COMPUTO

ISSN 2824-7795

Optimal projection for parametric importance sampling in high dimension

Maxime El Masri ONERA/DTIS, ISAE-SUPAERO, Université de Toulouse
Jérôme Morio ONERA/DTIS, Université de Toulouse
Florian Simatos ISAE-SUPAERO, Université de Toulouse

Abstract

In this paper we propose a dimension-reduction strategy in order to improve the performance of importance sampling in high dimension. The idea is to estimate variance terms in a small number of suitably chosen directions. We first prove that the optimal directions, i.e., the ones that minimize the Kullback–Leibler divergence with the optimal auxiliary density, are the eigenvectors associated to extreme (small or large) eigenvalues of the optimal covariance matrix. We then perform extensive numerical experiments that show that as dimension increases, these directions give estimations which are very close to optimal. Moreover, we show that the estimation remains accurate even when a simple empirical estimator of the covariance matrix is used to estimate these directions. These theoretical and numerical results open the way for different generalizations, in particular the incorporation of such ideas in adaptive importance sampling schemes.

Keywords: Importance sampling, High dimension, Gaussian covariance matrix, Kullback-Leibler divergence, Projection

Contents


```
import numpy as np
import scipy as sp
import scipy.stats
import matplotlib.pyplot as plt
from CEIS_vMFNM import *
from IPython.display import display, Math, Latex
from IPython.display import Markdown
from tabulate import tabulate
np.random.seed(10)
```

0.1 Application: large portfolio losses

The next example is a rare event application in finance, taken from (Bassamboo, Juneja, and Zeevi 2008), (Chan and Kroese 2012}). The unknown integral is $E = \int_{\mathbb{R}^{n+2}} \phi(\mathbf{x}) f(\mathbf{x}) \mathrm{d}\mathbf{x} = \mathbb{P}_f(\varphi(\mathbf{X}) \geq 0)$, with $\phi = \mathbb{I}_{\{\varphi \geq 0\}}$ and f is the standard $n+2$-dimensional Gaussian distribution. The function φ is the portfolio loss function defined as:

$$\varphi(\mathbf{x}) = \sum_{j=3}^{n+2} \mathbb{I}_{\{\Psi(x_1, x_2, x_j) \geq 0.5\sqrt{n}\}} - 0.25n, \tag{1}$$

with

$$\Psi(x_1, x_2, x_j) = \left(q x_1 + 3(1-q^2)^{1/2} x_j\right) \left[F_\Gamma^{-1}\left(F_{\mathcal{N}}(x_2)\right)\right]^{-1/2},$$

where F_Γ and $F_{\mathcal{N}}$ are the cumulative distribution functions of Gamma$(6,6)$ and $\mathcal{N}(0,1)$ random variables respectively.

The reference value of this probability E is reported in Table 1 for dimension $n = 100$. The optimal parameters $\mathbf{m}^*$ and Σ^* cannot be computed analytically, but they are accurately estimated by Monte Carlo with a large sample. It turns out that $\mathbf{m}^*$ and the first eigenvector $\mathbf{d}_1^*$ of Σ^* are numerically indistinguishable and that Algorithm 2 selects $k = 1$ projection direction, so that numerically, the choices $\widehat{\Sigma}_{\text{opt}}$ and $\widehat{\Sigma}_{\text{mean}}$ are indistinguishable and gives the same estimation results. Actually, the fact that these two estimators behave similarly does not seem to come from the fact that $\mathbf{m}^*$ and $\mathbf{d}^*$ are close: this relation can be broken for instance by a simple translation argument, but even then they behave similarly. The KL partial divergence and the spectrum with the associated ℓ-order are presented respectively in Figure 1a and in Figure 1b.

