# Improved fast Gauss transform
# User manual

VIKAS CHANDRAKANT RAYKAR, CHANGJIANG YANG, and RAMANI DURAISWAMI
Department of Computer Science, University of Maryland, CollegePark, MD 20783
{vikas,ramani}@cs.umd.edu

---



Categories and Subject Descriptors: [**IFGT User Manual**]: October 18, 2006

---

## 1. OKAY, WHAT IS THE GAUSS TRANSFORM?

In scientific computing literature the sum of multivariate Gaussian kernels is called as the *discrete Gauss transform.* More formally, for each *target point* $y_j \in \mathbf{R}^d$ the *discrete Gauss transform* is defined as,

$$G(y_j) = \sum_{i=1}^{N} q_i e^{-\|y_j - x_i\|^2/h^2} \quad \text{where,} \tag{1}$$

—$\{q_i \in \mathbf{R}\}_{i=1,\ldots,N}$ are the *source weights*,

—$\{x_i \in \mathbf{R}^d\}_{i=1,\ldots,N}$ are the *source points*, i.e., the center of the Gaussians,

—$\{y_j \in \mathbf{R}^d\}_{j=1,\ldots,M}$ are the *target points*,

—and $h \in \mathbf{R}^+$ is the *source scale* or *bandwidth.*

In other words $G(y_j)$ is the total contribution at $y_j$ of $N$ Gaussians centered at $x_i$ each with bandwidth $h$. Each Gaussian is weighted by the term $q_i$.

## 2. HOW RELEVANT IS THIS TO MACHINE LEARNING?

In most kernel based machine learning algorithms and non-parametric statistics the key computational task is to compute a linear combination of local kernel functions centered on the training data, i.e., $f(x) = \sum_{i=1}^{N} q_i k(x, x_i)$, which is the discrete Gauss transform for the Gaussian kernel. $f$ is the regression/classification function in case of regularized least squares, Gaussian process regression, support vector machines, kernel regression, and radial basis function neural networks. For non-parametric density estimation it is the kernel density estimate. Also many kernel methods like kernel principal component analysis and spectral clustering algorithms involve computing the eigen values of the Gram matrix. Training Gaussian process machines involves the solution of a linear system of equations. Solutions to such problems can be obtained using iterative methods, where the dominant computation is evaluation of $f(x)$.

## 3. WHAT IS THE IMPROVED FAST GAUSS TRANSFORM?

The computational complexity to evaluate the discrete Gauss transform (Equation 1) at $M$ target points is $\mathcal{O}(MN)$. This makes the computation for large scale problems prohibitively expensive. The *improved Fast Gauss Transform* (IFGT) is an $\epsilon - exact$ approximation algorithm that reduces the computational complexity to $\mathcal{O}(M+N)$, at the expense of reduced precision, which however can be arbitrary. The constant depends on the desired precision, dimensionality of the problem, and the bandwidth. Given any $\epsilon > 0$, it computes an approximation $\hat{G}(y_j)$ to $G(y_j)$ such that the maximum absolute error relative to the total weight $Q = \sum_{i=1}^{N} |q_i|$ is upper bounded by $\epsilon$, i.e.,

$$\max_{y_j} \left[ \frac{|\hat{G}(y_j) - G(y_j)|}{Q} \right] \leq \epsilon. \tag{2}$$

## 4. WHY IMPROVED?

The IFGT [Raykar et al. 2005; Yang et al. 2005] is an improved version of the *Fast Gauss Transform* (FGT) which belongs to the more general fast multipole methods [Greengard and Rokhlin 1987]. The FGT was first proposed in [Greengard and Strain 1991] and applied successfully to a few lower dimensional applications in mathematics and physics. However the algorithm has not been widely used much in statistics, pattern recognition, and machine learning applications where higher dimensions occur commonly. An important reason for the lack of use of the algorithm in these areas is that the performance of the proposed FGT degrades exponentially with increasing dimensionality. The FGT is practical upto three dimensions. For the FGT the constant factor in $\mathcal{O}(M+N)$ grows exponentially with increasing dimensionality $d$. For the IFGT the constant factor is reduced to asymptotically polynomial order. The reduction is based on a new multivariate Taylor's series expansion scheme combined with the efficient space subdivision using the $k$-center algorithm.

## 5. CAN YOU BRIEFLY DESCRIBE THE ALGORITHM?

The detailed complete description of the algorithm can be found in our technical report [Raykar et al. 2005]. The algorithm has four stages.

