k Dimensional Statistics

a general purpose package for multidimensional statistical analysis

Marian Ivanov
Alexandru Bercuci

Alice offline week
CERN 10.10.2007

Overview

- Binary search tree (kd tree)
- k nearest neighbors search using kd trees
- PDF estimation for n dimensional data sets

Motivation

- Fast navigation in multidimensional data sets
  - Tracking, finding closest clusters
  - Event mixing – finding closest events
- Reference data (TRD PID, possibly for calibration data)

kDTree Build

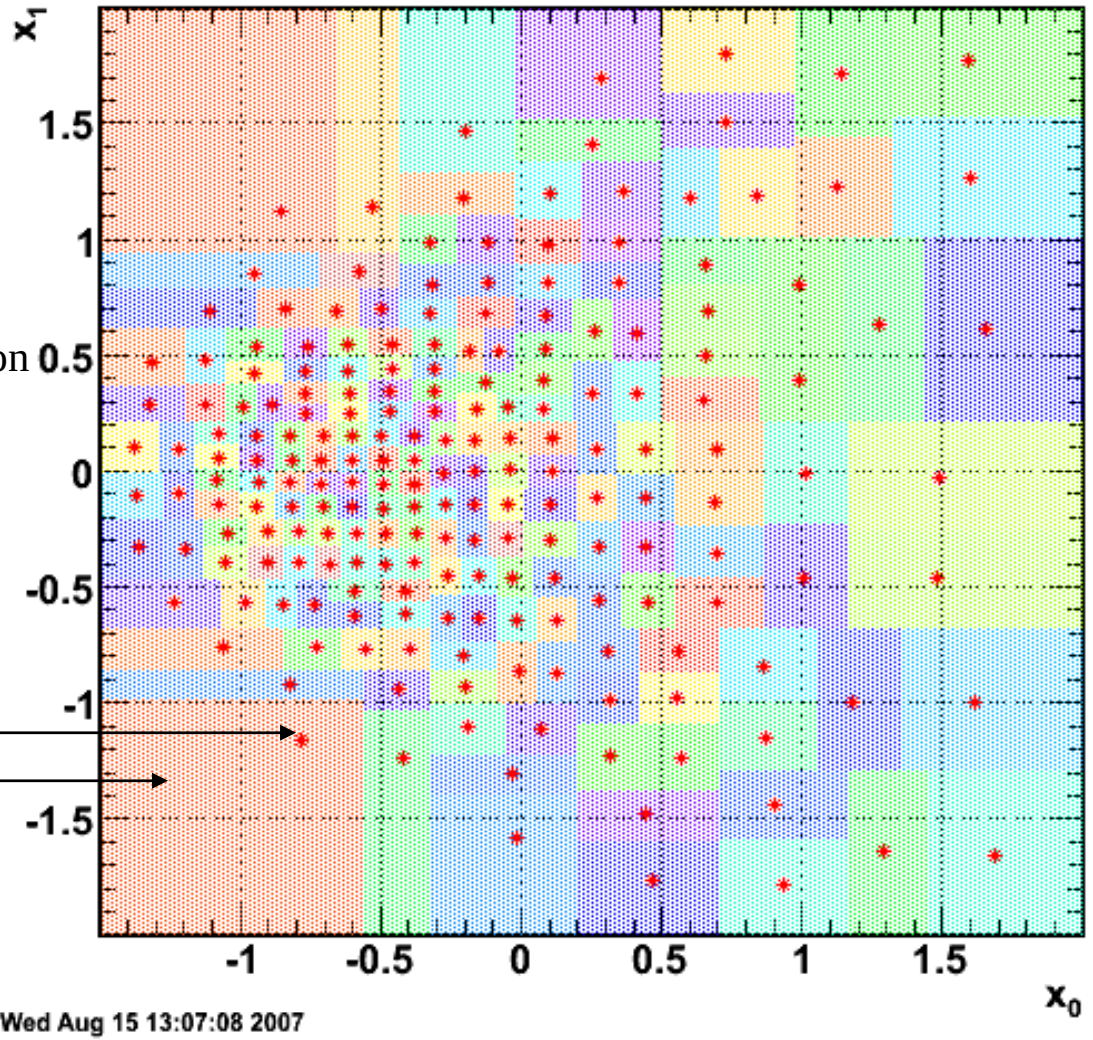
- stand alone kdTree implementation
- lightweight in terms of memory consumption

```
struct TKDNode{
  UChar_t fAxis;
  Value   fValue;
  ClassDef(TKDNode, 1)
} *fNodes;
```

- Fast data navigation

```
Index   *fIndPoints;
```

- TKDInterpolator adds:
  - user friendiness
  - visualization
  - COG evaluation



API

TKDTree<Int_t, Float_t>(Int_t npoints, Int_t ndim, UInt_t bsize, Float_t **data)

TKDInterpolator(TTree *t, const Char_t *var, const Char_t *cut, UInt_t bsize)

