In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
import torch
import torch.nn.functional as F
from torch.autograd.functional import hessian
from torch.distributions.multivariate_normal import MultivariateNormal
import seaborn as sns
import io
import base64
```

Instruções gerais: Sua submissão deve conter:

1. Um "ipynb" com seu código e as soluções dos problemas
2. Uma versão pdf do ipynb

Caso você opte por resolver as questões de "papel e caneta" em um editor de $\LaTeX$ externo, o inclua no final da versão pdf do 'ipynb'--- submetendo um único pdf.

Trabalho de casa 05: Processos Gaussianos para regressão

1. Durante a aula, discutimos como construir uma priori GP e o formato da posteriori preditiva para problemas de regressão com verossimilhança Gaussiana (com média definida pelo GP). O código abaixo cria um GP com kernel exponencial quadrático, mostra a priori preditiva e a posteriori preditiva. Experimente com o código e comente a influência de ambos os parâmetros do kernel exponencial quadrático, tanto na priori preditiva quanto na posteriori preditiva. Nos gráficos gerados, os pontos vermelhos são observações, as curvas sólidas azuis são a médias das preditivas e o sombreado denota +- um desvio padrão.

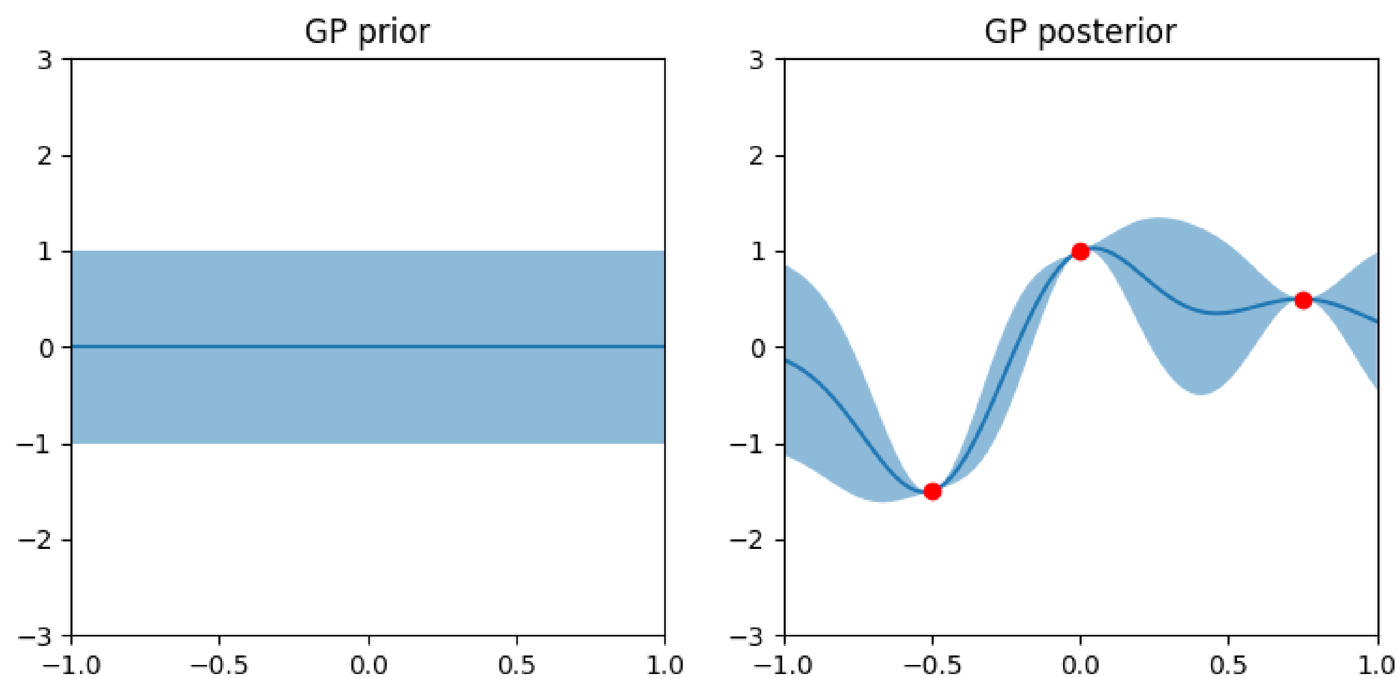
In [2]:

```
SEED = 42
np.random.seed(SEED)

s2 = 1e-04 # variância observacional

def rbf_kernel(x1, x2, gamma=10.0, c=1.0):
    assert(gamma>0)
    assert(c>0)
    return (-gamma*(torch.cdist(x1, x2)**2)).exp()*c

x = torch.linspace(-1, 1, 100)[:, None]