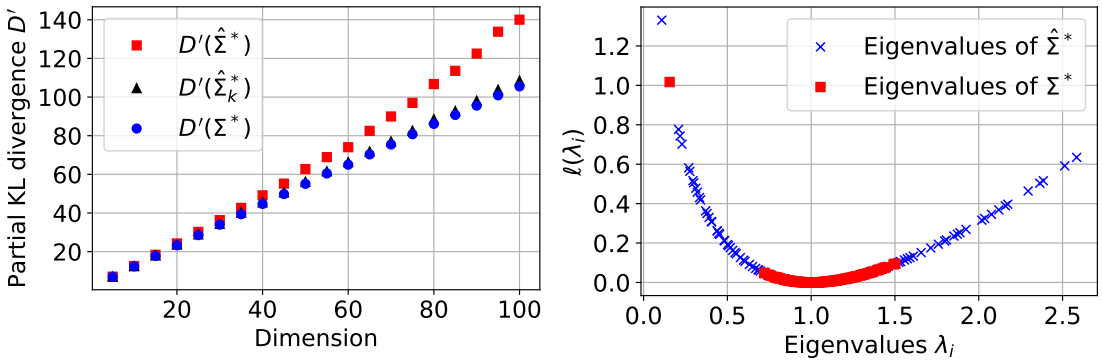
```
##########################################################################################################
# Figure 5. Evolution of the partial KL divergence and spectrum of the eigenvalues for the lar
##########################################################################################################

def Portfolio(X):
    N=np.shape(X)[0]
    nn=np.shape(X)[1]
    n=nn-2
    lamb=np.array(sp.stats.gamma.ppf(sp.stats.norm.cdf(X[:,0]),6,scale=1/6),ndmin=2).T
    eta=3*X[:,2:]
    ZZ=np.array(X[:,1],ndmin=2).T

    XX=(1/4*ZZ+np.sqrt(1-1/4**2)*eta)/np.sqrt(lamb)
    IndX=(XX>0.5*np.sqrt(n))*1
    PF=np.sum(IndX,axis=1)
    return(PF-0.25*n-0.1)

def Portfolio_md(X):
    N=np.shape(X)[0]
```

```
    nn=np.shape(X)[1]
    n=nn-2
    lamb=np.array( sp.stats.gamma.ppf(sp.stats.norm.cdf(X[:,0]),6,scale=1/6),ndmin=2).T
    eta=3*X[:,2:]
    ZZ=np.array(X[:,1],ndmin=2).T

    XX=(1/4*ZZ+np.sqrt(1-1/4**2)*eta)/np.sqrt(lamb)
    IndX=(XX>0.5*np.sqrt(n))*1
    PF=np.sum(IndX,axis=1)
    return(PF-0.3*n-0.1)

def Portfolio_ld(X):
    N=np.shape(X)[0]
    nn=np.shape(X)[1]
    n=nn-2
    lamb=np.array(sp.stats.gamma.ppf(sp.stats.norm.cdf(X[:,0]),6,scale=1/6),ndmin=2).T
    eta=3*X[:,2:]
    ZZ=np.array(X[:,1],ndmin=2).T

    XX=(1/4*ZZ+np.sqrt(1-1/4**2)*eta)/np.sqrt(lamb)
    IndX=(XX>0.5*np.sqrt(n))*1
    PF=np.sum(IndX,axis=1)
    return(PF-0.45*n-0.1)


DKL=np.zeros(20)
DKLp=np.zeros(20)
DKLm=np.zeros(20)
DKLstar=np.zeros(20)

n=100
bigsample=20*10**5
M=300

for d in range(5,n+1,5):

    if d<=30:
        phi=Portfolio_ld
    if d>70:
        phi=Portfolio
    else:
        phi=Portfolio_md
```

```
VA=sp.stats.multivariate_normal(mean=np.zeros(d+2),cov=np.eye(d+2))
X01=VA.rvs(size=bigsample)
ind1=(phi(X01)>0)
X1=X01[ind1,:]
X1=X1[:M*10,:]
#Mstar
Mstar=np.mean(X1.T,axis=1)
#Sigmastar
X1c=(X1-Mstar).T
Sigstar=X1c.dot(X1c.T)/np.shape(X1c)[1]


## g*-sample
VA0=sp.stats.multivariate_normal(mean=np.zeros(d+2),cov=np.eye(d+2))
X0=VA0.rvs(size=M*1000)

ind=(phi(X0)>0)
X=X0[ind,:]
X=X[:M,:]            # g*-sample of size M

## estimated mean and covariance
mm=np.mean(X,axis=0)

Xc=(X-mm).T
