An Independent Component Analysis experiment using Distributed Computing Group Renaissance.

Daniel Beatty

2007-05-10

Abstract

1 Introduction

The original assignment is

> This project is to illustrate how a modified Independent Component Analysis (ICA) can be used to restore an image corrupted with additive Gaussian noise.
>
> Choose any natural image (of a reasonable size), add 30% Gaussian noise to the image, assuming a noisy image model:

$$\vec{z} = \vec{x} + \vec{n} \tag{1}$$

Denoise the noisy image by using the Sparse Code Shrinkage Method (closely related to ICA) as described in the following in the following reference.

Compare the restored image using the above method with standard noise removal filters such the simple median filter or the Wiener filter.

MATLAB codes for various versions of ICA are available from the home page of A Hyvärien and the Laboratory of Computer and Information Science at the University of Helsinki.

The best examples of MATLAB for this exercise is found in [**?**, 159-162]. The versions of ICA included are gradient ascent, and projection pursuit. ICA procedures determine both the mixing matrix and independent components from source data as shown in equation **??**. Contributing parts to each of the ICA algorithms include families of principal component analysis (PCA), entropy, estimators, and random variable properties. This report illustrates the implementation of this family frameworks for the Objective-C language. At the time of this report, the version of Objective-C used is version 1.x.

There does exist another framework that could be used in Objective-C in the form of the IT++ libraries which are provided free under the GNU Public License. The difference in the these two framework are as follows.

First, the Distributed Computing Groups (DCG) version uses the Renaissance libraries as the basis for its matrix, and the IT++ uses a special version of matrix representation. While there is nothing particularly wrong with either one, getting the data back out of DCG-Renaissance is much simpler as the data structure is a simple two dimensional double precision floating point which can be accessed directly.

Second, the renaissance frameworks closely mirrors the proposed CI-Renaissance frameworks, which are a set of GPGPU equivalent frameworks that can take advantage of the graphics card hardware when available. Consequently the implementation of ICA in the DCG framework allows for one class per object for each type of ICA. This allows the developer to use selectors to dynamically chose which ICA algorithm is desired.

In order show that satisfaction of ICA objects ensures satisfaction of Sparse Coding Shrinkage, this report first examines the contributing frameworks of ICA, in section **??**. Section **??** shows the connection between the composing algorithms and and the many forms of ICA. Section **??** shows the connection between ICA and Sparse Code Shrinkage.

2 Families of Components

There is a concept learned from the Core Image model [**?**] that proves to be useful in these cases. That is to design each of the components for ICA as specific purpose objects. Each object consists just the necessary components necessary to construct the results for the object's purpose. Each object has a message apply which instructs the object to construct its results based on its existing members. As such, there is a family of components contributing to ICA, in addition to the specific purpose ICA objects themselves. The frameworks containing these objects include:

- Renaissance itself (DCGRenaissance)
- Random Vectors and Tests for Independence (DCGRandom)

- Estimators (DCGEstimators)
- Entropy (DCGEntropy)
- Principle Component Analysis which is both part of DCGRenaissance and scheduled to expand into a new framework, DCGPrinciples.

In other literature, the specific purpose objects are called kernels. To distinguish this paper from operating system literature, the term kernel here is reserved for Core Image kernels. The family of components happen to be Objective-C frameworks.

2.1 DCGRenaissance

2.1.1 DCGMatrix

Consists of basic operations necessary a matrix in linear algebra. Subsequent operations of linear algebra are designed to be separate and work on DCGMatrix. A convenience object named blasWork is used combine each operation objects in a MATLAB like scheme. The operation objects such as add, subtract, and multiply so that the code for these operations is written and proven just once and therefore reducing error. There are also operation objects for vector-matrix interaction.

2.1.2 Gram-Schmidt Orthogonalization

One significant operation object handles the Gram Schmidt Orthogonalization. Theorem **??** states that an vector space $\mathbf{V}$ can be transformed in to another vector space $\mathbf{W}$ such that each $\vec{w} \in \mathbf{W}$ is orthogonal. Gram Schmidt Orthogonalization is a procedure based on theorem **??** that constructs each $\vec{w_i} \in \mathbf{W}$ on right after the other, and is defined in algorithm **??**.

Theorem 2.1. *Suppose $w_1, w_2, ..., w_n$ form an orthogonal set of non-zero vector in $\mathbf{V}$. Let v be any vector s.t. $v \in \mathbf{V}$. Define v' s.t.*

$$v' = v - c_1 w_1 - c_2 w_2 - ... - c_n w_n \tag{2}$$

$$c_i = \frac{\langle v, w_i \rangle}{||w_i||^2} \tag{3}$$

Then v' is orthogonal to $w_1, w_2, ..., w_n$.

[**?**, 211]

Algorithm 1 Gram Schmidt Orthogonalization

```
Require: Column Wise Source Matrix: $\mathcal{V}$
  $N$ is the number of column vectors in $\mathcal{V}$
  Base case $\vec{v_0} = \vec{w_0}$
  initialize $\vec{\theta}^0$, $T$, and $i \leftarrow 0$
  $v_0 \leftarrow$ [V columnVector:1]
  [W insertVector:$v_0$ atColumn:1]
  for j = 2 to N do
    $v_j \leftarrow$ [V columnVector:j]
    for i = 1 to j do
      $(w_i)_{norm} \leftarrow ||w_i||$
      innerProduct $(i_p) \leftarrow \langle w_i, v_j \rangle$
      $negsum$ += $i_p/(w_i)_{norm}$
    end for
    result = $v_j - negsum$
    [W insertVector:result atColumn:j]
  end for
  return $W$
```

2.1.3 Eigenvalues: Characteristic Polynomials

Definition 2.2. *Consider an n-squared matrix* $\mathbf{A}$ *over a field* $\mathcal{K}$

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ ... & ... & ... & ... \\ a_{n1} & a_{n2} & ... & a_{nn} \end{pmatrix}$$

The matrix $t\mathbf{I}_n - \mathbf{A}$ *where* $\mathbf{I}_n$ *is the n-square identity matrix and* t *is an inter-mediate, is called characteristic matrix of* $\mathbf{A}$

[**?**, 281]

The determinant of the characteristic matrix is the characteristic equation (polynomial) of $\mathbf{A}$. The definition for eigenvalues and eigenvectors is stated in the quoted definition **??**.

