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 pickle
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)
```

```
np.random.rand(1,1)
```

```
array([[0.77132064]])
```

```
################################################################################
# Table 2. Numerical comparison on test case 1
################################################################################
def Somme(x):
    n=np.shape(x)[1]
    return(np.sum(x,axis=1)-3*np.sqrt(n))

E=sp.stats.norm.cdf(-3)
n=100
d=n
# dimension
phi=Somme

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

N=2000
M=500
B=10   # 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=[]
print(np.random.rand(1,1))
# Mstar
alpha=np.exp(-3**2/2)/(E*np.sqrt(2*np.pi))
Mstar=alpha*np.ones(d)/np.sqrt(d)
# Sigmastar
vstar=3*alpha-alpha**2+1
Sigstar= (vstar-1)*np.ones((d,d))/d+np.eye(d)

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)    # matrix of influential directions

#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),np.eye(n))
 X0=VA.rvs(size=M*1000)
 ind=(phi(X0)>0)
 X1=X0[ind,:]
 X=X1[: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)
    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)
    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)
    SIM.append(sig_mean_d)

   ### MODIF : Sigma_opt=Sigma_mean donc inutile de recalculer

#     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)
#     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)

   ###
    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) + np.log(np.pi ** (n / 2)) - sp.special.gammaln(n / 2 + 1)
    f_u = -A
    f_chi = (np.log(2) * (1 - n / 2) + np.log(Rvn) * (n - 1) - 0.5 * \
             Rvn ** 2 - sp.special.gammaln(n / 2))
    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]))))

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

Tabresult[0,0]=np.log(np.linalg.det(Sigstar))+n
Tabresult[0,1]=np.mean(dkli)
Tabresult[0,2]=np.mean(dklpst)
Tabresult[0,3]=np.mean(dklpst)
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(Eprjst-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((Eprjst-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

print(Tabresult)

Tabresult=np.round(Tabresult,1)

table=[["D'",Tabresult[0,0],Tabresult[0,1],Tabresult[0,2],Tabresult[0,3],
        Tabresult[0,4],Tabresult[0,5],"/"],
      ["Relative error (\%)",Tabresult[1,0],Tabresult[1,1],
       Tabresult[1,2],Tabresult[1,3],Tabresult[1,4],Tabresult[1,5],Tabresult[1,6]],
```

```
    ["Coefficient of variation (\%)",Tabresult[2,0],Tabresult[2,1],
     Tabresult[2,2],Tabresult[2,3],Tabresult[2,4],Tabresult[2,5],Tabresult[2,6]]]

Markdown(tabulate(
  table,
  headers=["","$\Sigma^*$", "$\widehat{\Sigma}^*$", "$\widehat{\Sigma}_{opt}$",\
           "$\widehat{\Sigma}_{mean}$", "${\widehat{\Sigma}^{+d}_{opt}}$",\
           "$\widehat{\Sigma}^{+d}_{mean}$", "vMFN"],
    tablefmt="pipe"))
```

```
[[0.02075195]]
```

```
[[ 9.73486966e+01  1.12015755e+02  9.73563826e+01  9.73563826e+01
   9.76914691e+01  9.75603401e+01             nan]
 [ 2.57126814e-01  1.57591697e+02 -1.48283641e+00 -1.48283641e+00
   1.10982023e+00 -5.94374390e-01 -3.50974386e-01]
 [ 6.85533631e+00  5.48038307e+02  2.27324417e+00  2.27324417e+00
   4.84456957e+00  4.96796423e+00  3.64048041e+00]]
```

Table 1: Numerical comparison of the estimation of $E \approx 1.35 \cdot 10^{-3}$ considering the Gaussian model with the six covariance matrices defined in **?@sec-def_cov** and the vFMN model, when $\phi = \mathbb{I}_{\varphi \geq 0}$ with φ the linear function given by **?@eq-sum**. As explained in the text, $\widehat{\Sigma}_{\text{mean}}$ and $\widehat{\Sigma}_{\text{opt}}$ are actually equal in this case.

	Σ^*	$\widehat{\Sigma}^*$	$\widehat{\Sigma}_{opt}$	$\widehat{\Sigma}_{mean}$	$\widehat{\Sigma}^{+d}_{opt}$	$\widehat{\Sigma}^{+d}_{mean}$	vMFN
D’	97.3	112	97.4	97.4	97.7	97.6	/
Relative error (%)	0.3	157.6	-1.5	-1.5	1.1	-0.6	- 0.4
Coefficient of variation (%)	6.9	548	2.3	2.3	4.8	5	3.6

```
np.random.rand(1,1)
```

```
array([[0.89818344]])
```

```
print(Tabresult)
```

```
[[ 9.730e+01  1.120e+02  9.740e+01  9.740e+01  9.770e+01  9.760e+01
         nan]
 [ 3.000e-01  1.576e+02 -1.500e+00 -1.500e+00  1.100e+00 -6.000e-01
  -4.000e-01]
 [ 6.900e+00  5.480e+02  2.300e+00  2.300e+00  4.800e+00  5.000e+00
   3.600e+00]]
```