contain ill-conditioned submatrices (i.e., numerically indistinguishable from singular submatrices) (5) certain error patterns can thus result in ill-conditioned system of linear equations for the error correction causing the loss of precision of possibly all digits in the recovered numbers. Designing appropriate real number coding scheme to correct different number of erasures is thus a challenging problem. In [7], a class of Reed-Solomon (RS) style erasure correction codes are presented, which are numerically best in the sense that the each of the generator matrix obtained has the condition number of the worst-conditioned sub matrix minimized [7]. The two-erasure correction codes are constructed analytically and the codes to correct three or more erasures are obtained through computations based on an approximation method. Based on the chosen framework by the author, it is also proved in [7] that it is impossible for any minimum redundancy code to correct all erasure patterns, when the number of erasures is more than one, for large generator matrices. In the HPC scenario, the case of large generator matrix corresponds to large number of processors.

Viewed from a slightly different perspective and following [1], a real number erasure code would be really elegant if (1) The code is guaranteed to correct certain number of erasures (2) The encoding and decoding operations involve fewer number of additions and subtractions only, rather than the multiply and accumulate (MAC) operations. This eliminates errors in representing the elements of generator and parity check matrix and also the errors involved in representing the product of two real numbers. (3) The erasure correction capability remains the same as the code length or generator matrix size increases. In a nutshell, we need a family of real-number codes which are asymptotically good and efficient, which simply means, as the code length *n* increases, the error-correction capability and rate are always maintained above some fixed values, and the encoding and decoding can be performed in polynomial time in *n*, respectively (see pp.46 of [16]).

Of late, expander codes have emerged as popular family of asymptotically good and efficient codes. Following [17] and [18], we consider the family of expander codes parameterized by a fixed-size linear code with some small block length and a family of expander graph with constant degree (say, d); the degree and the block length need to be matched. Using the small fixed-size linear block code C (can also be referred to as sub code or component code) and an expander graph G , one can build a larger linear code, which is the expander code, denoted as $C'(G,C)$. The rate and the error correction property of this expander code depend on the rate and distance of the component code as well as the spectral expansion of the expander graph [16]. If the component code C is the parity code, then the expander code $C'(G,C)$ is a *cycle code*, which we refer to as the *cycle expander code*, as the underlying graph is an expander graph. On the other hand, if the underlying graph G is not an expander graph, we end up with the conventional cycle codes. See [19], [20] (and the references there in) for more details on cycle codes. The parity code mentioned has block length d , message or information length $d-1$ and the minimum distance of 2.

In this paper, we propose a novel family of *real number* erasure correction *cycle expander codes* which are *guaranteed to correct three erasures*. The component codes are simple parity-based codes over the real field. The expander graphs chosen are the Ramanujan graphs constructed using the Projective Geometry (PG) based incidence matrices [21], [22], [23]. Some of the advantages of using the PG-based graphs are highlighted in [21],[22],[23]. The code construction follows that of Zemor [18] and involves using the edge-vertex incidence graph G' of the Ramanujan graph G . See also [17] for details on edge-vertex incidence graphs. These cycle expander codes facilitate reduced complexity encoding and further, the erasure correction with the proposed expander codes involves only additions and subtractions, rather than the MAC operations. The errors which may occur due to these computations can be taken care of by simply having a "longer" accumulator. The proposed expander codes are asymptotically good in the sense that the three-erasure correction capability is intact with the increase in block length. The rest of the paper is organized as follows. In Section 2, the necessary ingredients of the proposed expander code are briefly covered. Included here are the relevant details about cycle codes, expander graph preliminaries, PG based incidence matrices and the associated Ramanujan graphs, edge-vertex incidence graph and some remarks on decoding. In Section 3, a conceptual proof is provided towards the three-erasure correction capability of the proposed code.

Section 4 presents some relevant simulation results and discussion. Finally, conclusions are provided in Section 5.

2. Nuts and Bolts of the Proposed Code

The aim of this section is to cover the necessary nuts and bolts for the proposed expander code, including the relevant basics and terminology. Any coding scheme necessarily implies appropriate decoding strategy as well. Thus, the aspects related to decoding (in particular, erasure correction), are also touched upon. Since the code of interest is a cycle code based on expander graphs, we start with some necessary information about cycle codes.