Definition 2.3. *Let* $\mathbf{A}$ *be an n-square matrix over a field* $\mathcal{K}$*. A scalar* $\lambda \in K$ *is called an eigenvalue of* $\mathbf{A}$ *if there exists a nonzero (column) vector* $\vec{v} \in \mathcal{K}^n$ *for which equation* **??** *holds.*

$$\mathbf{A}\vec{v} = \lambda\vec{v} \tag{4}$$

Every vector satisfying this relation is then called an eigenvector of $\mathbf{A}$ *belonging to the eigenvalue* λ*.*

[**?**, 284]

There are a family of algorithms that find the eigenvalues and eigenvectors called Eigenvalue Decomposition (EVD). EVD is part of the LAPACK libraries and consequently part of the Accelerate framework. Thus for CPU bound operations, the LAPACK version is the best choice as it is already optimized for the CPU.

2.1.4 Principle Components

> The principal components of the observed data are selected so that the ith principal component is the linear combination of the observed data that accounts for the ith largest portion of the variance in the observations. [**?**, 329]

Let $\vec{x}$ be a sample from the observed data whose space is denoted $\mathbf{X}$. Let $\vec{y}$ be the principal components of $\vec{x}$. Then there is a transformation matrix $\mathbf{P}$ such that equation **??** is satisfied.

$$\vec{y} = \mathbf{P}\vec{x} \tag{5}$$

In order for $\mathbf{P}$ to satisfy equation **??**, it should maximize equation **??**. This choice is made to satisfy the least squares error.

$$J(\mathbf{P}) = E\{\vec{y}^2\} \tag{6}$$

Most texts show that for each column vector of $\mathbf{P}$ have a derivation as shown in equation **??**. The important feature of $\mathbf{P}$ to be noted is that the eigenvectors of the covariance matrix of **X** ordered by matching eigenvalues.

$$J(\vec{p}_1) = E\{y_1^2\} = E\{(\vec{p}_1^T \vec{x})^2\} = \vec{p}_1^T E\{\vec{x}\vec{x}^T\}\vec{p}_1 \tag{7}$$

$$||\vec{p}_1|| = 1 \tag{8}$$

Note that any **P** that satisfies the constraints of PCA is a valid PCA transformation matrix. Some methods simply approximate **P**, and are not a collection of eigenvectors. These methods are often called on-line methods because they approximate **P** as each $\vec{x}$ becomes available. Other methods obey equation **??** by obtaining the eigenvectors, and some texts call these methods batch methods as they work on **X** as a collected whole. Singular Value Decomposition, Karhunen-Loeve, Hotelling transforms, and Pearson all acquire the transformation matrix that determines the principal components.

For CPU bound computation of PCA, SVD is a classic well known algorithm that is part of the LAPACK libraries and has acceleration via the Accelerate framework [**?**]. For GPU bound, both classic methods and online methods should be considered. Those considerations at this time are beyond the scope of this report.

In section **??**, PCA and ICA are explored in perspective how to apply them image processing, filters and masks. In the context of masks, it is more convenient to take the transformation matrix and use it in the filtering. In some filtering processes, it is better to the principal components, and the inverse mixing matrix.

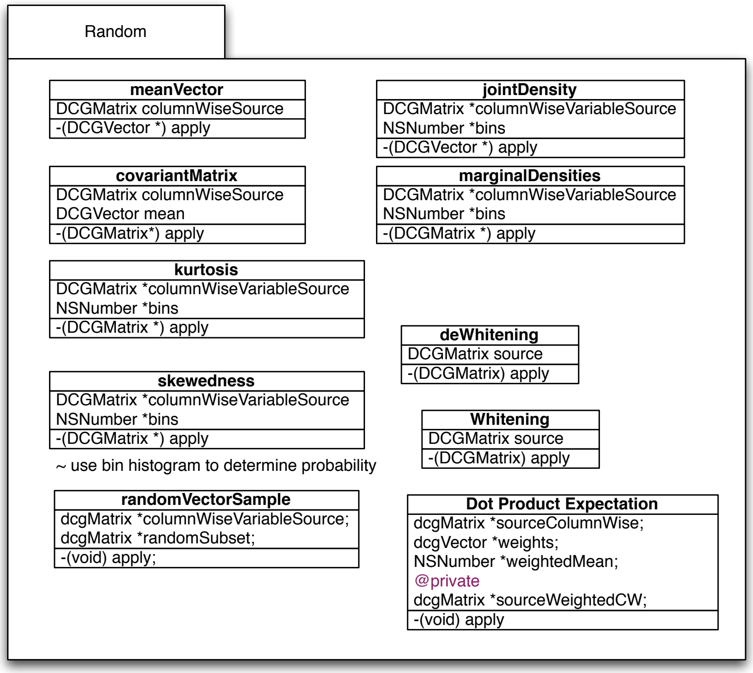


Figure 1: DCG Random Framework

2.2 DCGRandom

DCGRandom consists of property objects which correspond to general properties of random vectors. These property objects are defined here.

- Mean Vector

$$\vec{\mu} = E\{\vec{x}\} \tag{9}$$

- Covariance Matrix

$$\mathbf{\Sigma} = E\{(\vec{x} - \vec{\mu})^2\} \tag{10}$$

- Skewness

$$\mathbf{\Sigma} = E\{(\vec{x} - \vec{\mu})^3\} \tag{11}$$

- Kurtosis

$$\mathbf{\Sigma} = E\{(\vec{x})^4\} - (E\{\vec{x}^2\})^2 \tag{12}$$

- Whitening

$$A_w = \Lambda^{-1/2}\Phi \tag{13}$$

  where Λ are the eigenvalues of the covariance matrix and Φ are the normalized eigenvectors.