(1) Determine parameters of algorithm based on specified error bound, kernel bandwidth, and data distribution
(2) Subdivide the $d$-dimensional space using a $k$-center clustering based geometric data structure.
(3) Build a $p$ truncated representation of kernels inside each cluster using a set of decaying basis functions.
(4) Collect the influence of all the the data in a neighborhood using coefficients at cluster center and evaluate.

## 6. HOW DIFFERENT IS IT FROM THE NIPS 2005 PAPER YOU HAD?

The core IFGT algorithm was first presented in [Yang et al. 2005]. The error bound proposed in the original paper was not tight to be useful in practice. Also the paper

did not suggest any strategy for choosing the parameters to achieve the desired bound. Users found the selection of parameters hard. Incorrect choice of algorithm parameters by some authors sometimes lead to poor reported performance of IFGT. This version provides a method for automatically choosing the parameters.

## 7. ALL RIGHT, TELL ME HOW TO USE THE CODE?

The IFGT consists of three phases: parameter selection, space subdivision using farthest point clustering, and the core IFGT algorithm. Refer to example.m in the distribution.

---

**Gather the input data and the following parameters**

---

```
% the data dimensionality
d=2;
-----------------------------
% the number of sources
N=1000;
-----------------------------
% the number of targets
M=1000;
-----------------------------
% d x N matrix of N source points in d dimensions.
X=rand(d,N);
-----------------------------
% d x M matrix of M source points in d dimensions.
Y=rand(d,M);
-----------------------------
% the bandwidth
h=0.7;
-----------------------------
IF YOU HAVE NOT SCALED THE DATA TO LIE IN A UNIT HYPERCUBE DO IT NOW.
REMEMBER TO SCALE THE BANDWIDTH ALSO.
-----------------------------
% the desired error. For many tasks I found the error in the range
% 1e-3 to 1e-6 to be sufficient. However epsil can be arbitrary.
epsil=1e-3;
-----------------------------
% The upper limit on the number of clusters. If the number of clusters
% chosen is equal to Klimit then increase this value.
Klimit=round(0.2*sqrt(d)*100/h);
-----------------------------------------------------------------------
```

```
-------------------------------------------------------------------
Step 1 Choose the parameters required by the IFGT
% K           -- number of clusters
% p_max       -- maximum truncation number
% r           -- cutoff radius
[K,p_max,r]=ImprovedFastGaussTransformChooseParameters(d,h,epsil,Klimit);

-------------------------------------------------------------------
Step 2 Run the k-center clustering
% rx               -- maximum radius of the clusters
% ClusterIndex     -- ClusterIndex[i] varies between 0 to K-1.
% ClusterCenters   -- d x K matrix of K  cluster centers
% NumPoints        -- number of points in each cluster
% ClusterRadii     -- radius of each cluster
[rx,ClusterIndex,ClusterCenter,NumPoints,ClusterRadii]=...
KCenterClustering(d,N,X,double(K));
-------------------------------------------------------------------
Step 3 Update the truncation number
% Initially the truncation number was chosen based on an estimate
% of the maximum cluster radius. But now since we have already run
% the clustering algorithm we know the actual maximum cluster radius.
[p_max]=ImprovedFastGaussTransformChooseTruncationNumber(d,h,epsil,rx);

-------------------------------------------------------------------
Step 4 Compute the IFGT
% G_IFGT -- 1 x M vector of the Gauss Transform evaluated.
[G_IFGT]=ImprovedFastGaussTransform(d,N,M,X,h,q,Y,double(p_max),...
double(K),ClusterIndex,ClusterCenter,ClusterRadii,r,epsil);
-------------------------------------------------------------------
Direct Computation
[G_direct]=GaussTransform(d,N,M,X,h,q,Y);
IFGT_err=max(abs((G_direct-G_IFGT)))/sum(q);
-------------------------------------------------------------------
```

## 8. WHY DON'T YOU GIVE ME A SINGLE DLL?

I have provided separate programs for parameter selection, space subdivision, and the core IFGT algorithm. This is because when using the IFGT multiple times with different $q$ the parameter selection and k-center clustering have to be done only once. Also anyone interested can try different space sub-division schemes.

There is also IFGT.m–a wrapper function which combines all the above. Just provide the accuracy $\epsilon$.