TKDInterpolator::DrawNodes(UInt_t ax1, UInt_t ax2, Int_t depth)

kDTree costs

Memory consumption

Option 1: TKDNode

struct TKDNode{
UChar_t fAxis; 1 byte (3 padding bytes for alignment)
Value fValue; 4 bytes
ClassDef(...) (4/8 bytes for fgIsA pointer)
}
Total 12/16 bytes (32/64 bits machine)

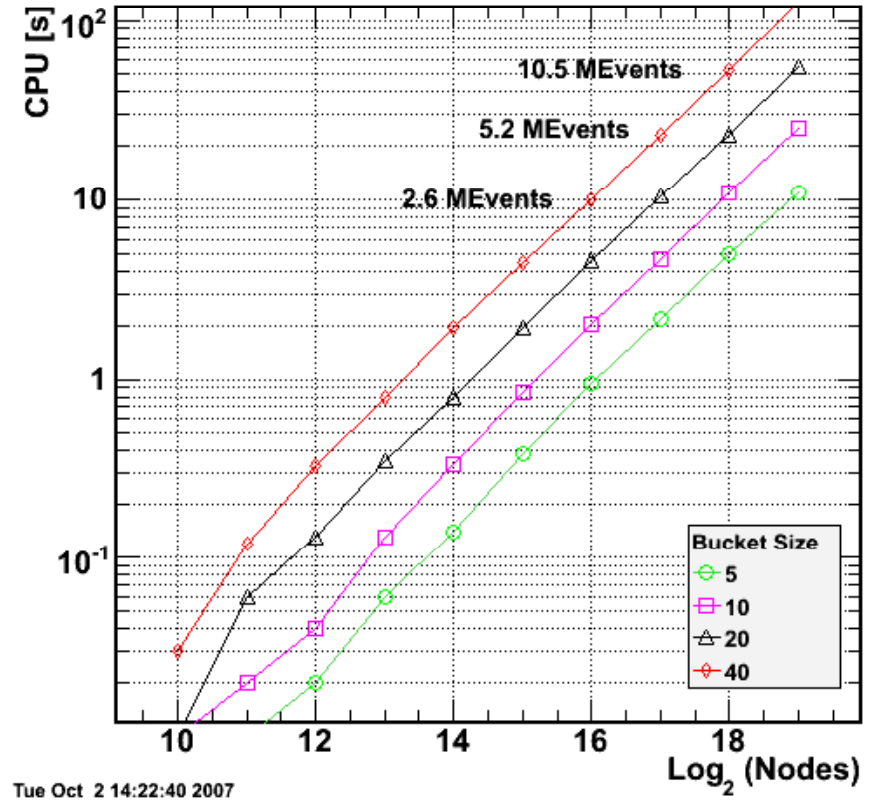
Option 2: Primitive data arrays

*UChar_t *fAxis;*
*Value *fValue;*

Real case: *1M points, 100 bs, 10K nodes*

Option 1 : 117.3 KB
Option 2 : 48.9KB

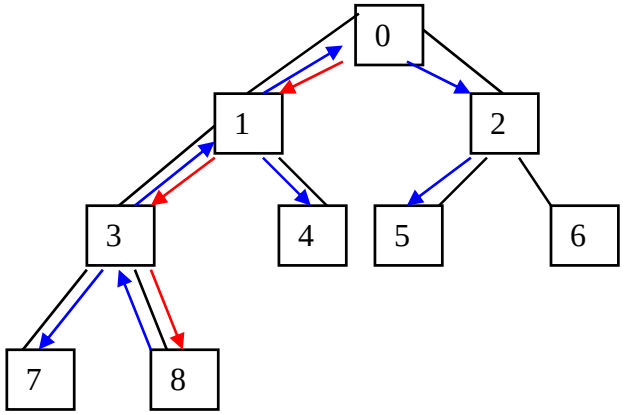
Speed (D = 2)