- Dewhitening

$$A_w^{-1} \tag{14}$$

One point about the operational objects defined in this framework is how they are reused. Projection pursuit, the primary method of ICA used in this report, computes ICA by maximizing the average entropy. Entropy is described in appendix **??**

3 ICA Family of Estimators

The ICA family of estimators are unique regarding the items being estimated. The standard ICA form is shown in equation **??**. The dataset denoted by $\vec{x}$ is the only observed part. The mixing matrix ($\mathbf{A}$,) the ICs ($\vec{s}$,) and the noise ($\vec{\nu}$) are all estimated. If the noise can be assumed to be normal, then it can be eliminated from the standard ICA form to give the standard noiseless ICA form, equation **??**. It is the inverse of the noiseless that allows for estimators of ICA to be constructed. Standard form of ICA w/o noise

$$\vec{x} = \mathbf{A}\vec{s} + n\vec{u} \tag{15}$$

$$\vec{x} = \mathbf{A}\vec{s} \tag{16}$$

$$\vec{s} = \mathbf{A}^{-1}\vec{x} \tag{17}$$

Construction of an ICA estimator from equation **??** requires a few algebraic manipulations. These manipulations are show in equations **??** through **??**.

$$y = \vec{b^T}\vec{x} \tag{18}$$

$$y = \vec{b^T}\mathbf{A}\vec{s} \tag{19}$$

$$\vec{q} = \mathbf{A}^T\vec{b} \tag{20}$$

$$y = \vec{q^T}\vec{s} \tag{21}$$

where

- $\mathbf{A}\vec{b}$ that maximizes y's non-normal characteristics.
- Such a $\vec{b}$ makes $\vec{q}$ have only one non-zero component.
- Therefore, y is an independent component.
- Determining the non-normal characteristics has to contend with $\pm s_i$ as IC(s) can be determined up to a sign.

- This determination becomes an angle of sorts.

One method for determining the non-normal characteristics is by kurtosis, shown in equation **??**. If y is whitened source of data, then equation **??** holds.

$$\kappa_4(y) = E\{y^4\} - 3(E\{y^2\})^2 \tag{22}$$

$$E\{y^2\} = 1 \Rightarrow \kappa_4(y) = E\{y^4\} - 3 \tag{23}$$

The measure of Non-normal characteristics can be accomplished by $|\kappa_4(\dot{})|$ and $\kappa_4(\dot{})^{2}$. Kurtosis has both additive and scaling properties as shown in equations **??** and **??**

$$\kappa_4(x_1) + \kappa_4(x_2) \equiv k_4(x_1 + x_2) \tag{24}$$

$$\kappa_4(\alpha x) = \alpha \kappa_4(x) \tag{25}$$

The objective is to apply kurtosis to y (or its definition as an IC element). Equation **??** shows the application of kurtosis to the characteristic whitened equation of ICA. A consequence of the data being whitened, shown in equation **??**, is a reduction of terms for the characteristic equation. From this consequence, the objective of finding each mixing vector translates to determining on the unit circle the maxima of $\kappa_4(y)$. In order to find the unit circle maxima, an assumption shown in equation **??**, the kurtosis of each IC is restricted to unit length. Therefore, gradient methods can be used to determine the kurtosis.

$$\kappa_4(y) = \sum_i q_i \kappa_4(s_i) \tag{26}$$

$$E\{y^2\} = 1 \Rightarrow \sum_i q_i^2 = 1 \tag{27}$$

$$\kappa_4(s_i) = 1 \Rightarrow F(\vec{q}) = \sum_i q_i^4 \tag{28}$$

3.1 Kurtosis Gradient Maximization

Let $\vec{z}$ be $\vec{x}$ whitened and $\vec{w}$ be a whitened version of $\vec{b}$. Then

$$y = \vec{w}^T \vec{z} \tag{29}$$

$$\kappa_4(y) = \sum_i w_i \kappa_4 z_i \tag{30}$$

$$\frac{\partial \kappa_4(y)}{\partial \vec{w}} = 4\text{sign}(\kappa_4(y))E\{\vec{z}(\vec{w}^T\vec{z})^3\} - 3\frac{\vec{w}}{||\vec{w}||} \tag{31}$$

$$\frac{\partial \kappa_4(y)}{\partial \vec{w}} \approx 4\text{sign}(\kappa_4(y))E\{\vec{z}(\vec{w}^T\vec{z})\} \tag{32}$$

$$\frac{\partial \kappa_4(y)}{\partial \vec{w}} \rightarrow 0 \tag{33}$$

Thus if the partial derivative is set to zero, then the goal is to determine the roots which are the maxima and minima. Note that $\vec{w}$ is to be normalized each iteration. Note also the expectation operator in many cases is treated as a classic mean of the argument inside.

Algorithm 2 Projection Pursuit

```
Require: data source $\vec{x}$
Require: Dot Product Expectation
Require: Random Vector producer
  Center data to make its mean zero
  Whiten the data to provide $\vec{z}$
  $\vec{w} \leftarrow$ random vector
  Provide initial value for $\gamma$
  repeat
    $y = E\{\vec{w}^T \mathbf{Z}\}$
    $\Delta\vec{w} = E\{\bar{y}^3 \mathbf{Z}\}$
    $\vec{w} += \Delta\vec{w}$
    $\vec{w} \leftarrow \frac{\vec{w}}{||\vec{w}||}$
  until $\Delta\vec{w} \rightarrow 0$
  return $\vec{w}$
```

3.2 Non-normality by Negentropy

One characteristic noted in [?, 94] is that normal variables have the largest entropy of all random variables. Appendix **??** shows the theory of neg-entropy and its use in maximizing entropy. Two equations from the theory of neg-entropy, equations **??** through **??**, can be used to approximate non-normality. If $\vec{y}$ is used in these equations, then the fact that $\vec{y}$ is zero mean and unit variance (whitened) leads this estimation to the same estimation as kurtosis as shown equation **??**.

$$J(y) \approx k_1(E\{G_1(y)\})^2 + k_2[(E\{G_2(y)\}) - (E\{G_2(v)\})]^2 \tag{34}$$

This is why the entropy estimation via non-polynomial functions is considered.