2.1 Cycle Codes over Binary and Real Fields

In this section, the required concepts and definitions are borrowed heavily from [19] and [20]. Cycle codes are defined over graphs. As customary, we consider an undirected connected graph $G=(V,E)$, where V and E are the set of vertices and edges of G respectively. If an edge e connects a vertex v to some other vertex, then e is said to be incident to v. The graph induced by a subset E' of edges is the graph (V',E'), where V' is the set of vertices incident to at least one edge in E'. A cycle in a graph G is a collection of edges such that in the graph induced by them, all vertices have even degree. The cycle code associated to G, denoted as $C(G)$, is a binary code of block length n equal to the number of edges, i.e. $n=|E|$, and every code word corresponds to a cycle. From the construction perspective, it is useful to consider a spanning tree of G (which is a tree containing all vertices of G), since a convenient way of generating a cycle is by adding an edge to the spanning tree. Considering a spanning tree T and its complement $\overline{T}$, each edge e_i in $\overline{T}$ form a unique cycle c_i with those edges in T that form a path between the nodes at which e_i is incident. If the k edges in $\overline{T}$ (where $k=n-m+1$ with m being the number of nodes in G) and the remaining $m-1$ edges of T are numbered $e_1,e_2,\cdots,e_k$ and $e_{k+1},e_{k+2},\cdots,e_n$ respectively, then each cycle c_i can be represented by a binary vector $\boldsymbol{c}_i=[c_{i1}\quad c_{i2}\quad\cdots\quad c_{in}]$ with $c_{ij}=1$ if e_j is the edge in c_i and 0 otherwise. The vectors $\boldsymbol{c}_i$, $i=1,2,\cdots,k$ form the rows of the generator matrix $\boldsymbol{G}$ of the cycle code and hence are the basis vectors for the linear space spanned by the cycles of G. In a nutshell, $C(G)$ can be viewed as a subspace of F_2^n of cycles of G, where F_2^n is the n-dimensional vector space over the binary field F_2. As far as encoding is concerned, the cycle codes can be encoded in linear time. This follows from the theorem in [19] suggesting a linear time algorithm to calculate the values on the edges in T given the values on the edges in $\overline{T}$, which are the information bits. Before, remarking on the decoding of the cycle codes, it is worth noting that the examination of parity check matrices of cycle codes suggests that they can be viewed as Low Density Parity Check (LDPC) codes [24] with column weight 2. The associated Tanner graph of the parity check matrix can be used for decoding using different flavors of sum-product algorithms [25]. For erasure correction, simple Exclusive OR (XOR) operations are sufficient, where the idea is, if each node of the graph has d neighbors and if the values of $d-1$ edges are known, the remaining value can be determined as the XOR of the rest. The process is iterated until no new values can be determined.

So far, the discussion presented is for the cycle codes over binary fields. Since the focus of the paper is on real-number codes, how to extend these binary codes is really important. But, this can be very simply accomplished by treating the generator matrix as a matrix over the real field to generate the real-number code word from real number information symbols. This is similar in principle to that followed in [1]. For the erasure correction, the value of the requisite edge is obtained as the negative of the sum of the known $d-1$ edge values, again a simple extension of the XOR operation of the binary case.

The simple structure coupled with easy encoding and decoding of cycle codes make them really attractive and they have been under intense studies since the early days of coding theory to recent times [20]. It is worth examining these codes over *real field* when the underlying graph G is an expander. To the best of our knowledge, a study in this direction is not reported elsewhere. We start with a background on expander graphs before arriving at projective-geometrically constructed Ramanujan graphs, which are a class of expander graphs.

2.2 Expander Graphs

Informally, an expander graph is a graph $G=(V,E)$, in which every subset of vertices V_s expands quickly, in the sense that it is connected to many vertices in the set $\overline{V}_s$ of complementary vertices [26]. An Expander Graph is interesting as it satisfies two conflicting but desirable

properties: (1) being sparse and (2) "well connected". The property of well connectedness or expansion can be more formally defined in terms of edge expansion rate or vertex expansion ratio, the latter being more suitable for bipartite graphs (see [16],[26],[27] for more details). The expansion property is related to the first two largest eigenvalues of the adjacency matrix of the graph [16],[26]. The difference of these eigenvalues is known as spectral gap [16],[26]. A useful result to note at this juncture is that the edge expansion rate $h(G)$ of a *d*-regular graph *G* (i.e. every vertex of *G* has degree *d*) is bounded as in Eq.1

$$\frac{d-\lambda}{2} \le h(G) \le \sqrt{2d(d-\lambda)} \quad (1)$$

where λ is the second largest eigenvalue; of course, the largest eigenvalue in this case is *d*. Thus, smaller the λ for a given *d*, better is the expansion. The expander codes are based on explicit construction of expander graphs and include codes based on bipartite expander graphs, Sipser and Spielman constructions using regular expander graphs and Zemor's construction considering the special case of Ramanujan's graphs (see [16],[17],[18] for more details). A finite, connected *d*-regular graph is a Ramanujan graph if it satisfies

$$\lambda \le 2\sqrt{d-1} \quad (2)$$

Roughly, the Ramanujan graphs have large spectral gap [16]. There are various ways to construct Ramanujan graphs, and we would be using the PG for their explicit construction, following [21],[22],[23].

2.3 Projective Geometry Based Ramanujan Graphs

The projective geometry (PG) is essentially a geometric realization of linear algebra [28]. Restricting to the finite PG, a PG of dimension *l* and order is denoted as $PG(l,q)$, where *q* is a prime or power of a prime. $PG(l,q)$ is constructed using a vector space *S* of dimension $(l+1)$ over the field F_q [29]. This $(l+1)$-dimensional object can be viewed in geometrical terms, by calling the one-dimensional subspaces of *S* as points, 2-dimensional subspaces as lines, the 3-dimensional subspaces as planes, and the *l*-dimensional subspaces are called hyperplanes [29]. The *r*-dimensional objects described by $(r+1)$-dimensional subspaces are called as *r*-flats. Treating the geometric objects this way is justified by observations like any two points are precisely one common line since two different 1-dimensional subspaces (points) generate precisely one 2-dimensional space (a line) [29].