K = rbf_kernel(x, x) + torch.eye(x.shape[0])*s2
mu = torch.zeros_like(x)

fig, axs = plt.subplots(1, 2, figsize=(9, 4))

axs[0].plot(x, mu)
axs[0].fill_between(x.flatten(), mu.flatten()-K.diag(), mu.flatten()+K.diag(), alpha=0.5)
axs[0].set_xlim([-1, 1])
axs[0].set_ylim([-3, 3])
axs[0].set_title('GP prior')


xtrain = torch.tensor([-0.5, 0.0, 0.75])[:, None]
ytrain = torch.tensor([-1.5, 1.0, 0.5])[:, None]

def posterior_pred(x, xt, yt, gamma=10.0, c=1.0):
```

```
    Kxxt = rbf_kernel(x, xt, gamma, c)
    Kxt = rbf_kernel(xt, xt, gamma, c) + torch.eye(xt.shape[0])*s2
    Kinv = torch.linalg.inv(Kxt)
    Kxx = rbf_kernel(x, x, gamma, c)

    mu = Kxxt @ Kinv @ yt
    cov = Kxx - Kxxt @ Kinv @ Kxxt.T
    return mu, cov

post_mu, post_cov = posterior_pred(x, xtrain, ytrain)
axs[1].plot(x, post_mu)
axs[1].fill_between(x.flatten(), post_mu.flatten()-post_cov.diag(), post_mu.flatten()+post
axs[1].scatter(xtrain, ytrain, color='red', zorder=5)

axs[1].set_xlim([-1, 1])
axs[1].set_ylim([-3, 3])
axs[1].set_title('GP posterior')
```

Out[2]:

