CQF FitchLearning

FINAL PROJECT TUTORIAL I

Credit Correlation (Matlab) Pseudo-Samples for Copula

Dr Richard Diamond
CQF ARPM

July 2021

Knowledge | Skills | Conduct

Techniques for **Credit Pricing**

A. MATLAB Demo of *ksdensity()* to obtain pseudo-samples from historical data. Kernel function smoothes over **pdf**, and then **cdf** obtained.

B. Questions on spread pricing -- as requested.

Matlab **ksdensity()** converts to uniformly-distributed “scores”

```
%% Pseudo Samples by KS Method (via pdf and cdf estimation)

CDSPseudo = zeros(Nt_sampling-1,Nref);
for k=1:1:Nref
CDSPseudo(:,k) = ksdensity(CDSDiff(:,k),CDSDiff(:,k),'function','cdf'...
                ,'width',1.e-2);
end
```

Bandwidth 1.E-2=0.01. Regulate this parameter to 0.001 or smaller.

```
%% Empirical CDF Explorations - How smooth your CDF is?

k = 4 % for France -- Empirical CDF has sharp jump about zero
k =1  % for Italy -- Empirical CDF is SMOOTH and that makes all difference

[Fi,xi] = ecdf(CDSDiff(:,k));
figure()
stairs(xi,Fi,'b','LineWidth',2)
hold on

Fi_sm = ksdensity(CDSDiff(:,k), xi,'function','cdf'...
                ,'width',1.e-2);
plot(xi,Fi_sm,'r-','LineWidth',1.5)
xlabel('X1')
ylabel('Cumulative Probability')
legend('Empirical','Smoothed','Location','NW')
grid on
```

Install and load **ks library in R** – the best available alternative to Matlab

```
pseudo.uniform = function(X){
  # This function Calculates pseudo-uniform observations using ker
  # Requires 'ks' package to be loaded.

  # First we estimate the CDF
  Fhat <- kcde(X)
  # Plug in the values into the CDF to obtain pseudo-observations
  predict(Fhat, x=X)
}
```

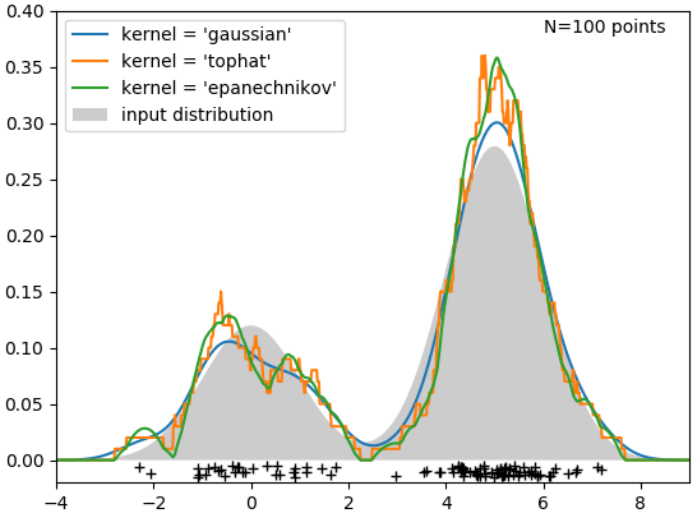
Python's **sklearn.**preprocessing.QuantileTransformer has uniform distribution as default output ('mapping to' uniform).

Said to be useful for the data with sparse range of values, eg "outliers that are common rather than rare." Though, outliers are getting collapsed to the [0,1] range, seen through saturation on 2D scatters.

```
trans = QuantileTransformer(n_quantiles=100, output_distribution='uniform')
data = trans.fit_transform(data)
```

Recommended to review: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html

In terms of full explicit kernel smoothing procedure, **Python** ecosystem leaves the job unfinished: you can get ***pdf*** fitted by advanced kernels. But, for **cdf U=F(X)** you need to complete this final step.



Recipe 1: **cdf** can be found by analytical integration over kernel function, then cdf expression to be used to compute empirical value.

Recipe 2: perform numerical integration with *integrate_box_1d()* or its underlying workhorse *special.ndtr* (package scipy.stats)

```
from scipy.stats import norm, gaussian_kde
kde = gaussian_kde(n)
```

```
from scipy.special import ndtr
stdev = np.sqrt(kde.covariance)[0, 0]
pde_cdf = ndtr(np.subtract.outer(x, n)).mean(axis=1)
plot(x, pde_cdf)
```

List of kernel functions for **pdf** only https://scikit-learn.org/stable/modules/density.html (Link 1)

We also know that KDE object has **cdf** and **inverse cdf**
https://www.statsmodels.org/dev/examples/notebooks/generated/kernel_density.html (Link 2)
kde = sm.nonparametric.KDEUnivariate(obs_dist)
kde.fit() # *Estimate the densities*