A $PG(l,q)$ has $N = \frac{(q^{l+1}-1)}{(q-1)}$ points and equally many hyperplanes. A useful notion in PG is that of homogeneous coordinates. The elements of the underlying vector space $S = F_q^{l+1}$ are the $(l+1)$-tuples $(x_1, x_2, \cdots x_{l+1})$ of field elements. Let $P = F_q(x_1, x_2, \cdots x_{l+1})$ be the 1-dimensional vector space generated by the nonzero $(l+1)$-tuple. As any nonzero scalar multiple of $(x_1, x_2, \cdots x_{l+1})$ generates the same "point" *P*, it is customary to write $P = (x_1 : x_2 : \cdots : x_{l+1})$ called as homogeneous coordinates with the implication of

$$(x_1 : x_2 : \cdots : x_{l+1}) = (\lambda x_1 : \lambda x_2 : \cdots : \lambda x_{l+1}) \quad (3)$$

for every $0 \ne \lambda \in F_q$ [29]. The term homogeneous coordinates makes sense as it simply means scaling is unimportant [30]. The homogeneous coordinates can be assigned naturally to hyperplanes as well [29]. Each hyperplane can be described as the set of points satisfying a nontrivial linear equation. In other words, the hyperplane consisting of all points $(x_1 : x_2 : \cdots : x_{l+1})$ such that

$$x_1 y_1 + x_2 y_2 + \cdots + x_{l+1} y_{l+1} = 0 \quad (4)$$

has homogeneous coordinates $[y_1 : y_2 : \cdots : y_{l+1}]$ [29].

The next concept relevant to us is that of *incidence* relations. They are simply the binary relations decsribing how the geometric objects (or equivalently the subspaces) meet [31]. Examples include, "lies on" between points and hyperplanes (as in "point *P* lies on hyperplane *H*"), and "intersects" (as in "line L_1 intersects line L_2") [31]. The incidence relationship we are interested in is that of between points and hyperplanes. This relationship can be captured in terms of a square incidence matrix as the number points is equal to the number of lines (which is *N*). This incidence matrix can then be used to generate a bipartite graph, which in our case is the Ramanujan graph we are looking for. It is to be noted that the requisite Ramanujan graphs can be obtained using PG over any field, including extension fields. However, for the results presented in this paper, we examined the graphs generated by the point-hyperplane incidence matrices of $PG(l,2)$,

i.e., PG over binary field F_2. An example is presented in the following towards a better comprehension of the construction.

Example: We consider the *projective plane* $PG(2,2)$ corresponding to $l=2$ and for this case we get 7 points and 7 hyperplanes (here, they are lines). The incidence matrix $\boldsymbol{M}$ for this case is given by

$$\mathbf{M}=\begin{bmatrix}1&1&0&1&0&0&0\\0&1&1&0&1&0&0\\0&0&1&1&0&1&0\\0&0&0&1&1&0&1\\1&0&0&0&1&1&0\\0&1&0&0&0&1&1\\1&0&1&0&0&0&1\end{bmatrix} \quad (5)$$

For this incidence matrix, the corresponding bipartite graph can be drawn as shown in Fig.1. A bipartite graph $G=(V_L \cup V_R, E)$ is called a (c,d)-regular graph if every vertex in V_L has degree c and every vertex in V_R has degree d [32]. The subscripts L and R refer to left side and right side respectively; a convention usually followed to group two sets of vertices. For better utilization of space, the bipartite graph in Fig.1 is tilted, making the left and right sides shifted to upper and lower sides, respectively. This display aspect is followed in the rest of the paper also. As the number of hyperplanes is equal to the number of points, we have $|V_L|=|V_R|=7$ in this case and further, $c=d=3$. These equalities of $|V_L|=|V_R|$ and $c=d$ are also true for other values of l and $q=2$. Thus, the graphs obtained this way are balanced d-regular bipartite graphs. Further, all these graphs satisfy Eqn.2 and thus are Ramanujan graphs. Continuing with our example, for $PG(l,2)$, we have $\lambda=1.4142$ which is less than $2\sqrt{d-1}=2.8284$.

It is useful to note that the Adjacency Matrix $\boldsymbol{A}$ required to compute the λ can be obtained from the incidence matrix $\boldsymbol{M}$ using the relationship

$$\mathbf{A}=\begin{bmatrix}\mathbf{O}&\mathbf{M}\\\mathbf{M}^T&\mathbf{O}\end{bmatrix} \quad (6)$$

Even though we considered the case of projective plane $PG(2,2)$ for simplicity and ease of depicting the graphs, the graphs obtained from $PG(l,2)$ with $l\geq 3$, i.e., higher-dimensional *projective spaces*, are very useful as well. The codes based on the graphs of projective spaces are also capable of guaranteeing 3-erasure correction and further can support higher code rates (see Section 3 and Section 4). The Ramanujan graphs so obtained can be used as "base graphs" to construct the requisite expander codes as discussed next.