```
Text(0.5, 1.0, 'GP posterior')
```

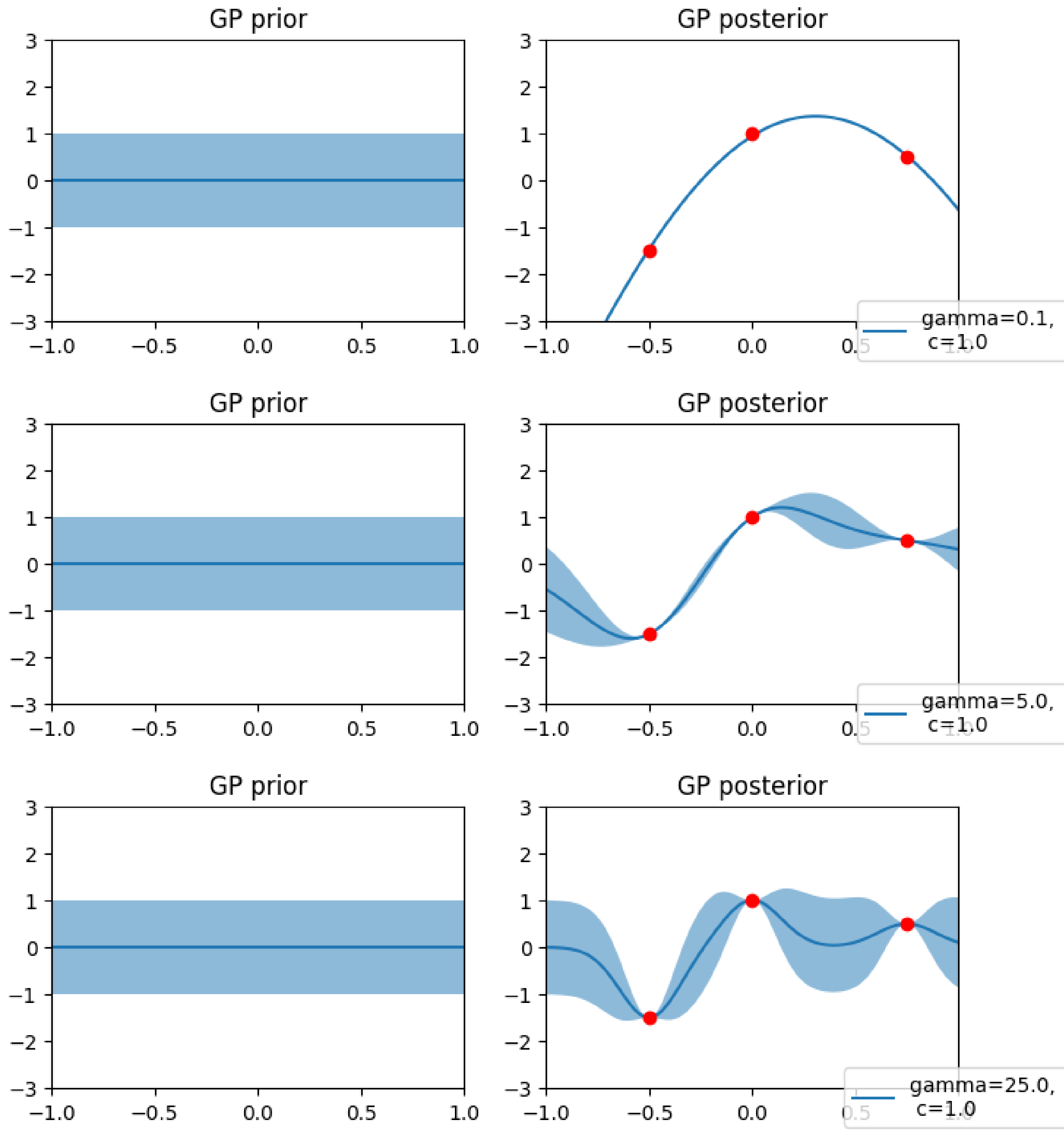


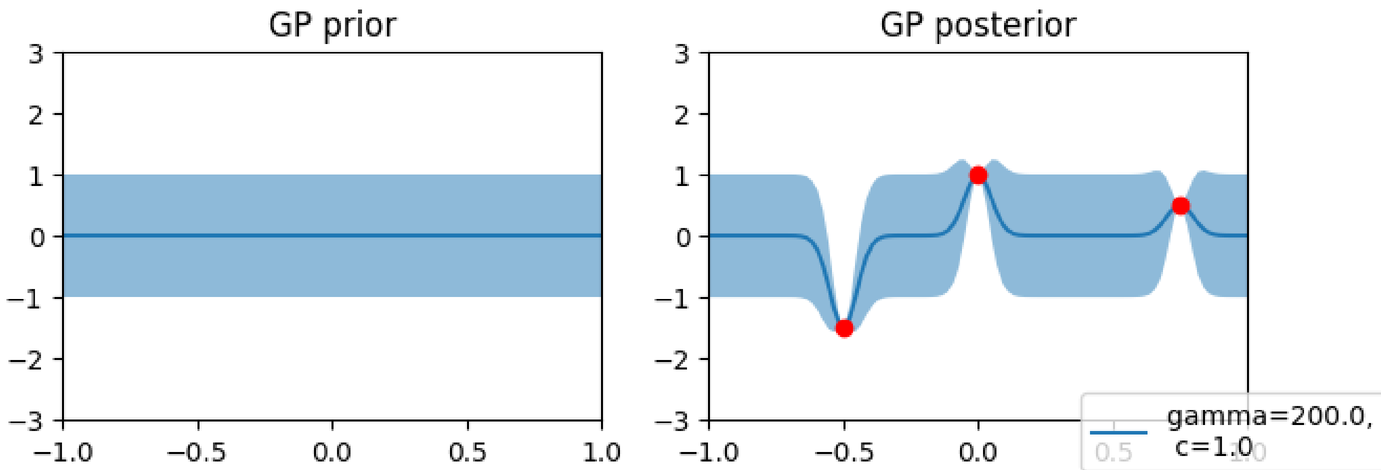
Vamos testar o código com diferentes valores de $gamma$ mas c fixo

In [13]:

```
for gamma, c in [(0.1, 1.0), (5, 1.0), (25.0, 1), (200.0,1.0)]:
    fig, axs = plt.subplots(1, 2, figsize=(8, 2.5))

    K = rbf_kernel(x, x, gamma, c) + torch.eye(x.shape[0])*s2
    mu = torch.zeros_like(x)

    axs[0].plot(x, mu)
    axs[0].fill_between(x.flatten(), mu.flatten()-K.diag(), mu.flatten()+K.diag(), alpha=(
    axs[0].set_xlim([-1, 1])
    axs[0].set_ylim([-3, 3])
    axs[0].set_title('GP prior')

    post_mu, post_cov = posterior_pred(x, xtrain, ytrain, gamma, c)
    axs[1].plot(x, post_mu)
    axs[1].fill_between(x.flatten(), post_mu.flatten()-post_cov.diag(), post_mu.flatten()+
    axs[1].scatter(xtrain, ytrain, color='red', zorder=5)

    axs[1].set_xlim([-1, 1])
    axs[1].set_ylim([-3, 3])
    axs[1].set_title('GP posterior')
```