sigma =Xc @ Xc.T/np.shape(Xc)[1]

## projection with the eigenvalues of sigma
Eig=np.linalg.eigh(sigma)
logeig=np.sort(np.log(Eig[0])-Eig[0])
delta=np.zeros(len(logeig)-1)
for j in range(len(logeig)-1):
    delta[j]=abs(logeig[j]-logeig[j+1])

k=np.argmax(delta)+1         # biggest gap between the l(lambda_i)

indi=[]
for l in range(k):
    indi.append(np.where(np.log(Eig[0])-Eig[0]==logeig[l])[0][0])

P1=np.array(Eig[1][:,indi[0]],ndmin=2).T                  # projection matrix
for l in range(1,k):
    P1=np.concatenate((P1,np.array(Eig[1][:,indi[l]],ndmin=2).T),axis=1)
```

```
    diagsi=np.diag(Eig[0][indi])
    sig_opt_d=P1.dot((diagsi-np.eye(k))).dot(P1.T)+np.eye(d+2)

    DKL[int((d-5)/5)]=np.log(np.linalg.det(sigma))+np.sum(np.diag(Sigstar.dot(np.linalg.inv(si
    DKLp[int((d-5)/5)]=np.log(np.linalg.det(sig_opt_d))+np.sum(np.diag(Sigstar.dot(np.linalg.i
    DKLstar[int((d-5)/5)]=np.log(np.linalg.det(Sigstar))+d+2

#### plot of partial KL divergence
plt.plot(range(5,n+1,5),DKL,'rs',label=r"$D'(\hat{\Sigma}^*)$")
plt.plot(range(5,n+1,5),DKLp,'k^',label=r"$D'(\hat{\Sigma}^*_k)$")
plt.plot(range(5,n+1,5),DKLstar,'bo',label=r"$D'(\Sigma^*)$")

plt.grid()
plt.xlabel('Dimension',fontsize=16)
plt.ylabel(r"Partial KL divergence $D'$",fontsize=16)
plt.legend(fontsize=16)
for tickLabel in plt.gca().get_xticklabels() + plt.gca().get_yticklabels():
    tickLabel.set_fontsize(16)
plt.show()

#### plot of the eigenvalues
Eig1=np.linalg.eigh(sigma)
logeig1=np.log(Eig1[0])-Eig1[0]+1
Table_eigv=np.zeros((n+2,2))
Table_eigv[:,0]=Eig1[0]
Table_eigv[:,1]=-logeig1

Eigst=np.linalg.eigh(Sigstar)
logeigst=np.log(Eigst[0])-Eigst[0]+1
Table_eigv_st=np.zeros((n+2,2))
Table_eigv_st[:,0]=Eigst[0]
Table_eigv_st[:,1]=-logeigst

plt.grid()
plt.xlabel(r"Eigenvalues $\lambda_i$",fontsize=16)
plt.ylabel(r"$\ell(\lambda_i)$",fontsize=16)
for tickLabel in plt.gca().get_xticklabels() + plt.gca().get_yticklabels():
    tickLabel.set_fontsize(16)

plt.plot(Table_eigv[:,0],Table_eigv[:,1],'bx',label=r"Eigenvalues of $\hat{\Sigma}^*$")
plt.plot(Table_eigv_st[:,0],Table_eigv_st[:,1],'rs',label=r"Eigenvalues of $\Sigma^*$")
plt.legend(fontsize=16)
```

```
plt.show()
```