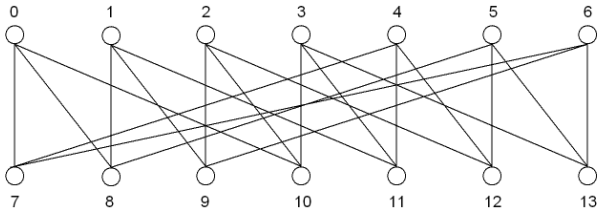

Fig. 1 Bipartite graph generated by the point-hyperplane incidence relations of $PG(2,2)$.

2.4 Zemor Construction of Expander Codes

Sipser and Spielman [17] suggested a construction of asymptotically good and efficient linear codes using d-regular expander graphs. By using bipartite Ramanujan graphs, Zemor [18] constructed the expander codes capable of correcting 12 times more errors compared to that of Sipser and Spielaman codes, without compromising the complexity of the decoding. As mentioned in the beginning of the paper, our construction is similar to Zemor's construction, but utilizes PG- based Ramanujan graphs, including those from higher dimensions. As the expander codes we are focusing are a class of LDPC codes, any construction should lead to a bipartite (Tanner) graph, with the appropriate identification of variable and constraint nodes. Similar to [18] (see also [17]), this is obtained using the edge-vertex incidence graph of PG-based Ramanujan graph $G=(V_L \cup V_R,\ E)$. This edge-vertex incidence graph $G'=(V_L' \cup V_R',\ E')$ has $V_L'=E$ and $V_R'=V_L \cup V_R$. Each "vertex" in V_L' is joined to both of its end points, hence one in V_L and another in V_R [32]; the edge set E' of G' is obtained this way. The resultant bipartite graph is $(2,d)$- regular graph. Since G' is constructed from the PG-based Ramanujan expander graph G, it also exhibits good expansion properties [33].

Continuing with our illustrative example, the resultant edge-incidence graph is as shown in Fig.2 with $|V_L'| = 21$, $|V_R'| = 14$, and $d = 3$. Also, as shown in the figure, it is useful, towards better visualization of decoding, to keep the vertices corresponding to V_L and V_R in the respective groups, rather than mixing them up (see Section 3).

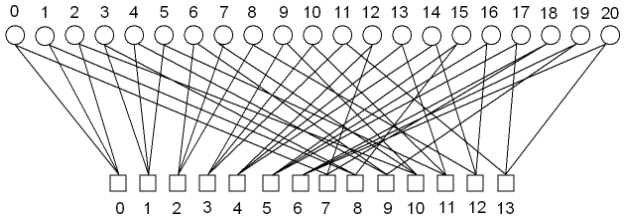

Fig. 2 Edge-vertex incidence graph of the bipartite graph in Fig.1.

To arrive at the expander codes, the next steps involved are (1) identifying the variable and constraint nodes of G' and (2) to bring component code into the scenario through constraint and variable nodes interaction. The vertices in $V_L' = \{v_1 \quad v_2 \quad \cdots \quad v_n\}$ and $V_R' = \{p_1 \quad p_2 \quad \cdots \quad p_t\}$ can be identified with the variable nodes and constraint nodes respectively [18],[32]; here $n = |V_L'|$ and $t = \frac{2n}{d}$. Each of the constraint nodes is connected to *d* variable nodes and for the expander code, it is required that these *d* variable nodes form a linear code word of length *d*. Because the restrictions imposed on the variables are linear, the resulting expander code will be linear as well. It is useful to define a mapping function $b(i, j)$ towards a formal definition of expander code, where, $b(i, j)$ returns the *j*-th neighbor (v_j) of the vertex $p_i \in V_R'$ [32].

Definition of Expander Code [32]: Let *C* be an error correcting code of block length *d*. The expander code $C'(G, C)$ consists of all code words $[x_1 \quad x_2 \quad \cdots \quad x_n]$ such that for all *i* (from 1 to *t*)

$$[x_{b(i,1)} \quad x_{b(i,2)} \quad \cdots \quad x_{b(i,d)}] \in C \qquad (7)$$

As remarked in Section 1, the performance of the expander code depends on both the expander graph and the component code. The formal quantification of the performance metrics are well documented (see [16],[19],[32], [33] for example). It is sufficient to note that if G' is a good expander and if the constraints are identified with sufficiently good codes, then the resulting expander code will be a good code [19]. As far as the component code *C* is concerned, different flavors are possible including RS codes [21],[22].

2.5 Cycle Expander Codes

For our real expander codes, the component code is the parity code as mentioned in Section 1. But, we need a systematic constructive procedure to carry out the encoding as it is necessary to ensure that neighbors of each of the constraint nodes are code words. Needless to say, this cannot be done in an ad hoc fashion. By following the concepts and methodology presented in Section 2.1 this can be carried out using the (complementary) spanning tree of G. Of course, the base graph G and the edge-vertex incidence graph G' are closely related; recollect that each variable node in G' corresponds to an edge in G, whereas each constraint node in G' corresponds to a node in G. Now the turn of decoding: the erasure correction decoding happens on the graph G' iteratively as mentioned in Section 2.1.