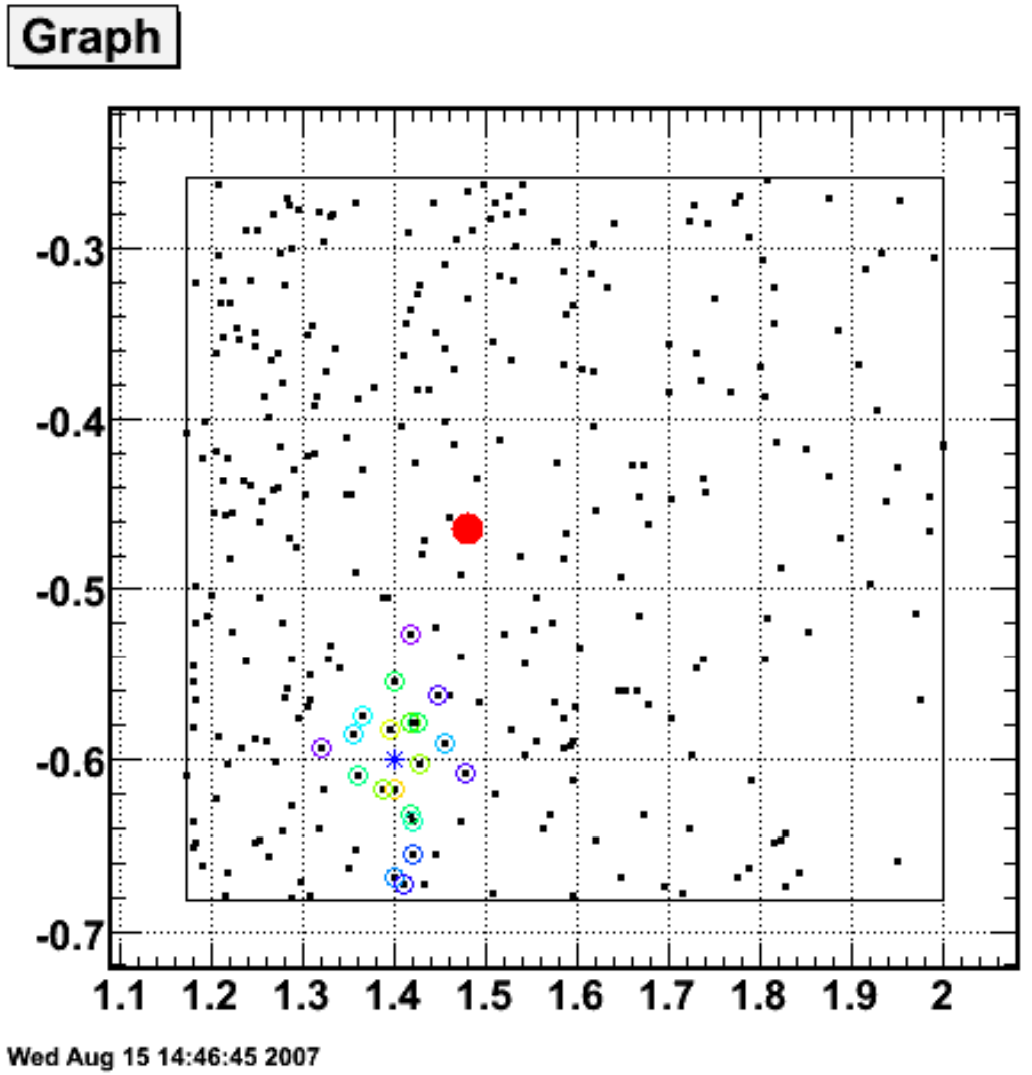


*) Intel(R) Celeron(R) CPU 2.66GHz, 1GB RAM

kNN

(k Nearest Neighbors)