(a) Evolution of the partial KL divergence as the dimension increases, with the optimal covariance matrix Σ^* (blue circles), the sample covariance $\widehat{\Sigma}^*$ (red squares), and the projected covariance $\widehat{\Sigma}^*_k$ (black triangles).

(b) Computation of $\ell(\lambda_i)$ for the eigenvalues of Σ^* (red squares) and $\widehat{\Sigma}^*$ (blue crosses) in dimension $n = 100$ for the large portfolio losses of Equation 1.

Figure 1: Partial KL divergence and spectrum for the function $\phi = \mathbb{I}_{\varphi \geq 0}$ with φ the function given by Equation 1.

```
##############################################################################################
# Table 5. Numerical comparison on the large portfolio loss application
##############################################################################################

n=100          # dimension
phi=Portfolio
E=1.82*10**(-3)

def mypi(X):
    nn=np.shape(X)[1]
    n=nn-2
    f0=sp.stats.multivariate_normal.pdf(X,mean=np.zeros(nn),cov=np.eye(nn))
    return((phi(X)>0)*f0)


N=2000
M=500
B=50     # number of runs

Eopt=np.zeros(B)
EIS=np.zeros(B)
```

```
Eprj=np.zeros(B)
Eprm=np.zeros(B)
Eprjst=np.zeros(B)
Eprmst=np.zeros(B)
Evmfn=np.zeros(B)

SI=[]
SIP=[]
SIPst=[]
SIM=[]
SIMst=[]

bigsample=1*10**6

VA=sp.stats.multivariate_normal(mean=np.zeros(n+2),cov=np.eye(n+2))
X01=VA.rvs(size=bigsample)
ind1=(phi(X01)>0)
X1=X01[ind1,:]

#Mstar
Mstar=np.mean(X1.T,axis=1)
#Sigmastar
X1c=(X1-Mstar).T
Sigstar=X1c.dot(X1c.T)/np.shape(X1c)[1]

Eigst=np.linalg.eigh(Sigstar)
logeigst=np.sort(np.log(Eigst[0])-Eigst[0])
deltast=np.zeros(len(logeigst)-1)

for i in range(len(logeigst)-1):
    deltast[i]=abs(logeigst[i]-logeigst[i+1])

## choice of the number of dimension
k_st=np.argmax(deltast)+1

indist=[]
for i in range(k_st):
    indist.append(np.where(np.log(Eigst[0])-Eigst[0]==logeigst[i])[0][0])

P1st=np.array(Eigst[1][:,indist[0]],ndmin=2).T
for i in range(1,k_st):
    P1st=np.concatenate((P1st,np.array(Eigst[1][:,indist[i]],ndmin=2).T),axis=1)       # matr
```

```
#np.random.seed(0)
for i in range(B):
############################## Estimation of the matrices

   ## g*-sample of size M
    VA=sp.stats.multivariate_normal(np.zeros(n+2),np.eye(n+2))
    X0=VA.rvs(size=M*1000)
    ind=(phi(X0)>0)
    X=X0[ind,:]
    X=X[:M,:]

    R=np.sqrt(np.sum(X**2,axis=1))
    Xu=(X.T/R).T

   ## estimated gaussian mean and covariance
    mm=np.mean(X,axis=0)
    Xc=(X-mm).T
    sigma =Xc @ Xc.T/np.shape(Xc)[1]
    SI.append(sigma)

   ## von Mises Fisher parameters
    normu=np.sqrt(np.mean(Xu,axis=0).dot(np.mean(Xu,axis=0).T))
    mu=np.mean(Xu,axis=0)/normu
    mu=np.array(mu,ndmin=2)
    chi=min(normu,0.95)
    kappa=(chi*n-chi**3)/(1-chi**2)

   ## Nakagami parameters
    omega=np.mean(R**2)
    tau4=np.mean(R**4)
    pp=omega**2/(tau4-omega**2)

   ###
    Eig=np.linalg.eigh(sigma)
    logeig=np.sort(np.log(Eig[0])-Eig[0])
    delta=np.zeros(len(logeig)-1)
    for j in range(len(logeig)-1):
        delta[j]=abs(logeig[j]-logeig[j+1])

    k=np.argmax(delta)+1

    indi=[]
```

```
    for l in range(k):
        indi.append(np.where(np.log(Eig[0])-Eig[0]==logeig[l])[0][0])

    P1=np.array(Eig[1][:,indi[0]],ndmin=2).T
    for l in range(1,k):
        P1=np.concatenate((P1,np.array(Eig[1][:,indi[l]],ndmin=2).T),axis=1)

    diagsi=np.diag(Eig[0][indi])
    sig_opt_d=P1.dot((diagsi-np.eye(k))).dot(P1.T)+np.eye(n+2)
    SIP.append(sig_opt_d)

   ###
    diagsist=P1st.T.dot(sigma).dot(P1st)
    sig_opt=P1st.dot(diagsist-np.eye(k_st)).dot(P1st.T)+np.eye(n+2)
    SIPst.append(sig_opt)

   ###
    Norm_mm=np.linalg.norm(mm)
    normalised_mm=np.array(mm,ndmin=2).T/Norm_mm
    vhat=normalised_mm.T.dot(sigma).dot(normalised_mm)
    sig_mean_d=(vhat-1)*normalised_mm.dot(normalised_mm.T)+np.eye(n+2)
    SIM.append(sig_mean_d)

   ###
    Norm_Mstar=np.linalg.norm(Mstar)
    normalised_Mstar=np.array(Mstar,ndmin=2).T/Norm_Mstar