In a nutshell, using the procedure suggested in Section 2.1, we can construct useful real cycle expander codes which are asymptotically good and efficient as well as guaranteed to correct three erasures. A conceptual proof for the latter is provided next.

3. A Conceptual Proof for Three-Erasure Correction Capability

To prove the three-erasure correction capacity of the proposed code, it is sufficient to show that, it is possible to correct *all* combinations of three erasures, whereas there are certain four and more erasure locations which cannot be corrected. Based on the detailed examination of PG graphs, it is possible to rigorously prove the error correction properties of the expander codes. These are available in [22] and would be submitted for publication soon. Here, we rather take a conceptual approach towards bringing out the three-erasure correction capability of the real cycle expander codes.

How the proposed codes are capable of correcting any pattern of three erasures can be conceptually brought as follows. As seen in Section 2.4, each variable node of the edge-vertex incidence graph is connected to two constraint nodes, one in V_L and another in V_R. Thus, if a code symbol (corresponding to variable node) is erased, it has to be corrected by one of the two constraint nodes. A constraint node can correct an erased symbol provided the remaining $d-1$ variable nodes connected to it are intact. Now, note that a variable node in G' corresponds to an

edge in G and there are no multiple edges in G. Using these facts, it can be observed that any group of three variable nodes is connected to at least four constraint nodes. Further, there are at least two of these four constraint nodes which are connected to only one variable node in this group. Hence, if the symbols corresponding to variable nodes in this group are erased they can be corrected over the iterations, starting from the correction by the constraint node, which is "seeing" one erased symbol. This corrected information is fed back in the Tanner graph to facilitate the correction of other errors.

The next task is to bring out the fact that there are *certain* four-erasure locations which cannot be corrected. It is to be noted that certain four erasure patterns, in fact, many of them can be corrected as the proposed code demonstrates graceful degradation in erasure correction performance (see Section 4).

A group of erased nodes cannot be corrected if *both* the constraint nodes connected to any erased node, are connected to at least one other erased node. Since the edge-vertex incidence graph is constructed from the point-hyperplane incidence graph (i.e. the base graph), it is possible to carefully examine the base graph in order to find out the variable nodes that form such a deadlock in the edge-vertex incidence graph. One way to find out such variable nodes from the base graph is as follows. First, consider the plane obtained by the intersection of any two hyperplanes; we call this plane as *base plane*. For example, if $l = 3$ and considering the intersection of first and last hyperplanes, we get the base plane (base line in this case) as $\{0,\ 11,\ 14\}$. Then, find the hyperplanes which are incident on this base plane (base line). For the case of $l = 3$, these are given by 0, 1, and 4. Now, consider the subgraph of the base graph, which corresponds to the incidence relation of the points on the base line (i.e. 0, 11, 14) and the hyperplanes incident on the base line (i.e. 0, 1, 4). This is shown in Fig. 3. Note that in the base graph, usually the vertices corresponding to the points are labeled from 0 to 14 and those corresponding to the hyperplanes are labeled from 15 to 30. Thus, in Fig 3, the vertices corresponding to hyperplanes 0, 1, 4 are labeled as 15, 16 and 19 respectively. This subgraph is a complete bipartite graph. Now, if any two vertices corresponding to points and hyperplanes in this subgraph are considered, the four edges joining them form a cycle. One such cycle is shown in Fig. 3 by dotted edges. The portion of the edge-vertex incidence graph containing the variable nodes corresponding to these four dotted edges and their corresponding constraint nodes is shown in Fig. 4. The edge between nodes 0 and 15 is labeled as 0 and corresponds to variable node 0; edge between nodes 0 and 19 is labeled as 3 and corresponds to variable node 3; and so on. Observe that variable nodes 0, 3, 98 and 101 and their corresponding constraint nodes form a deadlock, where each constraint node is connected to at least two variable nodes. Thus, if symbols corresponding to these variable nodes are erased, it is not possible to correct them.

Using the above procedure, it is possible to find various locations of the code symbols that, if erased, cannot be corrected. It is important to emphasize that for $l = 2$, the cycle of length 4 does not exist in the base graph and the code is able to correct four erasures.

For erasures of size greater than four, the arguments similar to above can be built to arrive at the erasure patterns which cannot be corrected.

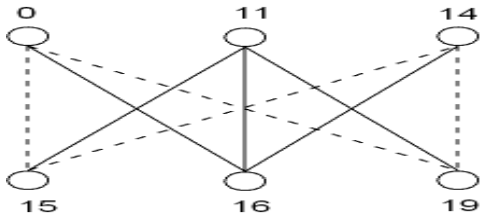

Fig. 3 Subgraph obtained from base line-hyperplane incidence relation.

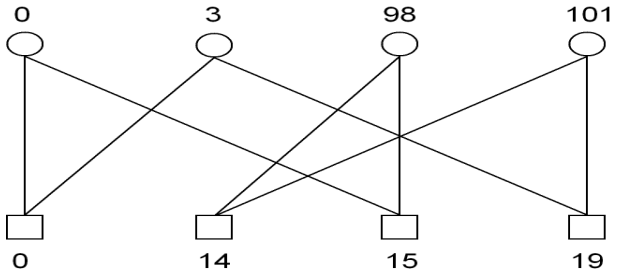

Fig. 4 Subgraph of edge-vertex incidence graph corresponding to Fig. 3.

