Notebooks exportation

Notebook conversion from markdown using pandoc

Romain Madar[1]

[1]Laboratoire de Physique de Clermont-Ferrand

2018-02-14

Abstract

These files were obtained from jupyter notebook and converted into a markdown using nbconvert. The markdown files and figures are taken as input to produce a global pdf using pandoc.

Contents

1 Regression using gaussian processes

The material I used to build up this short tutorial can be found in http://katbailey.github.io/post/gaussian-processes-for-dummies/ (where the code were adapated from) and in this short lecture which gives a mathematical simplified (and intuitive) approach https://emtiyaz.github.io/pcml15/gp_by_carlos.pdf.

1.1 Introduction to gaussian processes

Gaussian processes (GP) is a statistical method allowing to interpolate measurements $\boldsymbol{y} = \{y_i\}$ made at points $\boldsymbol{x} = \{x_i\}$. The basica idea is to assume that each y_i is a random variable with a gaussian probability density function (PDF). Once the correlation matrix $K = \{K_{ij}\} \equiv \{\mathscr{K}(x_i, x_j)\} \equiv \mathscr{K}(\boldsymbol{x}, \boldsymbol{x})$ is specified, where $\mathscr{K}(x, x')$ is called the kernel, the joint PDF (or multi-dimentional PDF) for all $\{y_i\}$ can be fully written:

$$\text{PDF}(\boldsymbol{y}) = \mathscr{G}_n(\boldsymbol{x}; \boldsymbol{\mu}, K) = \frac{1}{\sqrt{\det(2\pi K)}} \exp\left(\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}} K^{-1} (\boldsymbol{x}-\boldsymbol{\mu})\right) \equiv \mathscr{N}(\boldsymbol{\mu}, K)$$

The game is then to predict the one dimensional gaussian PDF of an additional measurement to come y^* at a point x^*, knowing all the previous measurements (training samples) and assuming how each measurement at x will impact the measurement in x' (encoded by the kernel).

By expliciting the two sub-spaces, one generated by $\boldsymbol{x}$ (and $\boldsymbol{y}$) and the other generated by the measurement to be predicted $\boldsymbol{x}^*$ (and $\boldsymbol{y}^*$), it becomes easy to write the joint PDF of $(\boldsymbol{y}, \boldsymbol{y}^*)$:

$$\text{PDF}\left(\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{y}^* \end{bmatrix}\right) = \mathscr{N}\left(\boldsymbol{\mu}, \begin{bmatrix} \mathscr{K}(\boldsymbol{x}, \boldsymbol{x}) & \mathscr{K}(\boldsymbol{x}, \boldsymbol{x}^*) \\ \mathscr{K}(\boldsymbol{x}^*, \boldsymbol{x}) & \mathscr{K}(\boldsymbol{x}^*, \boldsymbol{x}^*) \end{bmatrix}\right)$$

which leads to the one dimensional gaussian PDF for the new value to be predicted $\text{PDF}(\boldsymbol{y}^*) = \mathscr{N}(\mu^*, \Sigma)$ with μ^* and Σ are perfectly detmined by the kernel, the previous measurements ($\boldsymbol{x}$ and $\boldsymbol{y}$) and the points where new measurements need to be predicted $\boldsymbol{x}^*$:

$$\boldsymbol{\mu}^* = \mathscr{K}(\boldsymbol{x}^*, \boldsymbol{x})^{-1} \mathscr{K}(\boldsymbol{x}, \boldsymbol{x})\, \boldsymbol{y}$$

$$\Sigma = \mathscr{K}(\boldsymbol{x}^*, \boldsymbol{x}^*) - \mathscr{K}(\boldsymbol{x}^*, \boldsymbol{x}) \mathscr{K}(\boldsymbol{x}, \boldsymbol{x})^{-1} \mathscr{K}(\boldsymbol{x}, \boldsymbol{x}^*)$$

1.2 Importation of the usual packages

```
import numpy as np
import matplotlib.pyplot as plt
```

```
import matplotlib        as mpl
%matplotlib inline

# Plot settings
mpl.rcParams['legend.frameon' ] = False
mpl.rcParams['legend.fontsize'] = 'xx-large'
mpl.rcParams['xtick.labelsize'] = 16
mpl.rcParams['ytick.labelsize'] = 16
mpl.rcParams['axes.titlesize' ] = 18
mpl.rcParams['axes.labelsize' ] = 18
mpl.rcParams['lines.linewidth'] = 2.5

# Define global paramter to build fake data
n=100
xmin=0
xmax=4
```

1.3 Definition of the kernel

The kernel specifies how two points in the feature space x and x' will be influence each other. There are several possibilities depending on which type or probleme need to be solve. The next piece of code define a function which plot the 1D and 2D plot of the correlation matrix obtained for any kernel function.

```
# Plot Kernel function
def plot_kernel_function(x, Kij, kernel_name=''):
    plt.figure(figsize=(15,5))
    plt.subplot(1,2,1)
    plt.title('1D kernel'+kernel_name)
    plt.plot( x, Kij[:,int(n/2)] , 'o-' )
    plt.subplot(1,2,2)
    plt.title('2D kernel'+kernel_name)
    plt.imshow( Kij , extent=[xmin,xmax,xmin,xmax], aspect='auto',
 ↪  origin='lower')
    plt.colorbar()
    plt.tight_layout()
    return
```