    vhatst=normalised_Mstar.T.dot(sigma).dot(normalised_Mstar)

    sig_mean=(vhatst-1)*normalised_Mstar.dot(normalised_Mstar.T)+np.eye(n+2)
    SIMst.append(sig_mean)

############################################### Estimation of the integral
   ###
    Xop=sp.stats.multivariate_normal.rvs(mean=mm, cov=Sigstar,size=N)
    wop=mypi(Xop)/sp.stats.multivariate_normal.pdf(Xop,mean=mm, cov=Sigstar)
    Eopt[i]=np.mean(wop)

   ###
    Xis=sp.stats.multivariate_normal.rvs(mean=mm, cov=sigma,size=N)
    wis=mypi(Xis)/sp.stats.multivariate_normal.pdf(Xis,mean=mm, cov=sigma)
    EIS[i]=np.mean(wis)
```

```
    ###
     Xpr=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_opt_d,size=N)
     wpr=mypi(Xpr)/sp.stats.multivariate_normal.pdf(Xpr,mean=mm, cov=sig_opt_d)
     Eprj[i]=np.mean(wpr)

    ###
     Xpm=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_mean_d,size=N)
     wpm=mypi(Xpm)/sp.stats.multivariate_normal.pdf(Xpm,mean=mm, cov=sig_mean_d)
     Eprm[i]=np.mean(wpm)

    ###
     Xprst=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_opt,size=N)
     wprst=mypi(Xprst)/sp.stats.multivariate_normal.pdf(Xprst,mean=mm, cov=sig_opt)
     Eprjst[i]=np.mean(wprst)

    ###
     Xpmst=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_mean,size=N)
     wpmst=mypi(Xpmst)/sp.stats.multivariate_normal.pdf(Xpmst,mean=mm, cov=sig_mean)
     Eprmst[i]=np.mean(wpmst)

    ###
     Xvmfn = vMFNM_sample(mu, kappa, omega, pp, 1, N)
     Rvn=np.sqrt(np.sum(Xvmfn**2,axis=1))
     Xvnu=Xvmfn.T/Rvn

     h_log=vMF_logpdf(Xvnu,mu.T,kappa)+nakagami_logpdf(Rvn,pp,omega)
     A = np.log(n+2) + np.log(np.pi ** ((n+2) / 2)) - sp.special.gammaln((n+2) / 2 + 1)
     f_u = -A
     f_chi = (np.log(2) * (1 - (n+2) / 2) + np.log(Rvn) * ((n+2) - 1) - 0.5 * Rvn ** 2 - sp.spe
     f_log = f_u + f_chi
     W_log = f_log - h_log

     wvmfn=(phi(Xvmfn)>0)*np.exp(W_log)
     Evmfn[i]=np.mean(wvmfn)

### KL divergences
dkli=np.zeros(B)
dklp=np.zeros(B)
dklm=np.zeros(B)
dklpst=np.zeros(B)
dklmst=np.zeros(B)
dklpca=np.zeros(B)
```

```
for i in range(B):
    dkli[i]=np.log(np.linalg.det(SI[i]))+sum(np.diag(Sigstar.dot(np.linalg.inv(SI[i]))))
    dklp[i]=np.log(np.linalg.det(SIP[i]))+sum(np.diag(Sigstar.dot(np.linalg.inv(SIP[i]))))
    dklm[i]=np.log(np.linalg.det(SIM[i]))+sum(np.diag(Sigstar.dot(np.linalg.inv(SIM[i]))))
    dklpst[i]=np.log(np.linalg.det(SIPst[i]))+sum(np.diag(Sigstar.dot(np.linalg.inv(SIPst[i]))
    dklmst[i]=np.log(np.linalg.det(SIMst[i]))+sum(np.diag(Sigstar.dot(np.linalg.inv(SIMst[i]))

Tabresult=np.zeros((3,7)) # table of results

Tabresult[0,0]=np.log(np.linalg.det(Sigstar))+n+2
Tabresult[0,1]=np.mean(dkli)
Tabresult[0,2]=np.mean(dklpst)
Tabresult[0,3]=np.mean(dklmst)
Tabresult[0,4]=np.mean(dklp)
Tabresult[0,5]=np.mean(dklm)
Tabresult[0,6]=None

Tabresult[1,0]=np.mean(Eopt-E)/E*100
Tabresult[1,1]=np.mean(EIS-E)/E*100
Tabresult[1,2]=np.mean(Eprjst-E)/E*100
Tabresult[1,3]=np.mean(Eprmst-E)/E*100
Tabresult[1,4]=np.mean(Eprj-E)/E*100
Tabresult[1,5]=np.mean(Eprm-E)/E*100
Tabresult[1,6]=np.mean(Evmfn-E)/E*100

Tabresult[2,0]=np.sqrt(np.mean((Eopt-E)**2))/E*100
Tabresult[2,1]=np.sqrt(np.mean((EIS-E)**2))/E*100
Tabresult[2,2]=np.sqrt(np.mean((Eprjst-E)**2))/E*100
Tabresult[2,3]=np.sqrt(np.mean((Eprmst-E)**2))/E*100
Tabresult[2,4]=np.sqrt(np.mean((Eprj-E)**2))/E*100
Tabresult[2,5]=np.sqrt(np.mean((Eprm-E)**2))/E*100
Tabresult[2,6]=np.sqrt(np.mean((Evmfn-E)**2))/E*100

Tabresult=np.round(Tabresult,1)

table=[["D'",Tabresult[0,0],Tabresult[0,1],Tabresult[0,2],Tabresult[0,3],Tabresult[0,4],Tabres
      ["Relative error (\%)",Tabresult[1,0],Tabresult[1,1],Tabresult[1,2],Tabresult[1,3],Tabre
    ["Coefficient of variation (\%)",Tabresult[2,0],Tabresult[2,1],Tabresult[2,2],Tabresult[2
Markdown(tabulate(
  table,
  headers=["","$\Sigma^*$", "$\hat{\Sigma}^*$", "$\hat{\Sigma}_{opt}$", "$\hat{\Sigma}_{mean}$
    tablefmt="pipe"))
```