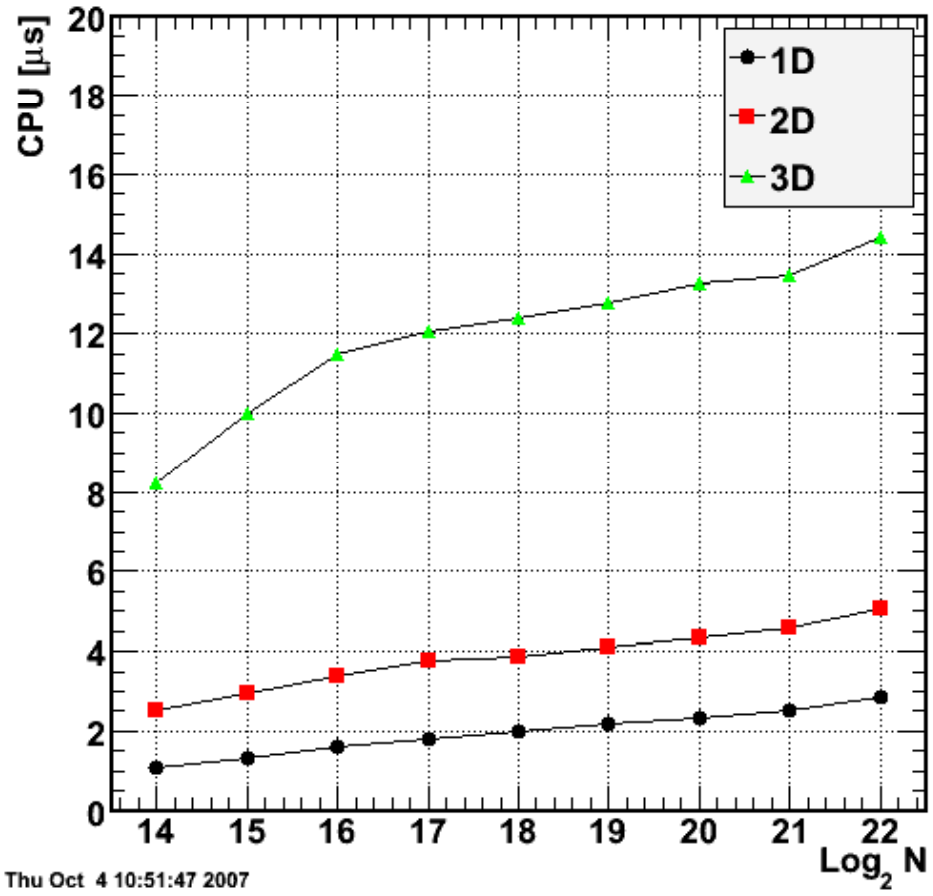


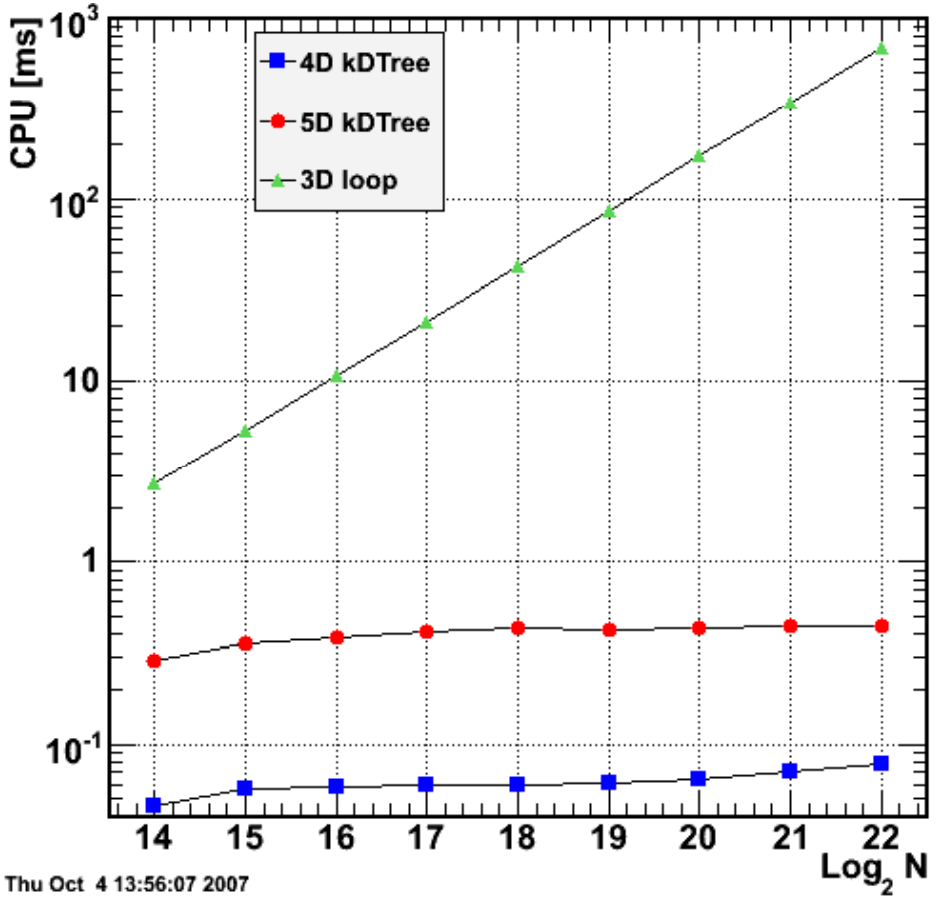
API

TKDTree<Int_t, Float_t>::FindNearestNeighbors(const Float_t *p, const Int_t kNN, Int_t *&i, Value &d)

TKDInterpolator::DrawNode(UInt_t tnode, UInt_t ax1, UInt_t ax2)