Four examples of usual kernel are implemented and defined below:

$$\text{Radial-basis function} : \mathscr{K}(x_i, x_j) = \exp\left(\frac{(x_i - x_j)^2}{2\ell^2}\right)$$

$$\text{Rational Distance} : \mathscr{K}(x_i, x_j) = \left(1 + \frac{(x_i - x_j)^2}{\alpha \ell^2}\right)^{-\alpha}$$

$$\text{Dot product} : \mathscr{K}(x_i, x_j) = \sigma + x_i \cdot x_j$$

$$\text{Exponential sinus squared} : \mathscr{K}(x_i, x_j) = \exp\left(-2 \times \sin\left(\frac{\pi}{p}\frac{x_i - x_j}{\ell}\right)^2\right)$$

```
# Exponential square distance kernel
def kernel_ESD(a, b, param):
    sqdist = np.sum(a**2,1).reshape(-1,1) + np.sum(b**2,1) - 2*np.dot(a, b.T)
    return np.exp(-.5 * (1/param**2) * sqdist)

# Define the rational quadratic kernel
def kernel_RQ(a, b, param):
    alpha=param[0]
    l=param[1]
    sqdist = np.sum(a**2,1).reshape(-1,1) + np.sum(b**2,1) - 2*np.dot(a, b.T)
    return np.power(1 + sqdist/(l*alpha), -1*alpha)

# Define the dot product kernel
def kernel_DP(a, b, param):
    return param + np.dot(a,b.T)

# Define the Exp-Sine-Squared kernel
def kernel_ESS(a, b, param):
    p=param[0]
    l=param[1]
    sqdist = np.sum(a**2,1).reshape(-1,1) + np.sum(b**2,1) - 2*np.dot(a, b.T)
    phi    = np.pi/p * sqdist/l
    return  np.exp( -2 * np.sin(phi)**2  )
```

```
x=np.linspace(xmin,xmax,n).reshape(-1,1)
plot_kernel_function(x, kernel_ESD(x,x,0.5)      , ' ESD ($\ell=0.5$)'    )
plot_kernel_function(x, kernel_RQ(x,x,[0.1,0.5]) , ' RQ ($0.1,\ell=0.5$)' )
plot_kernel_function(x, kernel_DP(x,x,5)         , ' DP ($\sigma=0.1$)'   )
plot_kernel_function(x, kernel_ESS(x,x,[5,1.0])  , ' ESS ($p=2,\ell=1.0$)')
```