3.2.1 Negentropy Gradient Algorithm

An non-polynomial approximation is shown in equation **??**. Assuming that $G(\cdot)$ is defined as in G_1 or G_2, then applying constraints in equation **??** through equation **??** lead to algorithms **??** and **??**. Note that v in these equations is a standardized normal variable.

$$J(y) \approx [(E\{G(y)\}) - (E\{G(v)\})] \tag{35}$$

$$G_1(y) = \frac{1}{a_1} \log\cosh a_1 y \tag{36}$$

$$G_2(y) = -\exp(\frac{-y^2}{2}) \tag{37}$$

$$E\{(\vec{w}^T\vec{z})^2\} \to ||\vec{w}||^2 = 1 \tag{38}$$

$$\Delta\vec{w} \propto \gamma E\{\vec{z}(\vec{w}^T\vec{z})\} \tag{39}$$

$$\vec{w} = \frac{\vec{w}}{||\vec{w}||}\gamma = E\{G(\vec{w}^T\vec{z}) - E\{G(v)\}\} \tag{40}$$

Algorithm 3 FastICA Stochastic Negentropy Projection Pursuit

```
Require: data source $\vec{x}$
  Center data to make its mean zero
  Whiten the data to provide $\vec{z}$
  $\vec{w} \leftarrow$ random vector
  Provide initial value for $\gamma$
  repeat
    $\Delta\vec{w} \propto \vec{z}g(\vec{w}^T\vec{z})$
    $\vec{w} + = \Delta\vec{w}$
    $\vec{w} \leftarrow \frac{\vec{w}}{||\vec{w}||}$
    $\Delta\gamma \propto (G(\vec{w}^T\vec{z})E\{G(v)\}) - \gamma$
  until $\Delta\vec{w} \to 0$
  return $\mathbf{W}$
```

Algorithm 4 FastICA Negentropy Projection Pursuit

```
Require: data source $\vec{x}$
  Center data to make its mean zero
  Whiten the data to provide $\vec{z}$
  $\vec{w} \leftarrow$ random vector
  repeat
    $\vec{w}_{old} \leftarrow \vec{w}$
    $\vec{w} \leftarrow E\{\vec{z}g(\vec{w}^T\vec{z}) - E\{g'(\vec{w}^T\vec{z})\}\}$
    $\vec{w} \leftarrow \frac{\vec{w}}{||\vec{w}||}$
    $\Delta\vec{w} = \vec{w} - \vec{w}_{old}$
  until $\Delta\vec{w} \rightarrow 0$
  return W
```

3.3 Stability Analysis

Each version of projection pursuit hinges on two theorems to guarantee convergence. The solution of $J(y)$ is accomplished by an approximation of Newton's method to solve $J(\vec{w}^T\vec{z})$.

$$\vec{w} \leftarrow E\{\vec{z}(\vec{w}^T\vec{z})\} \tag{41}$$

$$\approx E\{\vec{z}g(\vec{w}^T\vec{z}) - E\{g'(\vec{w}^T\vec{z})\}\vec{w}^T\} \tag{42}$$

Theorem 3.1. *Assume that the input data follows the ICA model with whitened data*

$$\vec{z} = \mathbf{VA}\vec{s} \tag{43}$$

where $\mathbf{V}$ *is the whitening matrix, and that* G *is a sufficiently smooth even function. Then local maxima (resp. minima) of* $E\{G(\vec{w}^T\vec{z})\}$ *under the constraint* $||\vec{w}|| = 1$ *include those rows of the mixing matrix* $\mathbf{VA}$ *such that the correspond-*

ing independent components s_i satisfy

$$E\{s_i g(\vec{s_i}) - g'(\vec{s_i})\} > 0 \tag{44}$$

where $g(\dot{)} = G'(\dot{)}$, and $g'(\dot{)} = G''(\dot{)}$.

[**?**, 187]

Therefore any nonquadratic can be used for $G(\dot{)}$.

Theorem 3.2. *Assume that the input data follows the ICA data model in (8.1), and that G is a sufficiently smooth even function, then the asymptotically stable points of the algorithm in*

$$\Delta\vec{w} \propto \gamma E\{\vec{z} g(\vec{w}^T \vec{z})\} \tag{45}$$

$$\vec{w} = \frac{\vec{w}}{||\vec{w}||} \tag{46}$$

include the ith row of the inverse of the whitened mixing matrix **VA** *such that the corresponding independent components s_i satisfies*

$$g(\dot{)} = G'(\dot{)} \tag{47}$$

$$E\{s_i g(s_i) - g'(s_i)\}[E\{G(s_i)\} - E\{G(v)\}] > 0 \tag{48}$$

where v is a standardized normal variable.

[?, 187]

Algorithm 5 FastICA Deflationary Orthogonalization

```
Require: data source $\vec{x}$
Require: number of ICs to estimate $m$
  Center data to make its mean zero
  Whiten the data to provide $\vec{z}$
  for p = 1 to M do
    $\vec{w_p} \leftarrow$ random vector
    repeat
      $\vec{w_{old}} \leftarrow \vec{w_p}$
      $\vec{w_p} \leftarrow E\{\vec{z}g(\vec{w_p}^T\vec{z}) - E\{g'(\vec{w_p}^T\vec{z})\}\}$
      Make orthogonal (GSO step) $\vec{w_p} \leftarrow \vec{w_p} - \sum_{j=1} p - 1(\vec{w_p}^T\vec{w_j})\vec{w_j}$
      $\vec{w_p} \leftarrow \frac{\vec{w_p}}{||\vec{w_p}||}$
      $\Delta\vec{w_p} = \vec{w_p} - \vec{w_{old}}$
    until $\Delta\vec{w_p} \rightarrow 0$
  end for
  return W
```

4 Sparse Coding Shrinkage

Images also have characteristic equations, such as in equation **??**. The goal of Sparse Coding Shrinkage is to estimate a_i and s_i by forcing the spareness constraint. If the image values are treated as the $\vec{x}$ in ICA and a_i and s_i are their ICA equivalents, then ICA can estimate this characteristic equation.

$$I(x,y) = \sum_{i=1}^{n} a_i(x,y)s_i \tag{49}$$

Sparse Coding Shrinkage represents basis vectors such that only a small number of basis

Algorithm 6 FastICA Symmetric Orthogonalization

```
Require: data source $\vec{x}$
Require: number of ICs to estimate $m$
  Center data to make its mean zero
  Whiten the data to provide $\vec{z}$
  for p = 1 to M do
    $\vec{w_p} \leftarrow$ random vector
  end for
  repeat
    $\mathbf{W}_{old} = \mathbf{W}$
    for all $\vec{w_i} \in \mathbf{W}$ do
      $\vec{w_p} \leftarrow E\{\vec{z}g(\vec{w_p}^T\vec{z}) - E\{g'(\vec{w_p}^T\vec{z})\}\}$
    end for
    Apply GSO to $\mathbf{W}$
    $\Delta\mathbf{W} = |\mathbf{W}_{old} - \mathbf{W}|$
  until $\Delta\mathbf{W} \rightarrow \mathbf{0}$
  return $\mathbf{W}$
```

vectors are activated at the same time. [**?**, 397]. It is good for compression and de-noising. In compression, sparse coding shrinkage endeavors to find the rare samples and allow them to be coded special from other more mundane samples. In denoising, sparse coding shrinkage provides the independent components, and a selection of interest determines the "really active" components. It is the de-noising part that the interest of this report.