4. Some Results and Discussion

Elaborate simulations are carried out to analyze the performance of the proposed codes in terms of their erasure correction capacity. Also studied are few issues relating to the floating point arithmetic. Some of these results are captured in this section, augmented with a discussion on different aspects and behavior of the codes.
The construction of the code, as discussed in Section 2, involves four steps. In the first step, a PG-based

Ramanujan graph G is constructed using the point-hyperplane incidence relation of $PG(l,2)$ (see Section 2.3). To simplify the implementation, the points corresponding to a particular hyperplane are first identified, and then, the remaining hyperplanes are obtained by applying the cyclic shift to this hyperplane. The second step consists of obtaining the edge-vertex incidence graph G' of the Ramanujan graph G. As seen earlier, the resultant graph is $(2,d)$-regular bipartite graph. The third step is to find the location of information symbols. The nodes in G' corresponding to the edges in the complement $\overline{T}$ of the spanning tree T of graph G give the locations of the information symbols. The requisite spanning tree of G is constructed using the Prim's algorithm, a well known algorithm in Computer Science (see for example, Chapter 6 of [34]). The last step is to find the parity symbols using the simple addition and subtraction operations such that the sum of the values at each of the constraint nodes is zero.

The effects of floating-point arithmetic are studied by fixing the number of digits to the right of decimal point. The values of information symbols are generated uniformly at random in the interval $[0,\ 1]$ and are truncated according to the pre-selected resolution. Note that by r-digit resolution, we mean that information symbols contain r digits to the right of the decimal point. For parity symbols, $r+\delta$ digits can be considered to the right of the decimal point to counter the effect of possible subtractions. For the given number of erasures, the erasure locations are selected at random. The decoding is computationally very simple and involves only additions and subtractions over real fields.

Some of the codes considered for simulation are given in Table 1 and their erasure correction capacity is shown in the Fig.5 for resolution of $r=8$. The guaranteed erasure correction performance claimed when number of erasures ≤ 3 is evident from the figure. Also, when number of erasures increases for a given dimension, it is clear that the performance shows graceful degradation, a useful feature indeed. Further, for a given number of erasures, the increase in the dimension increases the percentage of erasure correction, a "better" graceful degradation.

Now, let us consider the effect of resolution. Table 2 gives average of l_1-norm (called as residue) between actual and decoded codewords, following [6], for $r=8$. Note that ε denotes the number of erasures. It is easy to see that the mean residue increases with PG dimension l. This is because the degree of the bipartite graph d increases with l, and thus, calculations of parity symbols require more number of additions and/or subtractions. Also, the mean residue increases with the number of erasures, as more erasures require more addition/subtraction operations for recovering them.

Table 1: Specification of Codes obtained using various PG Dimensions

l	n	k	d	Rate
2	21	8	3	0.3810
3	105	76	7	0.7238
4	465	404	15	0.8688
5	1953	1828	31	0.9218
6	8001	7748	63	0.9684

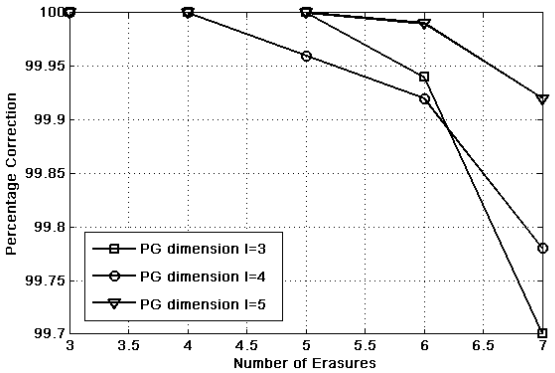

Fig. 5 Erasure Correction Capacity vs. Number of Erasures for different dimensions of PG.

Table 2: Mean Residue between Actual and Decoded Codewords for $r=8$.

l \ ε	3	4	5
3	9.89×10^{-17}	8.98×10^{-16}	3.65×10^{-15}
4	1.39×10^{-16}	1.28×10^{-15}	4.69×10^{-15}
5	2.03×10^{-16}	1.50×10^{-15}	6.68×10^{-15}
6	2.48×10^{-16}	1.88×10^{-15}	7.27×10^{-15}

It is to be noted that, compared to the codes of [7], for the proposed expander codes, the three erasure correction capability *is intact* even when n increases and is high. This is attributable to the frame work we have used: that of expander graphs. In this framework, the requisite computation for erasure correction is local to the constraint nodes of the graph, whether n is small or not. Thus, the ill-condition situation which manifests with

larger n in [7] is taken care of by the "divide-and-conquer" strategy intrinsic to the expander codes. Further, from the Table 1, it can be easily observed that the rate increases with the dimension of PG (and hence n). These aspects demonstrate that the proposed real expander codes are asymptotically good and efficient. Here, it may be worth mentioning about [35], where, some theoretical results are available towards using some Ramanujan Cayley graphs to arrive at asymptotically optimal class of cycle codes. But, the codes are over binary field and the explicit frame work of expanders is not considered in [35].