kNN speed





*) Intel(R) Celeron(R) CPU 2.66GHz, 1GB RAM

kD Interpolation - LOESS

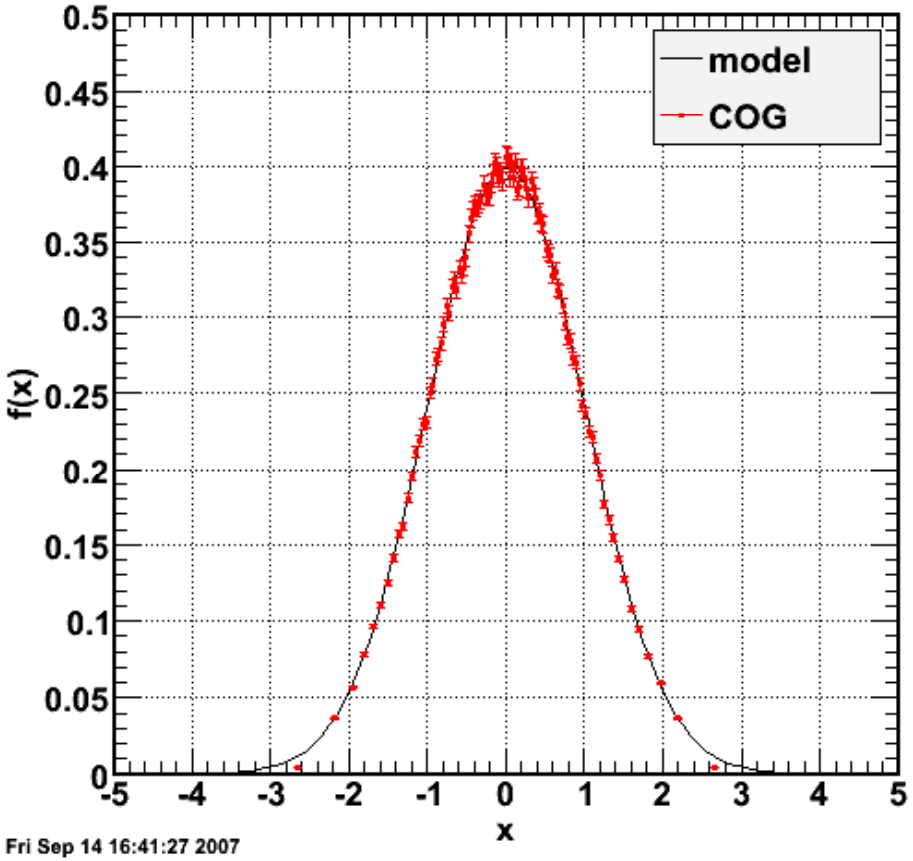
Locally Weighted Polynomial Regression *aka* LOESS

- Each point is evaluated by low-degree polynomial fit to *a subset of the data*, with *explanatory variable values near the point* whose response is being estimated.
- data set = {x, $f(x)$, $\sigma(x)$}
- statistics
- bucket size, smoothing (α)

$f=N_{bs}/S$

$\sigma=f/N_{bs}^{1/2}$

$N_{NN}=(1+\alpha)*N_{param}$



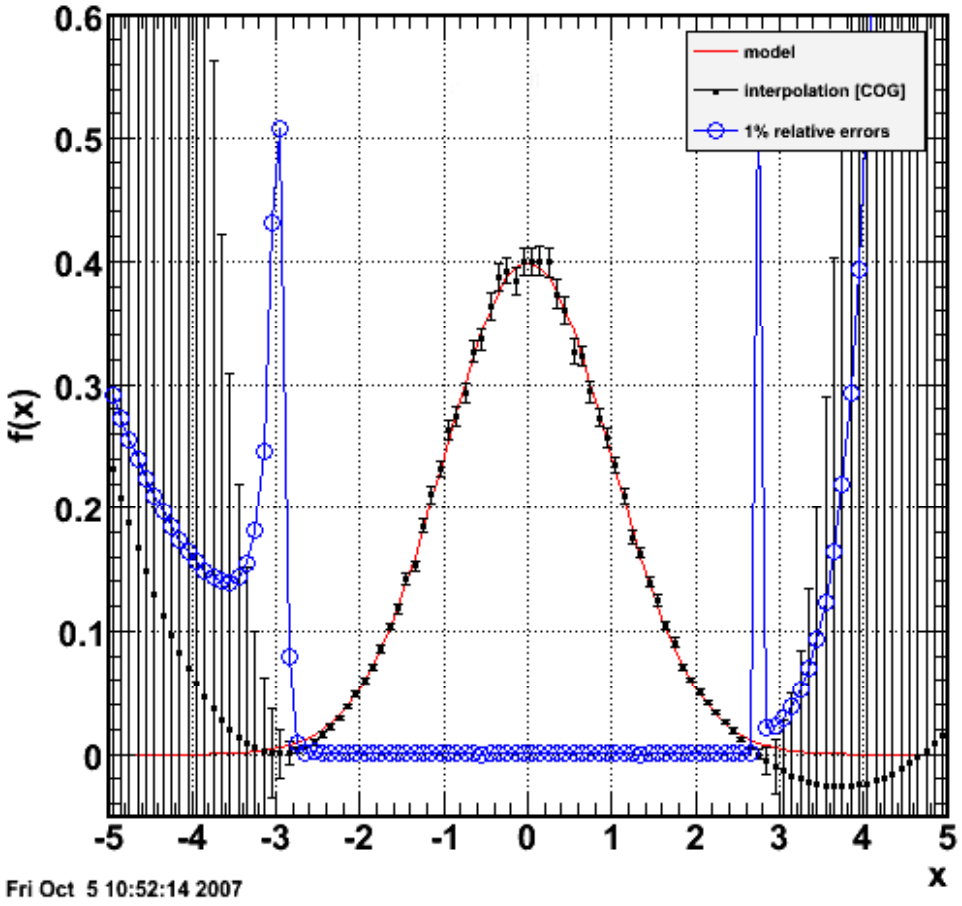
Interpolation Method 1

- - The polynomial is fit using *weighted least squares*, giving more weight to points near the point whose response is being estimated and less weight to points further away
- - The traditional weight function used for LOESS is the *tri-cube weight function*

$$\omega(x_i) = (1 - d^3)^3; \quad d = D(X, x_i) / D_{\max}$$

$$f(X) = P_2(X, \vec{p});$$

$$\sigma(X) = \sum\sum \frac{\partial f}{\partial p_i} \frac{\partial f}{\partial p_j} Cov(p_i, p_j)$$