From all accounts of sparse coding shrinkage it is an application of ICA such that the local mean component is removed and the local variance is normalized. Furthermore, PCA is applied to the data. Sparse coding shrinkage does not specify that the procedure be restricted on how the data is arranged to obtain the mixing matrix.

[**?**, 391-400] used a neighborhood patch scheme, sampling $l \times l$ sections an arranging their values as one vector. The complete set of samples compose the source matrix, and the mixing matrix with their independent components may be estimated. Some experiments used a sliding neighborhood window as opposed to separating the matrix into patches.

4.1 Insert and Remove Noise on Patch Oriented Sparse Coding Shrinkage

Algorithm 7 Insert and Remove Noise on Patch Oriented Sparse Coding Shrinkage

```
Require: Image source $I$
  Establish patches
  Insert a random sample patches column wise into data source $\mathcal{X}_p$
  Estimate $\tilde{\mathbf{W}}$ for $\tilde{\mathbf{S}} = \tilde{\mathbf{W}}^T \mathcal{X}_p$
  Insert noise
  Construct $\hat{\mathbf{S}}$ using $\tilde{\mathbf{W}}$ s.t. $\hat{\mathbf{S}} = \tilde{\mathbf{W}}^T \mathcal{X}$
  Zero out whole columns of $\hat{\mathbf{S}}$ and construct $\hat{\mathbf{I}}$ with it.
  Show $I$, noisy $I$ and $\hat{\mathbf{I}}$.
```

Algorithm 8 Patch Oriented Source De-constructor

```
Require: Image source $I_m$
  $M$ is the number of rows in $I_m$
  $N$ is the number of columns in $I_m$
  $T$ is the total number of patches
  $M_p$ is the number of row patches. $M_p = \text{ceiling}(M/l)$
  $N_p$ is the number of column patches. $N_p = \text{ceiling}(N/l)$
  $T = M_p N_p$
  for each $\vec{h}_i \in \mathbf{H}$ do
    for $x \in L(\vec{h}_i)$ do
      for $y \in L(\vec{h}_i)$ do
        $\vec{h}_i(x\%l + y) = I_m(x, y)$
      end for
    end for
  end for
  return H
```

These algorithm give rise to the need of a framework called patch world. In actuality, patch world should consist of several smaller frameworks.. A patch framework is needed for

Algorithm 9 Patch Oriented Source Builder

```
Require: Patch Oriented Source $\mathbf{H}$
  for $x_p \in M_p$ do
    for $y_p \in N_p$ do
      Insert $\vec{h_i}$ into $I(x, y)$
    end for
  end for
  return $\mathbf{I_m}$
```

Algorithm 10 Patch Oriented Source Builder

```
Require: Image source $I$
  $M$ is the number of rows in I
  $N$ is the number of columns in I
  $T$ is the total number of patches
  $M_p$ is the number of row patches. $M_p = \text{ceiling}(M/l)$
  $N_p$ is the number of column patches. $N_p = \text{ceiling}(N/l)$
  $T = M_p N_p$
  for each $\vec{h_i} \in \mathbf{H}$ do
    for $x \in L(\vec{h_i})$ do
      for $y \in L(\vec{h_i})$ do
        $\vec{h_i}(x\%l + y) = I(x, y)$
      end for
    end for
  end for
  return $\mathbf{X}$
```

Algorithm 11 Insert and Remove Noise on Patch Oriented Sparse Coding Shrinkage

```
Require: Image source $I$
  Establish patches, $\mathbf{P}$
  Insert a random sample patches column wise into data source $\mathcal{X}_p$
  Estimate $\tilde{\mathbf{W}}$ for $\tilde{\mathbf{S}} = \tilde{\mathbf{W}}^T \mathcal{X}_p$
  Insert noise
  Construct $\hat{\mathbf{S}}$ using $\tilde{\mathbf{W}}$ s.t. $\hat{\mathbf{S}} = \tilde{\mathbf{W}}^T \mathbf{P}$
  Zero out whole rows of $\hat{\mathbf{S}}$ and construct $\hat{\mathbf{I}}$ with it.
  Show $I$, noisy $I$ and $\hat{\mathbf{I}}$.
```

neighborhoods, co-existing patches, column-wise neighbors, and others not thought of at the time of this document.

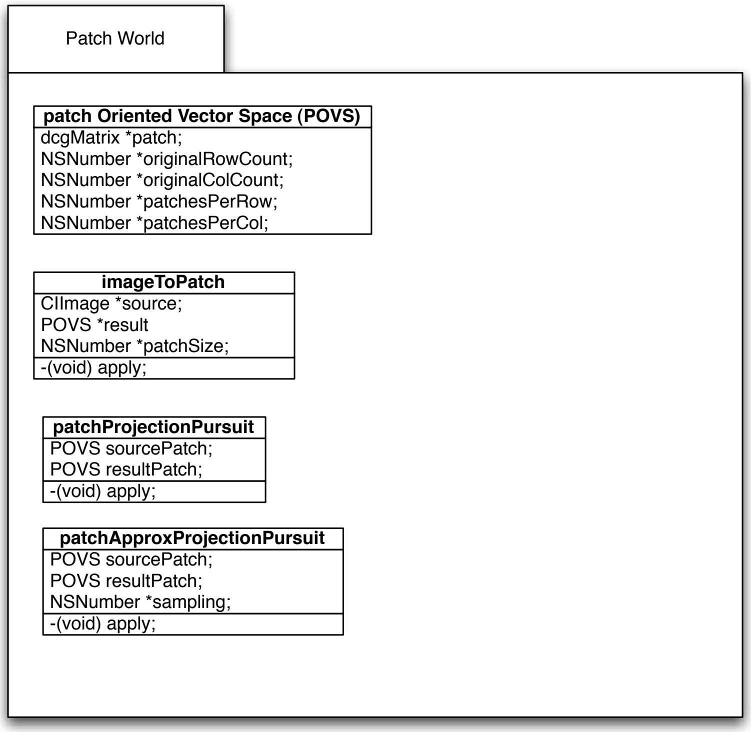


Figure 2: Patch World Framework

4.2 Connection between Independent Component Analysis and Sparse Code Shrinkage

The constraints on Sparse Code Shrinkage are included in the following list. Accompanying these requirements are comments that show the connection to the objects of ICA.