Table 1: Numerical comparison of the estimation of $E \approx 1.82 \cdot 10^{-3}$ considering the Gaussian density with the six covariance matrices defined in **?@sec-def_cov** and the vFMN model,$\phi = \mathbb{I}_{\varphi \geq 0}$ with φ given by Equation 1.

	Σ^*	$\hat{\Sigma}^*$	$\hat{\Sigma}_{opt}$	$\hat{\Sigma}_{mean}$	$\hat{\Sigma}^{+d}_{opt}$	$\hat{\Sigma}^{+d}_{mean}$	vMFN
D'	104.3	122.6	107.5	107.6	108	107.7	nan
Relative error (%)	-2.6	15.1	0.4	0.1	-1.6	0.2	-1.1
Coefficient of variation (%)	17.1	482.3	5.8	7.7	8.9	7.7	6.9

Bassamboo, Achal, Sandeep Juneja, and Assaf Zeevi. 2008. "Portfolio Credit Risk with Extremal Dependence: Asymptotic Analysis and Efficient Simulation." *Operations Research* 56 (3): 593–606. https://doi.org/10.1287/opre.1080.0513.

Chan, Joshua C. C., and Dirk P. Kroese. 2012. "Improved Cross-Entropy Method for Estimation." *Statistics and Computing* 22 (5): 1031–40. https://doi.org/10.1007/s11222-011-9275-7.