API

Double_t TKDInterpolator::Eval(const Double_t *point, Int_t npoints, Float_t &result, Float_t &error)
void TKDInterpolator::SetInterpolationMethode(const Bool_t)
void TKDInterpolator::SetAlpha(const Float_t alpha)

Interpolation Method 2

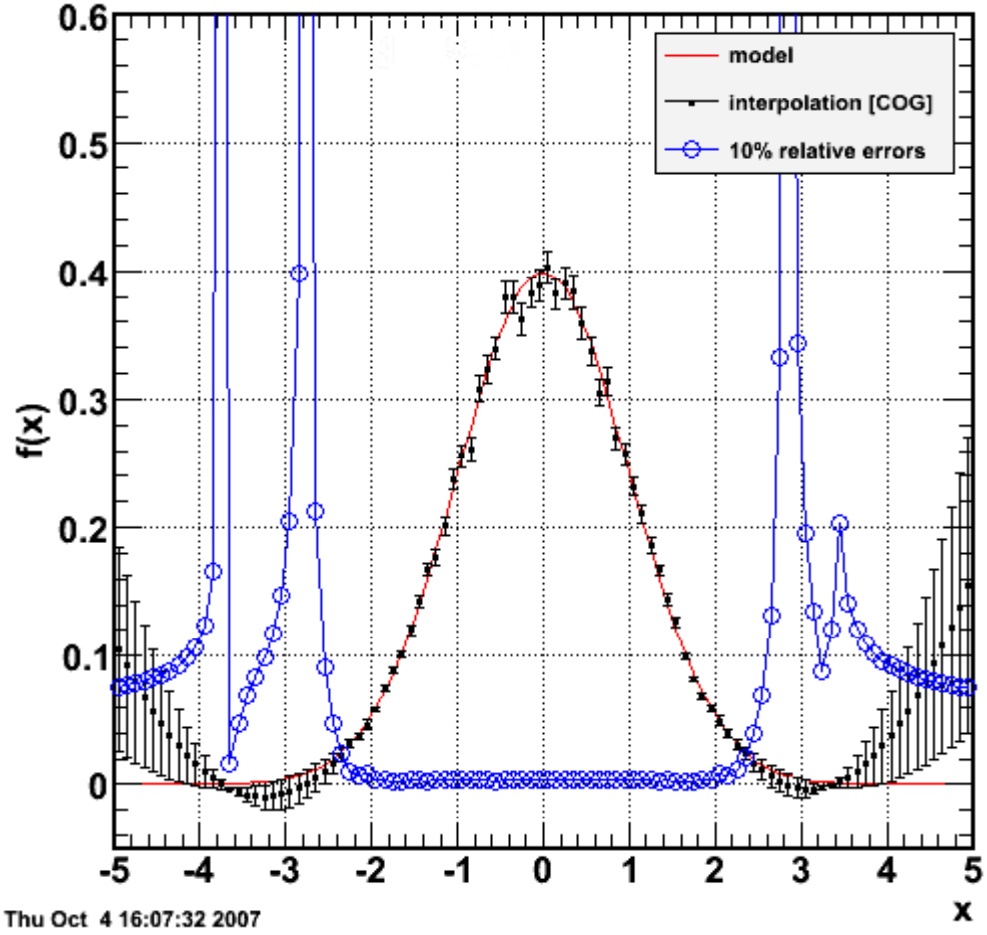
-Least square *plane fit* using integral over bucket