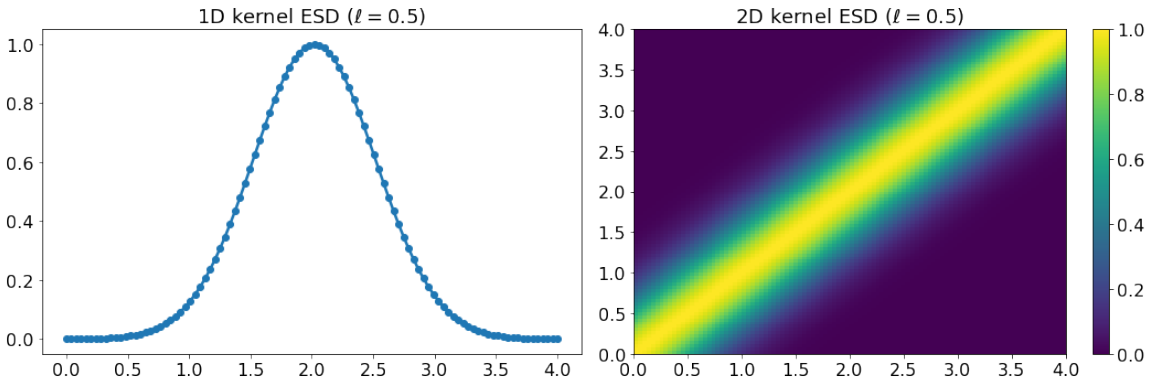

Figure 1: png

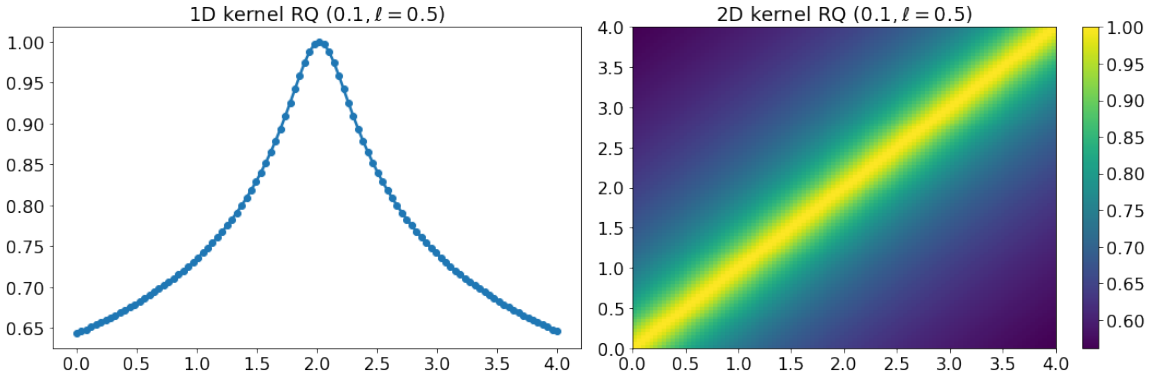

Figure 2: png

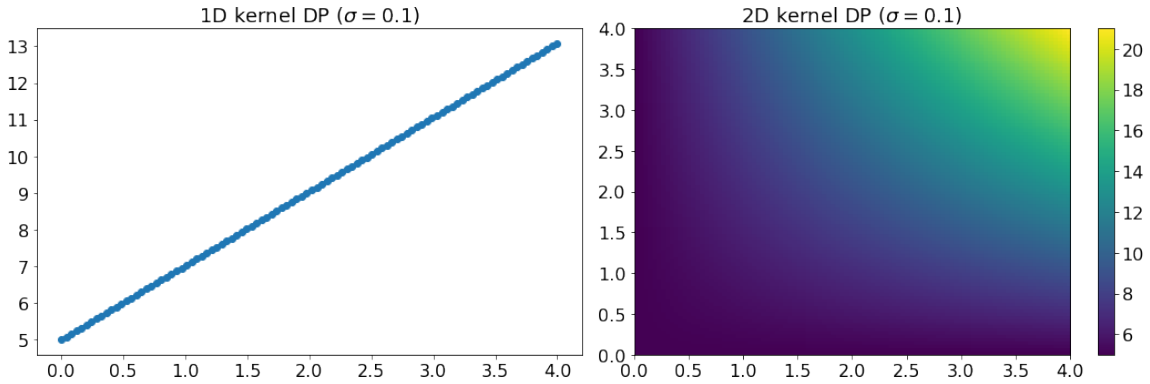

Figure 3: png

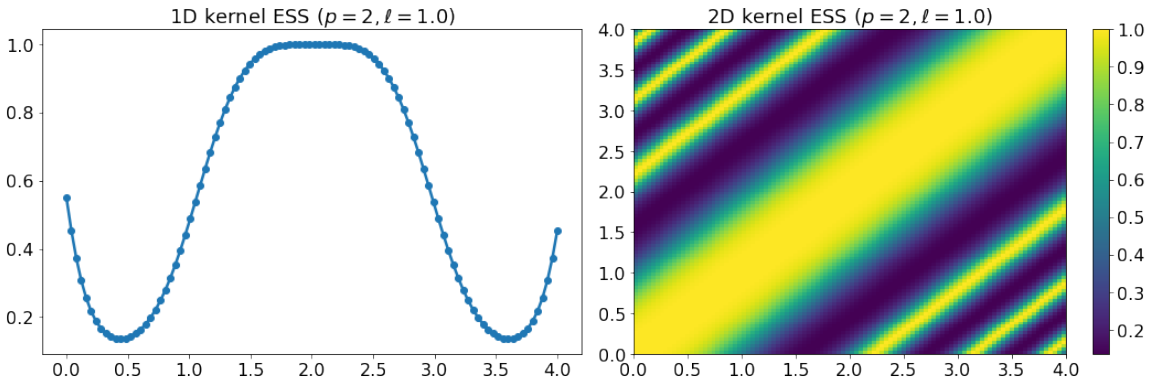

Figure 4: png

1.4 Priors functions

What happen if the procedure is performed but without any measurements? We can investigate the typical shape of the function using randomly distributed y_i (with a gaussian PDF). This is exactly what is done below: this shows the prior function that will be refined once y_i values comes in. The following code shows how the prior functions look like for the four different kernels discussed before.

```
# Define 3 sets of random yi
yi_random = np.random.normal(size=(n,3))

def plot_gp_priors(_xtest,_kernel_function, param):

    # Compute Kernel
    K_11 = _kernel_function(_xtest, _xtest, param)

    # Get cholesky decomposition (square root) of the covariance matrix
    L = np.linalg.cholesky( K_11 + 1e-12*np.eye(len(_xtest)) )

    # Define 5 priors based on gaussian-distributed {yi}
    f_prior = np.dot(L, yi_random)

    # Let's plot 3 sampled functions.
    plt.figure(figsize=(15,5))
    plt.plot(_xtest, f_prior)
    plt.xlim(np.min(_xtest),np.max(_xtest))
    plt.title('GP priors')
    plt.tight_layout()

    return
```

```
# Plot priors using test data
xtest = np.linspace(xmin, xmax, n).reshape(-1,1)
plot_gp_priors(xtest,kernel_ESD,0.5)
plot_gp_priors(xtest,kernel_RQ,[0.1,0.5])
plot_gp_priors(xtest,kernel_DP,5)
#plot_gp_priors(xtest,kernel_ESS,[8,4.0])
```