## 9. WHAT IS THIS DATA ADAPTIVE VERSION?

Another novel idea is that a different truncation number can be chosen for each data point depending on its distance from the cluster center. A good consequence of this strategy is that only a few points at the boundary of the clusters have high truncation numbers. Theoretically we expect to get a much better speed up

since for many points $p_i < p_{max}$. However some computation resources are used in determining the truncation numbers based on the distribution of the data points. As a result this scheme sometimes gives only a slight improvement especially when there is structure in the data. The actual error is much closer to the target error.

```
-------------------------------------------------------------------------
% G_DAIFGT -- 1 x M vector of the Gauss Transform evaluated.
% T         -- 1 x N vector  truncation number used for each source.
-------------------------------------------------------------------------
[G_DAIFGT,T]=DataAdaptiveImprovedFastGaussTransform(d,N,M,X,h,q,Y,...
double(p_max),double(K),ClusterIndex,ClusterCenter,ClusterRadii,r,epsil);
-------------------------------------------------------------------------
```

## 10. HOW GOOD ARE THE PARAMETERS CHOSEN?

We have tried to make the error bounds as tight as possible. However it is still a bound and the actual error is much less than the target error, typically 3 magnitudes lower than the target error. So my suggestion is to go easy on the target error. Say if you want can error of $10^{-6}$ you can safely set $\epsilon = 10^{-3}$. A good test to to try it out on a smaller sample size so that you can check the actual error with the target error. Empirically we also observed that the error does not vary significantly with the size of the dataset. Also if you wish you can experiment with different $K$ and $p$.

## 11. CAN YOU HANDLE VERY SMALL BANDWIDTHS?

For data scaled to a unit hypercube I got good speedups when the bandwidths scaled as $h = c\sqrt{d}$ (note that $\sqrt{d}$ is the length of the maximum diagonal of a unit hypercube) for some constant $c$. For very small kernel bandwidth where each training point only influences the immediate vicinity speedup is poor. More importantly the cutoff point (i.e., when the IFGT takes less time than the direct method) increases, to be useful for moderately sized datasets. It is not advisable to use IFGT when the truncation number $p_{max}$ is very high. For very small bandwidths you can use the dual-tree methods [Gray and Moore 2003] which rely only on data structures and not on series expansions. The dual-tree methods give good speedup for small bandwidths while the series based methods such as IFGT give good speedup for large bandwidths. For many machine learning tasks in large dimensions the bandwidth has to be moderately large to get good generalization. See Table I for a recommendation of which method to use.

## 12. WHAT ABOUT THE SPARSE DATA-SET REPRESENTATION METHODS?

There are many strategies for specific problems which try to reduce this computational complexity by searching for a sparse representation of the data. All these try to find a reduced subset of the original data-set using either random selection or greedy approximation. In these methods there is no guarantee on the approximation of the kernel matrix in a deterministic sense. The IFGT uses all the available data. Also the IFGT can be combined with these methods to obtain much better speedups.

Table I. Summary of the better performing algorithms for different settings of dimensionality $d$ and bandwidth $h$ (assuming data is scaled to a unit hypercube). The bandwidth ranges are approximate.

| | Small dimensions $d \leq 3$ | Moderate dimensions $3 < d < 10$ | Large dimensions $d \geq 10$ |
|---|---|---|---|
| Small bandwidth $h \leq\approx 0.1$ | Dual tree [$kd$-tree] | Dual tree [$kd$-tree] | Dual tree [Anchors] |
| Moderate bandwidth $0.1 \leq\approx h \leq\approx 0.5\sqrt{d}$ | FGT, IFGT | IFGT | |
| Large bandwidth $h \geq\approx 0.5\sqrt{d}$ | FGT, IFGT | IFGT | IFGT |

## 13. SO, WHAT NEXT?

So scale your data, use the code, mail me if you have any questions, brood over things, and remember to have fun.

## REFERENCES

Gray, A. G. and Moore, A. W. 2003. Nonparametric density estimation: Toward computational tractability. In *SIAM International conference on Data Mining*. 5

Greengard, L. and Rokhlin, V. 1987. A fast algorithm for particle simulations. *J. of Comp. Physics 73,* 2, 325–348. 2

Greengard, L. and Strain, J. 1991. The fast Gauss transform. *SIAM Journal of Scientific and Statistical Computing 12,* 1, 79–94. 2

Raykar, V. C., Yang, C., Duraiswami, R., and Gumerov, N. 2005. Fast computation of sums of Gaussians in high dimensions. Tech. Rep. CS-TR-4767, Department of Computer Science, University of Maryland, CollegePark. 2

Yang, C., Duraiswami, R., and Davis, L. 2005. Efficient kernel machines using the improved fast Gauss transform. In *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, and L. Bottou, Eds. MIT Press, Cambridge, MA, 1561–1568. 2