$$I/S = \frac{1}{x_2 - x_1}\int_{x_1}^{x_2}(p_0 + p_1 x + p_2 x^2)dx \quad =$$

$$= p_0 + p_1\frac{x_2 + x_1}{2} + p_2\frac{x_2^2 + x_2 x_1 + x_1^2}{3}$$

$$= p_0 + p_1 W_0 + p_2 W_1$$

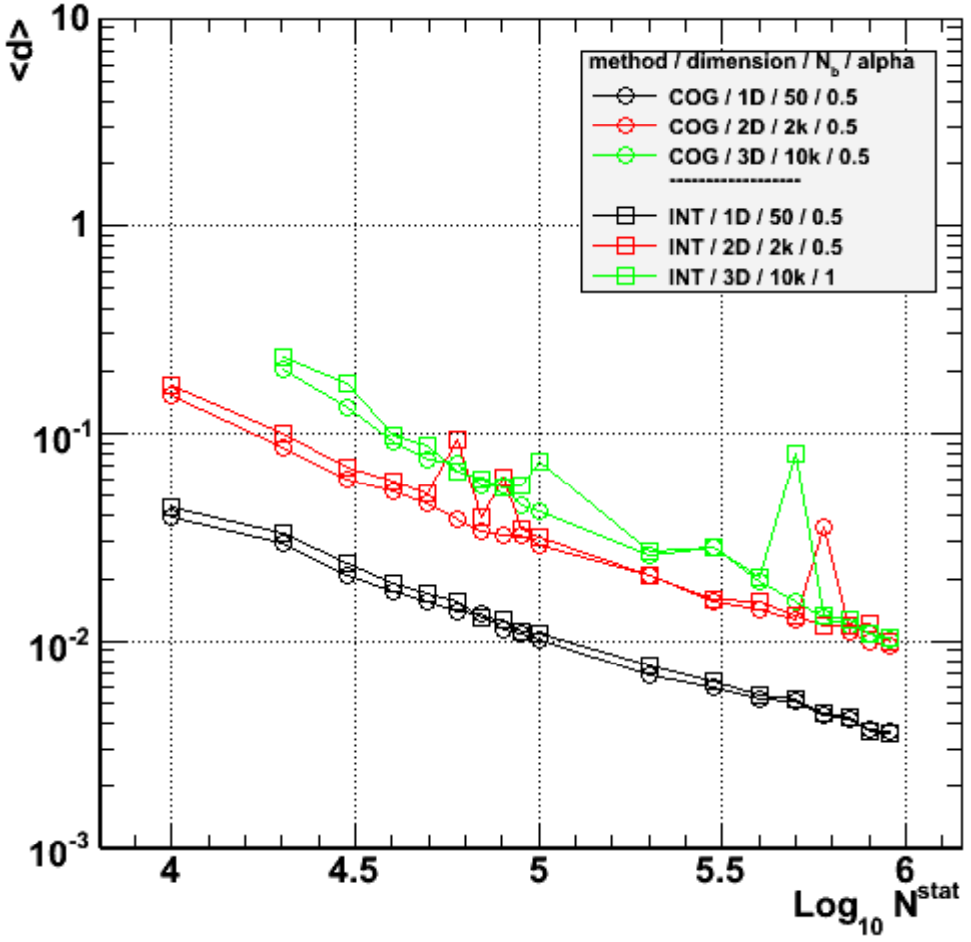
$$\Rightarrow f(X), \sigma(X)$$



TKDInterpolator quality

$$\chi^2 = \frac{1}{N}\sum\frac{(E_i - M_i)^2}{\sigma_i^2} \approx 1$$

$$< d >= \sqrt{\frac{1}{N}\sum\frac{(E_i - M_i)^2}{M_i}} \approx \sqrt{\frac{1}{N}}$$



TKDInterpolator costs

- NodeInfo{

```
 Int_t        fNDim;
 Float_t     *fRefPoint;
 Float_t      fRefValue;
 TMatrixD *fCov;   } only Method 2 (INT)
 TVectorD  *fPar;  }
};
```