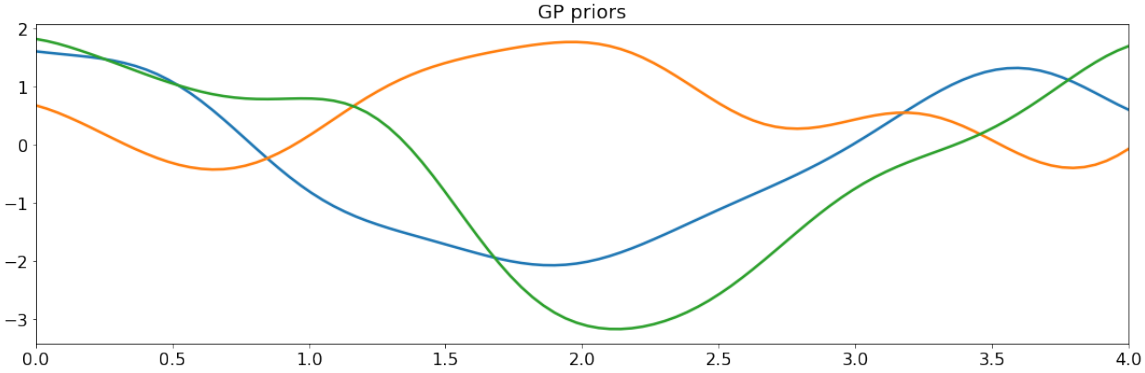

Figure 5: png

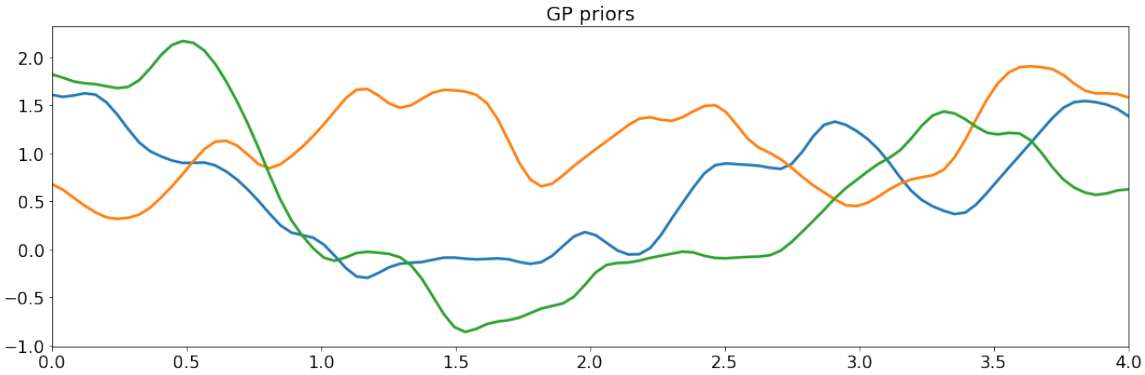

Figure 6: png

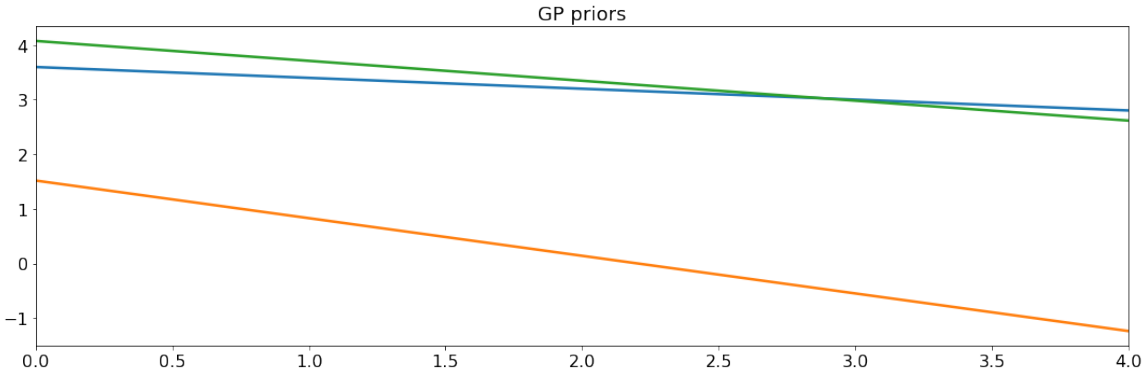

Figure 7: png

```
# Underlying function
def truth_function(x):
    return 2*x*np.sin(x)

# Noiseless training data
Ndata=5
xtrain = np.sort(0.5*(xmax-xmin)*np.random.rand(Ndata)+xmin).reshape(Ndata,1)
ytrain = truth_function(xtrain)
```

```
def perform_gp_prediction(_xtrain, _ytrain, _kernel_function, _param):

    # Apply the kernel function to our training points
    K = _kernel_function(_xtrain, _xtrain, _param)
    L = np.linalg.cholesky( K + 1e-12*np.eye(len(_xtrain)) )

    # Compute the mean at our test points.
    K_s = _kernel_function(_xtrain, xtest, _param)
    Lk = np.linalg.solve(L, K_s)
    mu = np.dot(Lk.T, np.linalg.solve(L, _ytrain)).reshape((n,))

    # Compute the standard deviation so we can plot it
    K_ss = _kernel_function(xtest, xtest, _param)
    s2 = np.diag(K_ss) - np.sum(Lk**2, axis=0)
    stdv = np.sqrt(s2)

    # Get function from the posterior at our test points.
    L = np.linalg.cholesky( K_ss + 1e-12*np.eye(n) - np.dot(Lk.T, Lk) )
    f_post = mu.reshape(-1,1) + np.dot(L, np.random.normal(size=(n,2)))

    # Plots things
    plt.figure(figsize=(12,6))
```

```
    plt.plot(np.linspace(xmin,xmax,500),truth_function(np.linspace(xmin,xmax,500)),
    label='truth' )
    for i in range(len(f_post[1])):
        plt.plot(xtest, f_post[:,i], label='posterior n{:.0f}'.format(i))
    plt.gca().fill_between(xtest.flat, mu-2*stdv, mu+2*stdv, color="#dddddd",
    label='$\pm \, 1 \, \sigma$')
    plt.plot(xtest, mu, 'r--', lw=3, label='$\mu(x)$')
    plt.plot(_xtrain, _ytrain,'o', color='black', ms=10, zorder=10,
    label='data')
    plt.axis([xmin, xmax, -8, 8])
    plt.title('Gaussian Process Interpolation')
    plt.legend()
    plt.tight_layout()

    return
```

```
perform_gp_prediction(xtrain,ytrain,kernel_ESD,2)
```