1. First, using a noise-free training set of $\vec{x}$, use some sparse coding method for determining the orthogonal matrix $\mathbf{W}$ so that the components s_i in $\vec{s} = \mathbf{W}\vec{x}$ have as sparse

distributions as possible. Estimate a density model $p_i(s_i)$ for each sparse component, using the models in **??** and **??**.

- Note that if $\vec{x}$ is already whitened, then the ICA mixing matrix is a transformation matrix satisfying the constraints on $\mathbf{W}$.
- The exponential and Laplace density functions are specific components to the non-polynomial projection pursuit family.

2. Invert the relation **??** to obtain estimates of the noise-free $\mathbf{x}$, given by $\hat{\mathbf{x}}(t) = \mathbf{W}^T\hat{\mathbf{s}}(t)$.

The matrices $\mathbf{W}$ and $\mathbf{A}$ are mixing matrices. Under the constraint that be both $\vec{s}$ and $\vec{x}$ are both zero mean and unit variant, $\mathbf{W} = \mathbf{A}^{-1} = \mathbf{A}^T$.

5 Filters using ICA and PCA

A Implementation of Projection Pursuit

Listing 1: Projection Pursuit in Objective C

```
-(void) apply
// Override on how the ICA algorithm is
// computed.  Namely, this one is a
// straight computation of projection
// pursuit.
{
```

```
[self prepareData];
int N = [samples colSize];


[randomAgent setLength:[NSNumber numberWithInt:N]];
[randomAgent apply];
dcgVector *w = [randomAgent randomVector];
dcgVector *wold;
dcgVector *deltaW;
double y;
Q = [NSNumber numberWithDouble:[w L2norm]];
do
 {
  lastQ = Q;
  wold = w;
  y = [self dotProductExpectation:w];
  [meanVector setSourceColumnWise:
   [blasWork multiplyMatrix:whitenedData
         byScalar:pow(y, 3.0)]];
         deltaW = [meanVector apply];


  w = [self addMixVector:w withDeltaVector:deltaW];
  Q = [NSNumber numberWithDouble:
                    [blasWork dotProduct:w
                    byVector:wold]];
 }
 while (![self epsilonReached]);
}
```

Listing 2: Zero Mean Multivariate in Objective C

```
-(void) apply
{
 dcgBlasService *blasWork;
 blasWork = [[dcgBlasService alloc]
    init];
 dcgMatrix *workingMatrix;
 workingMatrix = [dcgMatrix zeroMatrix:
  [sourceColumnWise rowSize]
  columns:[sourceColumnWise colSize]];
 int N = [sourceColumnWise colSize];
 int i;
 DCGMeanVector *meanVectorAgent;
 meanVectorAgent = [[DCGMeanVector alloc] init];
 [meanVectorAgent
```

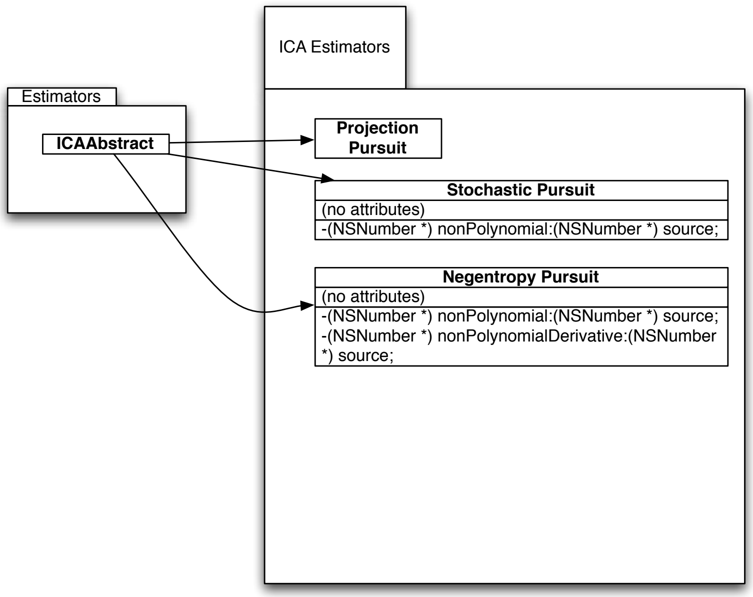
```
  setSourceColumnWise:sourceColumnWise];
 [self setMean:[meanVectorAgent apply]];
 for ( i = 0; i < N; i++)
 {
  [workingMatrix insertVector:mean atColumn:i];
 }
 [self setZeroMeanMultivariate:
    [blasWork subtractMatrix:sourceColumnWise
    bMatrix:workingMatrix]];
 [blasWork release];
 [meanVectorAgent release];
}
```

B Theory of Estimators

The theory of estimators is based on definition **??** and includes many linear and nonlinear methods to obtain the parameters thereof. Estimators that achieve estimated parameters close to the actual parameters are called constant.

Definition B.1. *An estimator, $\hat{\theta}$, of the parameter vector, $\vec{\theta}$, is the mathematical expression that estimates the parameters of an analytical solution for a given set of measurements (samples).*

There are a few constraints that any estimator should satisfy. Estimators for which



ICA Estimators
Estimators
ICAAbstract
Projection Pursuit
Stochastic Pursuit
(no attributes)
-(NSNumber *) nonPolynomial:(NSNumber *) source;
Negentropy Pursuit
(no attributes)
-(NSNumber *) nonPolynomial:(NSNumber *) source;
-(NSNumber *) nonPolynomialDerivative:(NSNumber *) source;

Figure 3: Estimators (ICA) Framework

equations **??** and **??** are called unbiased.

$$E\{\hat{\theta}\} \approx E\{\vec{\theta}\} \tag{50}$$

$$E\{\hat{\theta}|\vec{\theta}\} = \vec{\theta} \tag{51}$$

Estimators not meeting this criteria are called biased. The bias vector is defined

$$\vec{b} = E\{\tilde{\theta}\} \tag{52}$$

$$\vec{b} = E\{\tilde{\theta}|\vec{\theta}\} \tag{53}$$

If $\vec{b} \rightarrow \vec{0}$ as the number of measurements goes up, then the estimator is called asymmetrically unbiased. [**?**, 79]

Estimators often times require discrete items (called bins when forced on continuous values) in order to generate statistical models for datasets. One important concept is the loss function as defined in Definition **??**

Definition B.2. *A loss function $h(\tilde{\theta})$ is a metric of the relative importance of specific estimation errors.*

[**?**, 79]

Another important concept for estimators is their accuracy. The article efficient is used as a ranking of an estimator as defined in Definition **??**.