- Example 2D/32 bits

M1 : 32 bytes/node

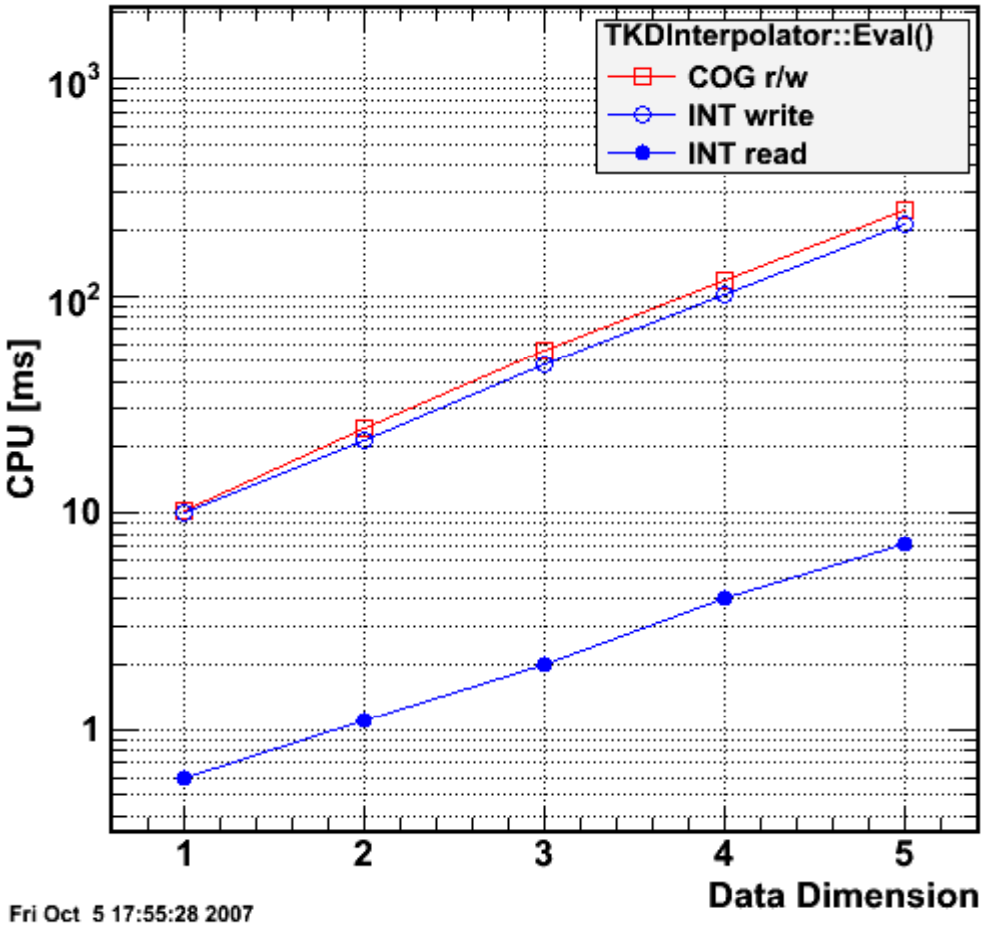
M2 : 352 bytes/node

- Size on disk (KB)

	1D	2D	3D	4D	5D
COG	8	8	8	12	12
INT	28	72	168	356	672

- Eval method (bs = 400)

- Interpolation speed (100k/400)



Conclusions

- A first implementation of the package is available in $ALICE_ROOT/STAT module.
- Example applications in $ALICE_ROOT/STAT/Macros.
- Performance tests (CPU and memory).
- The code is stable.

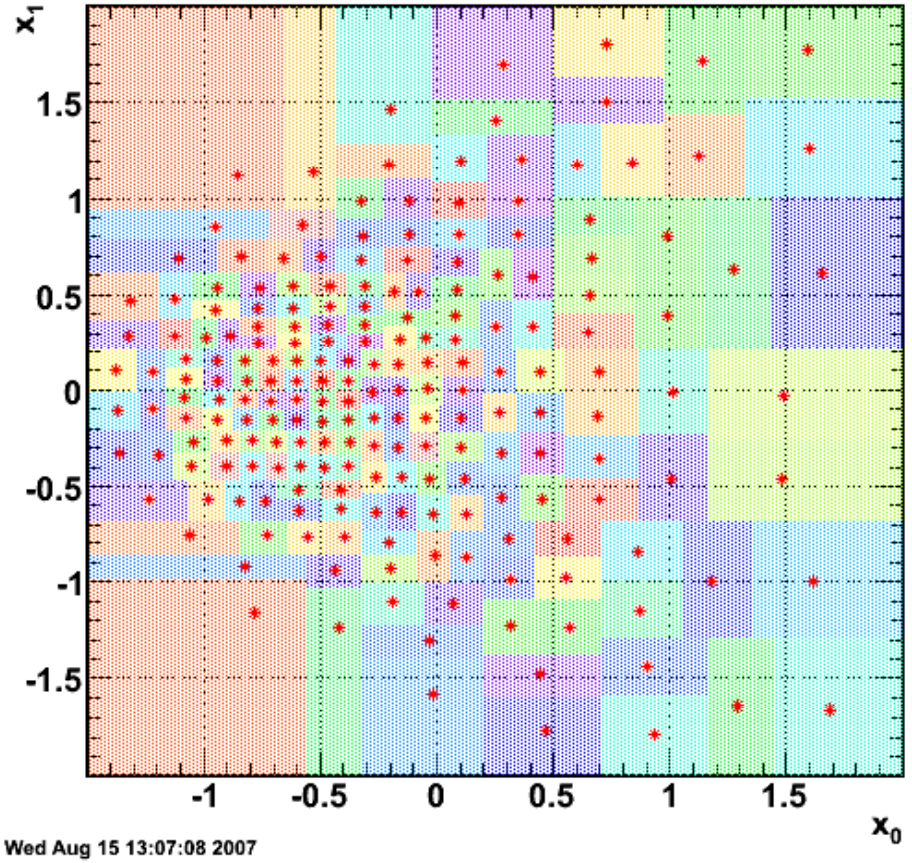
To Do

- Implementations for tracking and TRD PID
- Local regression – calibration data

Proof of Principle

Test steps

- 2D distribution f = Landau(x)*Landau(y)
- χ^2 between estimate and true values
- χ^2 as a function of statistics



Data preprocessing

- no
- outlier/tail compression
- PCA

Convergence of the interpolation procedure

$$\chi^2 = \sum_{ip} \frac{(O_{ip} - E_{ip})^2}{E_{ip}}$$