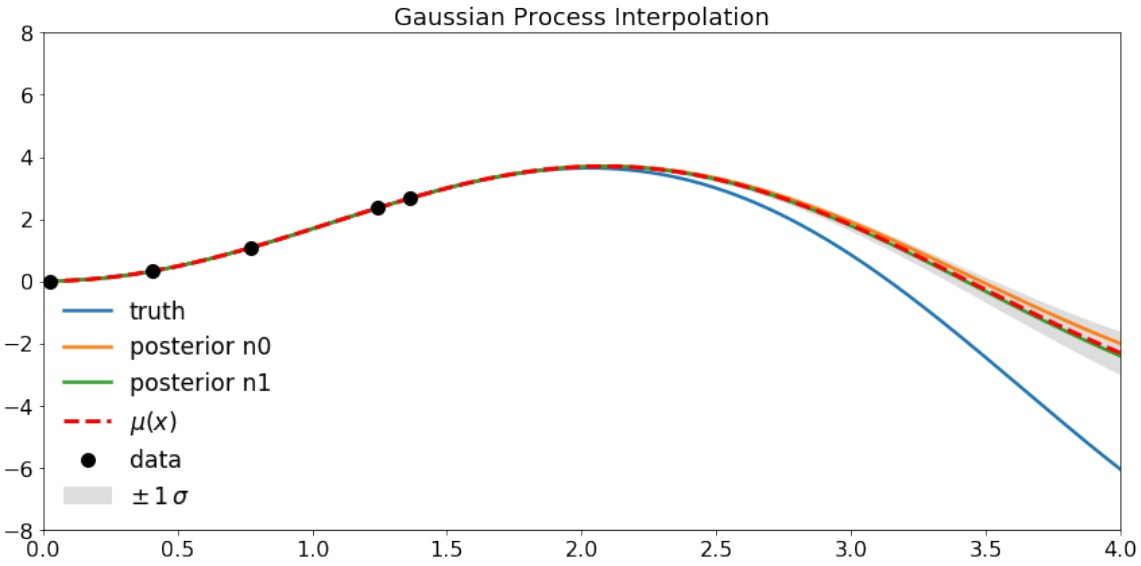

Figure 8: png

```
perform_gp_prediction(xtrain,ytrain,kernel_ESD,0.2)
```

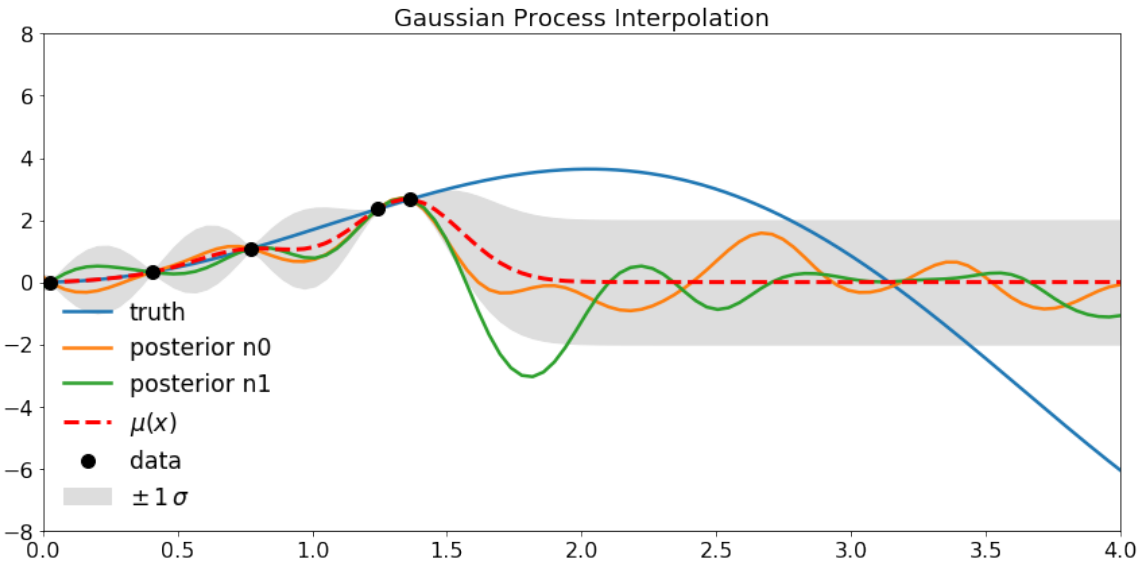

Figure 9: png

```
perform_gp_prediction(xtrain,ytrain,kernel_RQ,[4,0.1])
```

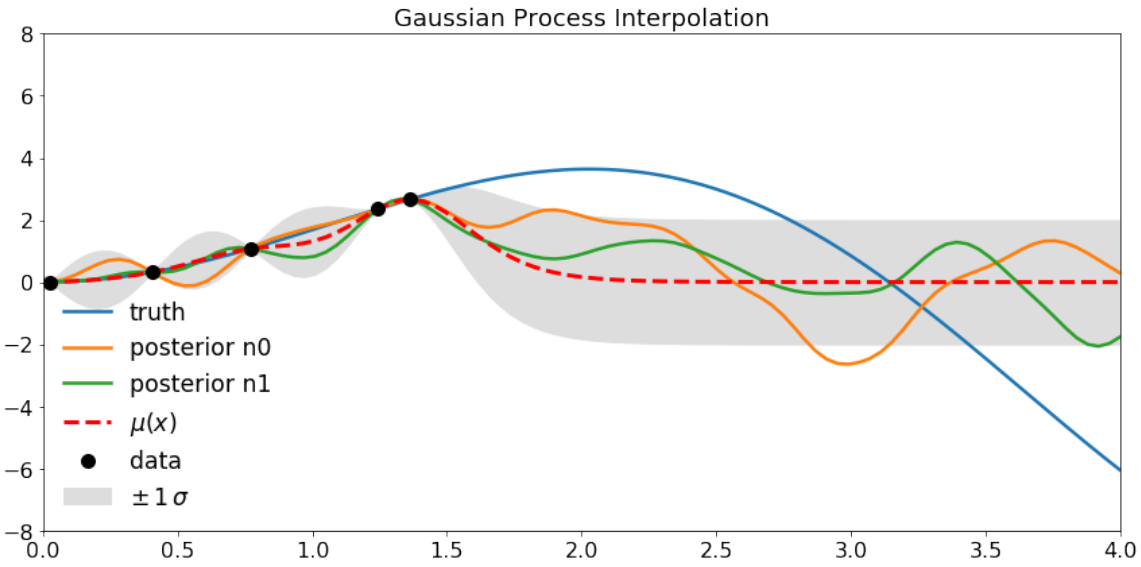

Figure 10: png

2 Dark Matter Coupled to Top Quark

2.1 General informations

The 2D plots need to be corrected. Right now, they seem to show the y-axis reversed ($y = -y$).

2.1.1 The model

We consider a simplified dark matter model with a new interaction field (mediator) is responsible for the weak interaction between the ordinary matter and the introduced dark matter candidate. The particular model we are interested in relies on a coupling between dark matter and the top quark, which was designed to lead to $t + E_T^{\text{miss}}$, so-called monotop final state: Revisiting monotop production at the LHC (JHEP01 (2015) 017). In the non-resonant model case of this publication (on which we will focus), the general lagrangian density is:

$$\mathscr{L}_{\text{DM}} = \mathscr{L}_{\text{kin}}[\chi, V_\mu] + V_\mu \left(a_L^{ij}\, \bar{u}_{i,L} \gamma^\mu u_{j,L} + a_R^{ij}\, \bar{u}_{i,R} \gamma^\mu u_{j,R} \right) + V_\mu \left(g_L\, \bar{\chi}_L \gamma^\mu \chi_L + g_R\, \bar{\chi}_R \gamma^\mu \chi_R \right)$$

where the matrices a_L^{ij} and a_R^{ij} describe the couplings in the flavour space for left-handed and right-handed components of quarks. The interaction between the dark matter and the mediator are determined by g_L and g_R. These parameters have a priori a real and an imaginary part.