At this juncture, it is important to mention that the proposed codes do not follow the strict definition of expander codes. This is because, the classical definition of the expander codes requires that the degree d has to remain constant as number of nodes n increases. But, in our case, d increases as n increases (see Table 1); similar to [21] and [22]. Thus, the codes presented can be called as expander-like codes, following [21] and [22]. This can lead to better error-correction properties ([16],[17],[18],[21],[22]) and in fact, "better" graceful degradation with increase in l mentioned earlier can be attributed to this fact. But, there is a drawback with this increase in d from a practical point since the accumulator has to "grow" to take care of addition of more number of values for the parity calculation. Even though, the increase in d is rather slow compared to n (Table 1), this issue has to be appropriately addressed when the value of n is very large (like hundreds of a thousand in present day HPC systems).

Before concluding the paper, it is worth remarking that the real field codes proposed are arrived and examined from a "traditional" perspective. Since, the real codes are closely linked to the area of CS (see [13],[14]), as mentioned earlier, it can be worth investigating these codes within this framework. In that case, the erasure correction is related to sparsity of the signal to be detected and the parity check matrix is related to the so called "measurement matrix" ([13],[14]). The erasure correction or more generally the error correction is related to the Restricted Isometry Property (RIP) ([11],[12],[13],[14]) of the parity check matrix. Interestingly, in this perspective there is a possibility to arrive at novel deterministic measurement matrices and hence the associated reduced complexity decoding algorithms (see [14] and the references therein).

5. Conclusion

The paper presented novel real number expander-like codes based on projective-geometrically constructed Ramanujan graphs. The codes allow for fast linear time encoding and simple decoding in terms of erasure correction. Apart from guaranteeing three-erasure correction, each of the codes in the family shows graceful degradation of the performance for the increasing number of erasures. The codes can be explored for their utility in different scenarios of fault-tolerant computing and in other applications based on the Compressed Sensing paradigm.

References

[1] Jiun Shiu and Ja-Ling Wu, "Class of Majority Decodable Real-Number Codes", IEEE Trans. Communications, Vol.44, No.3, pp.281-283, March 1996.

[2] T. G. Marshall, Jr., "Coding of Real-Number Sequences for Error Correction: A Digital Signal Processing Problem", IEEE J. Select. Areas Commun, Vol.SAC-2, No.2, pp.381-391, March 1984.

[3] Zhengdao Wang and Georgios B.Giannakis,"Complex-Field Coding for OFDM Over Fading Wireless Channels", IEEE Trans. Inf. Theory, Vol.49, No.3, pp.707-720, March 2003.

[4] Z. Chen and J. Dongarra, "Numerically Stable Real Number Codes Based on Random Matrices", Proc. 5th International Conference on Computational Science (ICCS2005), Atlanta, USA, May 2005.

[5] Z. Chen and J. Dongarra, "Algorithm-Based Fault Tolerance for Fail-Stop Failures", IEEE Trans. Parallel and Distributed Systems, Vol.19, No.12, Dec. 2008.

[6] Z. Chen, G.E. Fagg, E. Gabriel, J. Langou, T. Angskun, G. Bosilca and J. Dongarra, "Fault Tolerant High Performance Computing by a Coding Approach", Proc. ACM SIGPLAN Symp. Principles and Practice of Parallel Programming, PPOPP, Chicago, USA, Jun. 2005.

[7] Zizhong Chen, "Optimal Real Number Codes for Fault Tolerant Matrix Operations", Proc. Conf. High Performance Computing Networking, Storage and Analysis, Oregon, USA, Nov. 2009.

[8] James S. Plank, Kai Li and Michael Puening, "Diskless Checkpointing", Technical Report UT-CS-97-380, Dept. Computer Science, University of Tennessee, Dec. 1997.

[9] Charng-da Lu, "Scalable Diskless Checkpointing for Large Parallel Systems", PhD Dissertation, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 2005

[10] Derek Vadala, "Managing RAID on LINUX", O' Reilly, 2003

[11] R. Baraniuk, "Compressive Sensing", Lecture Notes, IEEE Signal Processing Magazine, Vol.24, pp.118-124, July 2007.

[12] E. J. Candès and M. Wakin, "An Introduction to Compressive Sampling", IEEE Signal Processing Magazine, 25(2), pp. 21 - 30, March 2008.

[13] E. J. Candes and T. Tao, "Decoding by Linear Programming", IEEE Transactions on Information Theory, Vol.51, No.12, pp. 4203-4215, Dec.2005.

[14] B.S. Adiga, M. Girish Chandra and Shreeniwas Sapre, "Guaranteed Error Correction Based on Fourier Compressive Sensing and Projective Geometry", *Accepted*, ICASSP, Prague, Chech Republic, May 2011.