Definition B.3. *An efficient estimator provides the smallest error covariance matrix among all unbiased estimators.*

[**?**, 81] Symmetry is also a means of ordering a matrix. If $\mathbf{A}$ and $\mathbf{B}$ are both symmetrical matrices and $\mathbf{B} - \mathbf{A}$ is positive definite, then $\mathbf{A} < \mathbf{B}$. One theorem, Cramer-Rao Lower Bound, provides for partial ordering amongst symmetric matrices. This theorem is defined in theorem **??**

Theorem B.4. *If $\hat{\theta}$ is any unbiased estimator of $\vec{\theta}$ based on the measurement data $\vec{x}$, then the covariance matrix of error in the estimator is bounded below by* *the inverse of the Fisher information matrix:*

$$J^{-1} \leq E\{(\vec{\theta} - \hat{\theta})(\vec{\theta} - \hat{\theta})^T | \vec{\theta}\} \tag{54}$$

$$J = E\{[\frac{\partial}{\partial \vec{\theta}} \ln p(\vec{x_t}|\vec{\theta})][\frac{\partial}{\partial \vec{\theta}} \ln p(\vec{x_t}|\vec{\theta})]^T | \vec{\theta}\} \tag{55}$$

[**?**, 83]

The last quality of an estimator considered is robustness. Robustness is defined in Definition **??**. Robustness determines an estimators ability accurately model a dataset despite the presence of outlier errors.

Definition B.5. *Robustness is an insensitivity to gross measurement errors, and errors in specification of the parametric models.*

[**?**, 83]

Both maximum-likelihood (ML) and expected maximization (EM) are covered in another report. They may be reused in the pursuit of ICA, but in specific derivations thereof. As this report does not cover the ML version of ICA, this report only eludes to it as an alternative estimator used to determine ICA.

B.1 Least Squares

Least Squares regression is a linear method to determine specific parameters of an assumed analytical solution. It is a classic curve fitting method which can be built to be solved by matrix solution methods like Gauss Jordan Elimination. From this algorithm, there are a

Algorithm 12 Least Squares Estimation (Regression) derived from definition found at [?].

```
Require: DCGVector $\vec{x}$ consisting of $m$ observations
Require: Assumed model, parameters, and derived functions.
  Take an initial guess at the model parameters $\lambda_i \ \forall_i \in [1, n]$.
  repeat
    Form matrix $A$ such that
```

$$\begin{pmatrix} \frac{\partial f(x)}{\partial \lambda_1}|x_1 & \dots & \frac{\partial f(x)}{\partial \lambda_n}|x_1 \\ \dots & \dots & \dots \\ \frac{\partial f(x)}{\partial \lambda_1}|x_m & \dots & \frac{\partial f(x)}{\partial \lambda_n}|x_m \end{pmatrix} \tag{56}$$

```
    Compute $R = A^T A$
    Compute $dB$ such that
```

$$B = \sum_{i=1}^{m} (y_i - f(x)) \tag{57}$$

```
    Determine $L = A^T dB$
    Determine $\partial\vec{\lambda}$ from $R(\partial\vec{\lambda}) = L$
    Adjust $\vec{\lambda}$ by $\partial\vec{\lambda}$
  until $d\vec{\lambda} \to 0$
```

family of least squares estimators. Each one with their own assumed model and parameters. There is a general class for polynomials. Other models such as Gaussian or other statistical models should be developed with rules stated above for the univariate cases.

- The desired property is to have more measurements than parameters (known or unknown)
- The goal is to choose an estimator that minimizes the effects of the error.

B.2 Maximum a Posteriori

Maximum a Posteriori is based on Bayes Theorem, theorem **??**. The characteristic equation for a given set of data is the sum of a set of distributions. Each parameter of this characteristic equation can be determined by least squares regression. This principal comes from the Optimal Estimator Theorem.

$$p_{\vec{\theta}|\vec{x_t}}(\vec{\theta}|\vec{x_t}) = \frac{p_{\vec{x_T}|\vec{\theta}}(\vec{x_T}|\vec{\theta})p_{\vec{\theta}}(\vec{\theta})}{p_{\vec{x_T}}(x_T)} \tag{58}$$

$$p_{\vec{x_T}}(x_T) = \int_{-\infty}^{\infty} p_{\vec{x_T}|\vec{\theta}}(\vec{x_T}|\vec{\theta})p_{\vec{\theta}}(\vec{\theta})d\vec{\theta} \tag{59}$$

Theorem B.6. ***Optimal Estimator Theorem:*** *Assume that the parameters $\vec{\theta}$ and the observations $\vec{x_T}$ have the joint probability density function $p_{\vec{\theta},\vec{x_t}}(\vec{\theta},\vec{x_t})$. The minimum mean-square estimator $\hat{\theta}_{MSE}$ of $\vec{\theta}$ is given by the conditional expectation:*

$$\hat{\theta}_{MSE} = E[\vec{\theta}\vec{x_T}] \tag{60}$$

[**?**, 94]

The goal of Maximum a Posteriori to find the parameter vector $\vec{\theta}$ that maximizes the posterior density of the Bayes Theorem equation. The parameters are also based on an analytically defined model. In this case, the $\vec{\theta}$ needs to maximize the numerator, as the denominator does not depend on $\vec{\theta}$.

$$p_{\vec{\theta}|\vec{x_t}}(\vec{\theta}|\vec{x_t}) \approx p_{\vec{x_T}|\vec{\theta}}(\vec{x_T}|\vec{\theta})p_{\vec{\theta}}(\vec{\theta})d\vec{\theta} \tag{61}$$

$$\ln p_{\vec{\theta}|\vec{x_t}}(\vec{\theta}|\vec{x_t}) \approx \ln(p_{\vec{x_T}|\vec{\theta}}(\vec{x_T}|\vec{\theta})p_{\vec{\theta}}(\vec{\theta})d\vec{\theta}) \tag{62}$$

$$\approx \ln(p_{\vec{x_T}|\vec{\theta}}(\vec{x_T}|\vec{\theta})) + \ln(p_{\vec{\theta}}(\vec{\theta})d\vec{\theta}) \tag{63}$$

$$\approx \frac{\partial}{\partial\vec{\theta}}\ln(p_{\vec{x_T}|\vec{\theta}}(\vec{x_T}|\vec{\theta})) + \frac{\partial}{\partial\vec{\theta}}\ln(p_{\vec{\theta}}(\vec{\theta})d\vec{\theta}) \tag{64}$$

Given these equation the solution can be defined by setting equation **??** to zero.

$$\frac{\partial}{\partial\vec{\theta}}\ln(p_{\vec{x_T}|\vec{\theta}}(\vec{x_T}|\vec{\theta})) + \frac{\partial}{\partial\vec{\theta}}\ln(p_{\vec{\theta}}(\vec{\theta})d\vec{\theta}) = 0 \tag{65}$$

Like ML and EM, MAP is mentioned here as an alternative means of determining ICA. MAP itself does not provide ICs. Rather it provides an intermediate step from which ICA may be determined.