2.1.2 Simplifications

In order to simplify the model, we restrict the possible couplings only between the first and third generation. In addition, in order to evade b-physics constraints we assume only right-handed couplings. Indeed, an interaction term with the left-handed components would automatically involve the b-quark because of the SU(2)_L symmetry, which is strongly excluded by current data. Finally, we assume vectorial interaction with the dark matter so that $g_R = g_L$. Within these assumptions, the dynamic of the new fields is described by the following lagrangian density:

$$\mathscr{L}_{\text{DM}} = \mathscr{L}_{\text{kin}}[\chi, V_\mu] + g_{\text{SM}}\, V_\mu\, \bar{t}_R \gamma^\mu u_R + g_{\text{DM}}\, V_\mu\, \bar{\chi} \gamma^\mu \chi$$

where $g_{\text{SM}} \equiv a_R^{13}$ and $g_{\text{DM}} \equiv g_R = g_L$. In this context, the model is fully predictive once four parameters are fixed: $(m_V, m_\chi, g_{\text{SM}}, g_{\text{DM}})$. The new interations leads to the two additional vertices $V t \bar{u}$ and $V \chi \bar{\chi}$ with respect to the SM.

2.1.3 Relevant Processes and Final States

There are two experimentally interesting final states to probe this model: $t + E_T^{\text{miss}}$ and $tt + X$. The first one is obtained via the process $gu \to tV \to t\chi\bar{\chi}$ while the second one can be obtained via $gu \to tV \to tt\bar{u}$ (together with two other diagrams like t-channel exchange of V in $uu \to tt$). This makes the two final states complementary since they allow together to probe the model of any value of BR_χ.

2.1.4 Analytical Expressions

Depending on the parameter values, the signature can be then quite different and it's important to properly model these features, via for e.g. BR_χ. Also, for a given final state, the width of the mediator can also lead to differences in event kinematics which also need to be accounted for. The following analytical expression give the relation between the mediator properties and the model paramters at the leading order.

$$\Gamma_V = \Gamma_{V\to\chi\chi} + \Gamma_{V\to t\bar{u}+\bar{t}u} \quad ; \quad \mathrm{BR}_\chi = \frac{\Gamma_{V\to\chi\chi}}{\Gamma_V}$$

$$\Gamma_{V\to\chi\chi} = \frac{m_V}{12\pi} g_{\mathrm{DM}}^2 \sqrt{1-4\frac{m_\chi^2}{m_V^2}} \left(1+2\frac{m_\chi^2}{m_V^2}\right) \quad ; \quad \Gamma_{V\to t\bar{u}+\bar{t}u} = \frac{m_V}{\pi} g_{\mathrm{SM}}^2 \left(1-\frac{m_t^2}{m_V^2}\right)\left(1-\frac{m_t^2}{2m_V^2}-\frac{m_t^4}{2m_V^4}\right)$$

2.2 Numerical analysis

2.2.1 General informations

A python module, ModelParaRelation, was written in order play with the analytical formulas above, the main goal being to have a feeling of how the mediator properties are related to the model parameters. The total width as a function of mediator mass is plotted and then the evolution of the invisible BR with respect to coupling constants is also looked at. Finally, we check how the couplings constants evolves as a function of the BR.

```
import numpy             as np
import matplotlib.pyplot as plt
import matplotlib        as mpl
import ModelParaRelation as model
%matplotlib inline
mV=1000; mDM=1;

# Ignore warning related to undefined division
np.seterr(divide='ignore', invalid='ignore')

# Plot settings
mpl.rcParams['legend.frameon' ] = False
mpl.rcParams['legend.fontsize'] = 'xx-large'
mpl.rcParams['xtick.labelsize'] = 16
mpl.rcParams['ytick.labelsize'] = 16
mpl.rcParams['axes.titlesize' ] = 18
```

```
mpl.rcParams['axes.labelsize' ] = 18
mpl.rcParams['lines.linewidth'] = 2.5
```

2.2.2 Mediator width as function of its mass

```
gDM = np.linspace(0.1,1.6,3)
gSM = np.linspace(0.1,1.6,3)
g2D = np.array( np.meshgrid(gSM,gDM) ) # 2D grid but 3*3 array
g2D = g2D.T                            # 2D grid but 3*3 transposed array
g2D = g2D.reshape(-1,2)                # 2D grid but 2*6 array --> what we
 ↪  want

# Mediator masses
m1 = np.linspace(model.mt/5, model.mt*10,1000)
m2 = np.linspace(500, 5000,1000)

# Plots
plt.figure(figsize=(14,6))
Title='Relative mediator width ; $m_{DM}=$' + '{:.0f} GeV'.format(mDM)

plt.subplot(121)
plt.loglog()
plt.title(Title)
plt.ylim(1e-4, 1e6)
plt.ylabel('$\Gamma_V \,/\, m_V$')
plt.xlabel('$m_V$ [GeV]')
for g in g2D:
    gSM=g[0];gDM=g[1]
    leg_label='$(g_{SM},g_{DM})=$'+'({:1.1f},{:1.1f})'.format(gSM,gDM)
    plt.plot(m1,model.get_total_width(gSM,gDM,m1,mDM)/m1, label=leg_label)
plt.legend(fontsize='11')

plt.subplot(122)
plt.loglog()
plt.title(Title)
plt.ylabel('$\Gamma_V \,/\, m_V$')
plt.xlabel('$m_V$ [GeV]')
plt.xlim(400, 3e4)
for g in g2D:
    gSM=g[0];gDM=g[1]
    leg_label='$(g_{SM},g_{DM})=$'+'({:1.1f},{:1.1f})'.format(gSM,gDM)
    plt.plot(m2,model.get_total_width(gSM,gDM,m2,mDM)/m2, label=leg_label)
```

```
plt.legend(fontsize='11')

plt.tight_layout()
plt.savefig('TotalWidth_vs_mV.pdf')
```