[15] S. Shinte, S. Katti, S. Jaggi, B.K. Dey, D. Katabi, and M. Medard, "Real and Complex Network Codes: Promises and

Challenges", IEEE Workshop on Network Coding and Applications, NetCod, Hong Kong. Jan.2008.
[16] Jose Miguel Perez Urquidi, "Expander Graphs and Error Correcting Codes", Master Thesis, Universite de Bordeaux, 2010.
[17] M. Sipser and D.A. Spielman, "Expander Codes", IEEE Trans. Inf. Theory, Vol.42, No.6, pp.1710-1722, Nov. 1996.
[18] G. Zemor, "On Expander Codes", IEEE Trans. Inf. Theory, Vol.47, No.2, pp.835-837, Feb. 2001.
[19] Amin Shokrollahi, "Expander Codes", Lecture Notes
[20] Tony De Souza, "Cycle Codes", Project Report, EPFL, June 2005.
[21] B.S. Adiga, S. Chaudhary, H. Sharma, S. Patkar, "System for Error Control Coding using Expander-like codes constructed from higher dimensional Projective Spaces, and their Applications", Indian Patent Application, 2455/MUM/2010.
[22] Hrishikesh Sharma, Swadesh Chaudhary, Sachin Patkar, "Subgraph Embeddings in Projective Space Lattices", Internal Technical Report, Dept. of Electrical Engineering, IIT Bombay, India, Nov 2009.
[23] Swadesh Choudhary, Tejas Hiremani, Hrishikesh Sharma and Sachin Patkar, "A Folding Strategy for DFGs derived from Projective Geometry based graphs", The 2010 International Congress on Computer Applications and Computational Science (CACS 2010), Singapore, Dec 4-6, 2010
[24] R. G. Gallager, "Low Density Parity Check Codes" , PhD Dissertation, MIT, 1963
[25] M. Girish Chandra, Harihara S.G, B.S. Adiga, Balamuralidhar. P, P.S. Subramanian, "Effect of Check Node Processing on the Performance of Message Passing Algorithm in the Context of LDPC Decoding for DVB-S2", *ICICS 2005*, Bangkok, Thailand, Dec.2005.
[26] M. A. Nielsen, "Introduction to Expander Graphs", June 2005
[27] D. Song, D.Zuckerman and D. Tygar, "Expander Graphs for Digital Stream Authentication and Robust Overlay Networks", Proc. IEEE Symposium on Security and Privacy (S&P02), California, May 2002.
[28] Nigel Hitchin, "Projective Geometry", Course Notes, 2003,
http://people.maths.ox.ac.uk/~hitchin/hitchinnotes/Projective_geometry/Chapter_1_Projective_geometry.pdf
[29] Jurgen Bierbrauer, "Finite Geometries", Course Notes, April, 2005
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.1808
[30] Stan Birchfield, "An Introduction to Projective Geometry (for computer vision), March 1998,
http://ai.stanford.edu/~birch/projective/projective.pdf
[31] http://en.wikipedia.org/wiki/Incidence_(geometry)
[32] V. Aravind, "Expander Codes", Lecture Notes,
www.cmi.ac.in/~ramprasad/lecturenotes/expcodes.pdf
[33] Prahladh Harsha, "Expander Codes", Course Notes, May, 2005,
www.tcs.tifr.res.in/~prahladh/teaching/05spring/lectures/lec6.pdf
[34] Gilles Brassard and Paul Brately, "Fundamentals of Algorithmics", Prentice Hall, 1996.
[35] J.P. Tillich and G. Zémor, "Optimal Cycle Codes Constructed from Ramanujan Graphs", SIAM Journal of Discrete Mathematics, Vol. 10, No. 3, p.447-459, August 1997.

B. S. Adiga obtained his BE (Electrical Engg.) and MTech (Industrial Electronics) degrees from Karnataka Regional Engineering College, Surathkal, India. He obtained his PhD in Computer Science from the Indian Institute of Science, Bangalore, India. He worked as a scientist at National Aerospace Laboratories, Bangalore, India, for nearly 20 years. Later on, he was with Motorola India Electronics Limited and Philips Innovation Labs, Bangalore, India. Presently, he is a Principal Scientist at the Innovation Labs, Tata Consultancy Services, Bangalore, India. His interests are in the broad areas of Signal Processing, Communications and Computing including, Error Control Coding, Compressive Sensing, High-Performance Computing and Multimedia Signal Processing.

M. Girish Chandra obtained his BE in Electronics from University Visvesvaraya College of Engineering, Bangalore and MTech from IIT Madras in Communication Systems and High Frequency Technology. He earned his PhD as a Commonwealth Scholar in Digital Communication from Imperial College, London. Presently, he is a Senior Scientist at the Innovation Labs, Tata Consultancy Services, Bangalore, India. Earlier, he was holding the position of Assistant Director at the Aerospace Electronics Division of National Aerospace Laboratories, Bangalore, India. His interests are in the broad areas of Communications and Signal Processing, including, Error Control Coding, Compressive Sensing, Cross-Layer Design and Multimedia Signal Processing.

Swanand Kadhe obtained his BE degree in Electronics and Telecommunication from University of Pune in 2007 and MTech degree from IIT Kanpur in Signal Processing, Communications and Networking in 2009. Currently, he is a researcher at the Innovation Labs, Tata Consultancy Services, Bangalore, India. His interests are in sparse graph-based codes, network coding, network information theory, cross layer design and role of feedback in wireless systems.