C Entropy and Mutual Information: Determining Independence

One of the most crucial features of the ICA estimator is the ability determine independence. Two fundamental concepts that lead to deterministic methods of independence are entropy and mutual information. As these two concepts are very much related, they are presented in this sub-section.

C.1 Measures of Information

The concept of entropy is best stated in its differential form, both discrete and continuous. Entropy can be treated as a measure of the amount of information contained in a specific data set. In the discrete case, this concept can determine encoding necessary to represent the data.

$$H(X) = \sum_i f(P(X = a_i)) \tag{66}$$

$$H(x) = -\int p_x(\eta) \log p_x(\eta) d\eta = \int f(p_x(\eta)) d\eta \tag{67}$$

Note that transforming entropy increases/decrease monotonically.

The metric of mutual information between scalar random variables, a measure of information that one random variable has on the other random variables in a set, is defined in terms of a single random variable x_i

$$I(\mathbf{X}, \mathbf{Y}) = H(\mathbf{Y}) - H(\mathbf{Y}|\mathbf{X}) = H(\mathbf{X}) - H(\mathbf{X}|\mathbf{Y}) \tag{68}$$

> Maximizing the joint entropy $H(Y_1, Y_2)$ is accomplished by maximizing the individual entropies while minimizing the mutual information $I(Y_1, Y_2)$. When $I(Y_1, Y_2)$ is zero, then Y_1 and Y_2 are statistically independent. [**?**, 662]

The purpose of Kullback-Leibler Divergence is provide a pseudo metric in terms of distance between two density functions, and therefore determine mutual information. It is not a true metric since it is not symmetric. Yet, the Kuller-Leibler Divergence does increase monotonically with mutual information.

$$\Delta(p|q) = \sum_i p(i)(\log p(i) - \log q(i)) \tag{69}$$

$$\Delta(p|q) = \int p(i)(\log p(i) - \log q(i)) \tag{70}$$

C.2 Theory of Maximizing Representation

The goal of maximum-entropy is to determine the probability density function that satisfies equation **??** such that the c_i constants are maximum. One example under some regularity conditions is equation **??** (under the constraint in equation **??**.

$$\int p(\eta)F_i(\eta)d\eta = c_i \tag{71}$$

$$p_0(\eta) = A\exp(\sum_i a_i F_i(\eta)) \tag{72}$$

$$\int p_0(\eta)d\eta = 1 \tag{73}$$

C.2.1 Negentropy

Negentropy is a concept derived from maximum entropy specifically for determining Gaussian properties. The objective is to have "a measure that is zero for a Gaussian variable and always nonnegative can be simply obtained from differential entropy." Negentropy, denoted $J(\vec{x})$, is defined in equation **??**. The entropy of x_{Gauss} is defined in terms the covariance matrix of $\vec{x}$ (Σ) and the dimension of $\vec{x}$.

$$J(\vec{x}) = H(\vec{x}_{Gauss}) - H(\vec{x}) \tag{74}$$

$$H(\vec{x}_{Gauss}) = \frac{1}{2}\log|\det\Sigma| + \frac{n}{2}[1 + \log 2\pi] \tag{75}$$

Properties of negentropy include:

- $J(\vec{x}) = 0$ i.f.f. $\vec{x}$ has a Gaussian distribution.
- The maximum Gaussian distribution satisfying equation **??** forces $J(\vec{x}) \geq 0$
- invariant to invertible linear transformation as proven in [**?**, 113].

C.2.2 Approximation by Cummulants

Polynomial density expansion approximation of negentropy, maximum entropy, and mutual information is defined for a standardized Gaussian model (zero mean and unit variance) as in equation **??**, and its derivative in equation **??**. The polynomial system is chosen for the practicality that they are orthogonal.

$$\phi(\eta) = \frac{1}{2\pi}\exp(\frac{-\eta}{2})\frac{\partial^i \phi(\eta)}{\partial \eta} = (-1)^i H_i(\eta)\phi(\eta) \tag{76}$$

Another approximation that maximize representation is the log of the cumulants. This approximation depends on the cumulants being relatively small.

$$\log(1+\epsilon) \approx \epsilon - \frac{\epsilon^2}{2} \tag{77}$$

Therefore, cumulants can be used to define a characteristic equation for the information carried in a dataset x.

$$p_x(\eta) \approx \phi(\eta)(1 + \kappa_3(x)\frac{H_3(\eta)}{3!} + \kappa_4(x)\frac{H_4(\eta)}{4!}) \tag{78}$$

$$J(x) \approx \frac{1}{12}\kappa_3(x)^2 + \frac{1}{48}\kappa_4(x)^2 \tag{79}$$

Thus a sum of the skewness squared and the kurtosis squared (both computable from the random framework.) The drawback is that this method is sensitive to outliers and it measures the tails and not the center. It is thus worth considering methods of determining entropy by non-polynomial functions. The constraints on such methods are as follows:

- Based on maximum entropy methods
- Confines the distribution from the infinite which will satisfy to a specific set of constraints.

These constraints allow for a first order approximation to be constructed for a one-dimensional random variable. GSO also for multiple-dimensional random variables to approximated by applying this process to each variable, and orthogonalizing the variables at the end of each iteration. An example for determining non-polynomial functions that achieve this are Gram-Charlier as shown in equation **??**.

$$J(x) \approx k_1(E\{G_1(x)\})^2 k_2(E\{G_2(x)\} - E\{G_2(\mathcal{V})\})^2 \tag{80}$$