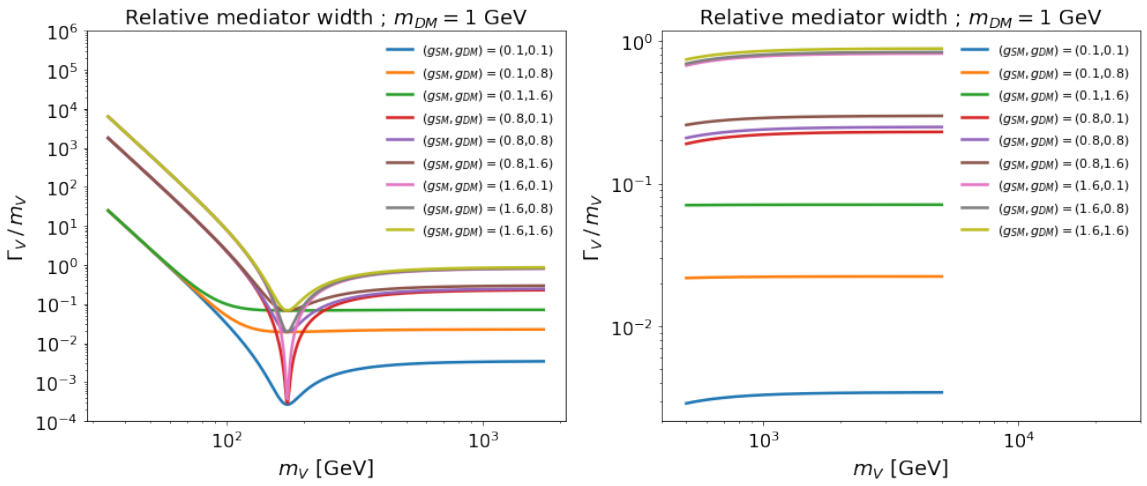

Figure 11: png

2.2.3 Mediator width and invisible BR: function SM coupling

```
gDM=[0.5,1,1.5]
g = np.linspace(0,1.5,150)

plotTitle='$m_V=$'+'{:.0f} TeV'.format(mV/1000.) + ' ; $m_{DM}=$' + '{:.0f}
↪  GeV'.format(mDM)

plt.figure(figsize=(14,6))
plt.subplot(121)
plt.title( plotTitle )
plt.xlabel('$g_{SM}$')
plt.ylabel('$\Gamma_V$ [GeV]')
for gdm in gDM: plt.plot( g, model.get_total_width(g,gdm,mV,mDM) ,
↪  label='$g_{DM}=$'+'{:1.1f}'.format(gdm))
plt.legend()

plt.subplot(122)
plt.title( plotTitle )
for gdm in gDM: plt.plot( g, model.get_BR(g,gdm,mV,mDM) ,
↪  label='$g_{DM}=$'+'{:1.1f}'.format(gdm))
```

```
plt.xlabel('$g_{SM}$')
plt.ylabel('BR$_{\chi}$')
plt.legend()

plt.tight_layout()
plt.savefig('SMcouplingsDep.pdf')
```

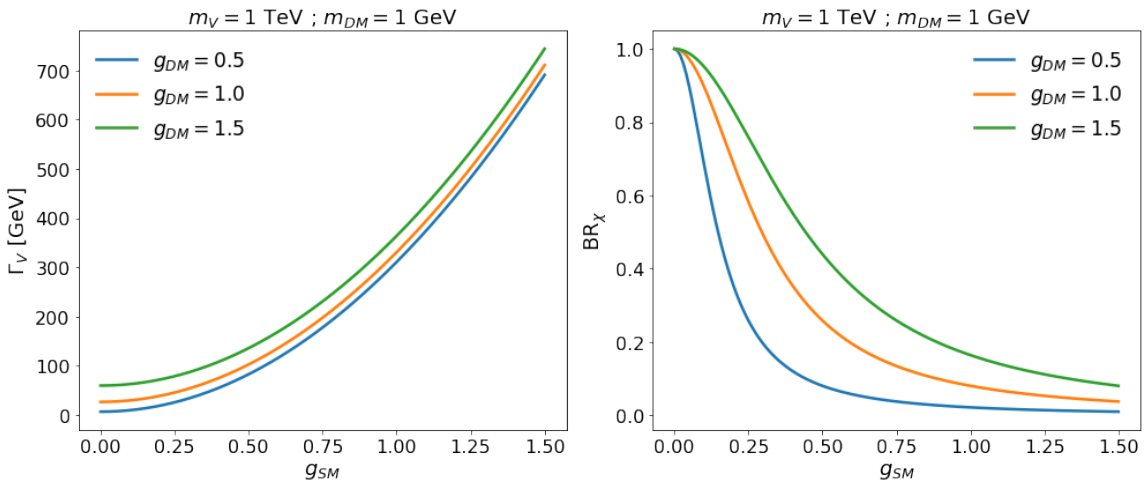

Figure 12: png

2.2.4 Mediator width and invisible BR: function of both couplings

```
gSM     = np.linspace(0,1.5,150)
gDM     = np.linspace(0,1.5,150)
gSM,gDM = np.meshgrid(gSM,gDM)
BR      = model.get_BR(gSM,gDM,mV,mDM)
Gamma   = model.get_total_width(gSM,gDM,mV,mDM)

plt.figure(figsize=(14,6))
plotTitle='$m_V=$'+'{:.0f} TeV'.format(mV/1000.) + ' ; $m_{DM}=$' + '{:.0f}
↪  GeV'.format(mDM)

plt.subplot(121)
#plt.pcolor(gSM, gDM, Gamma, cmap='Blues') --> This makes some blank line
↪  because of the meshgrid
plt.imshow(Gamma, interpolation='none', cmap='Blues',          # choose
↪  Z(x,y), interpolation and color
           extent=[gSM.min(), gSM.max(), gDM.min(), gDM.max()], # give proper
↪  range for x/y axis
```

```
            aspect='auto')                                          # fill the
 ↪  whole figure
plt.title('$\Gamma_V$ [GeV] ; ' + plotTitle)
plt.xlabel('$g_{SM}$')
plt.ylabel('$g_{DM}$')
plt.colorbar()

plt.subplot(122)
plt.imshow(BR, interpolation='none', cmap='Blues',
           extent=[gSM.min(), gSM.max(), gDM.min(), gDM.max()],
           aspect='auto' )
plt.title('BR$_{\chi}$ ; ' + plotTitle)
plt.xlabel('$g_{SM}$')
plt.ylabel('$g_{DM}$')
plt.colorbar()

plt.tight_layout()
plt.savefig('BRandGammavsCouplings.pdf')
```

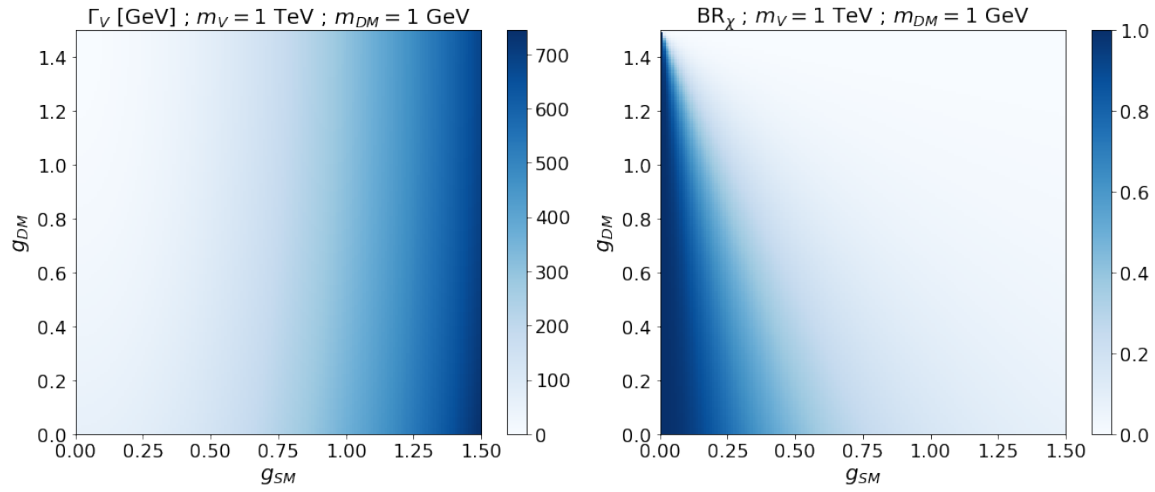

Figure 13: png

2.2.5 Mediator couplings as function of invisible BR and width

```
mV=1000; mDM=1;
GV    = np.linspace(0.0,500,150)
BR    = np.linspace(0.0,1.0,150)
GV,BR = np.meshgrid(GV,BR)
g_SM  = model.get_gSM_from_BRwidth(BR,GV,mV,mDM)
```

```
g_DM  = model.get_gDM_from_BRwidth(BR,GV,mV,mDM)


plt.figure(figsize=(14,6))
plotTitle='$m_V=$'+'{:.0f} TeV'.format(mV/1000.) + ' ; $m_{DM}=$' + '{:.0f}
↪  GeV'.format(mDM)

plt.subplot(121)
plt.imshow(g_SM, interpolation='none', cmap='Blues',
           extent=[GV.min(), GV.max(), BR.min(), BR.max()],
           aspect='auto')
plt.title('$g_{SM}$ ; ' + plotTitle)
plt.xlabel('$\Gamma_{V}$ [GeV]')
plt.ylabel('BR$_{\chi}$')
plt.colorbar()

plt.subplot(122)
plt.xlim(GV.min(), GV.max())
plt.ylim(BR.min(), BR.max())
plt.imshow(g_DM, interpolation='none', cmap='Blues',
           extent=[GV.min(), GV.max(), BR.min(), BR.max()],
           aspect='auto')
plt.title('$g_{DM}$ ; ' + plotTitle)
plt.xlabel('$\Gamma_{V}$ [GeV]')
plt.ylabel('BR$_{\chi}$')
plt.colorbar()

plt.tight_layout()
plt.savefig('BRandGammavsCouplings.pdf')
```

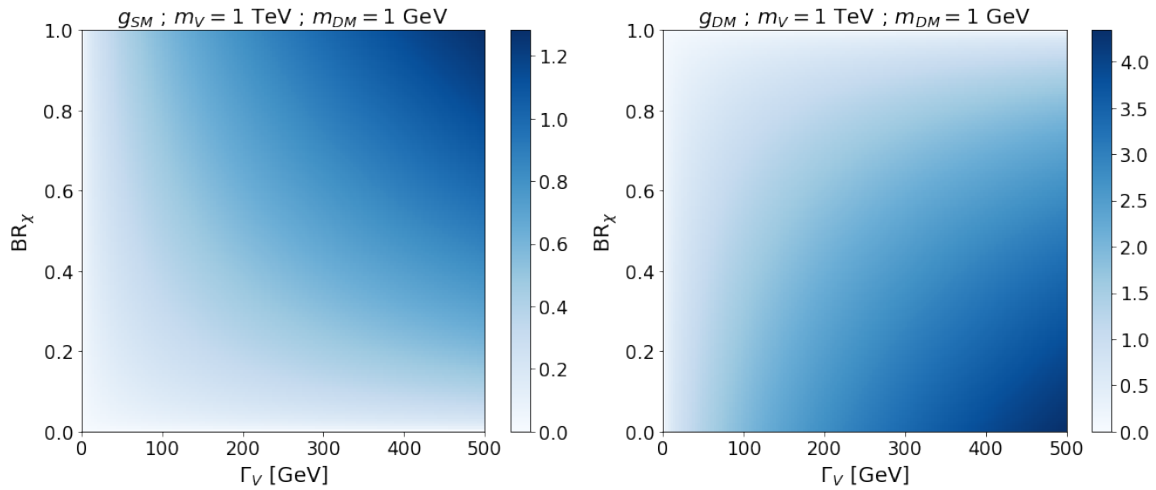

Figure 14: png