```python
    # make legend with gamma and c
    fig.legend(['gamma={:.1f}, \n c={:.1f}'.format(gamma, c)], loc='lower right')
```



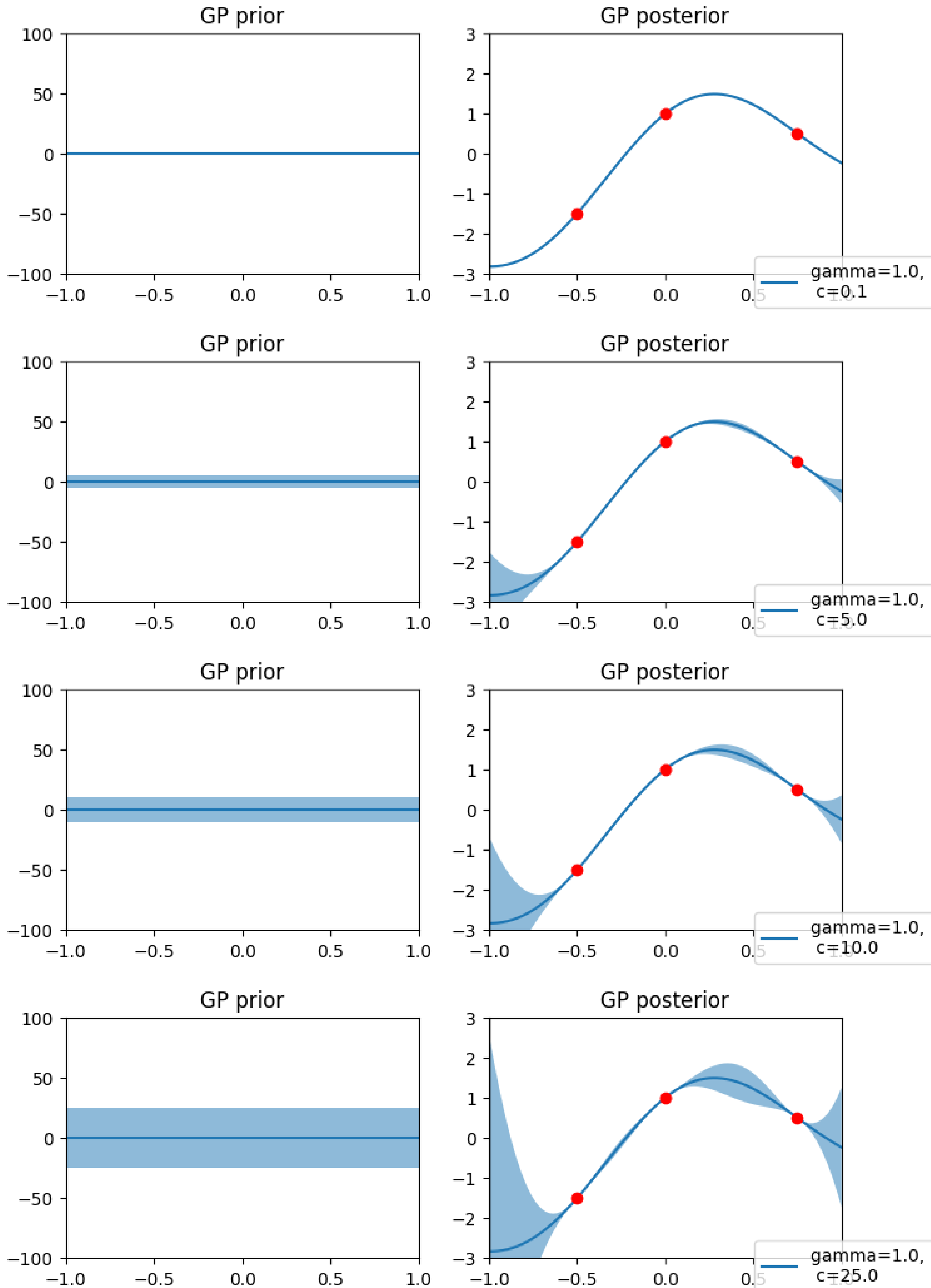


POdemos observar que alterar o valor de $gamma$ na função kernell impacta significativamente na suavidade da posteriori, tanto na curva em si quanto nos sombreados. Ao alterar o valor de $gamma$ para valores muito altos temos que a curva antes de 0 é praticamente uma reta,mudando apenas na proximidade dos ddos para se ajustar aos mesmos. Em contrapartida, valores de $gamma$ pequenos demais fazem com que a curva se adapte aos pontos a nível de desvio padrão desprezível, isto é, o sombreado praticamente desaparece. Notavelmente, alterar o valor de $gamma$ não altera o gráfico da priori.

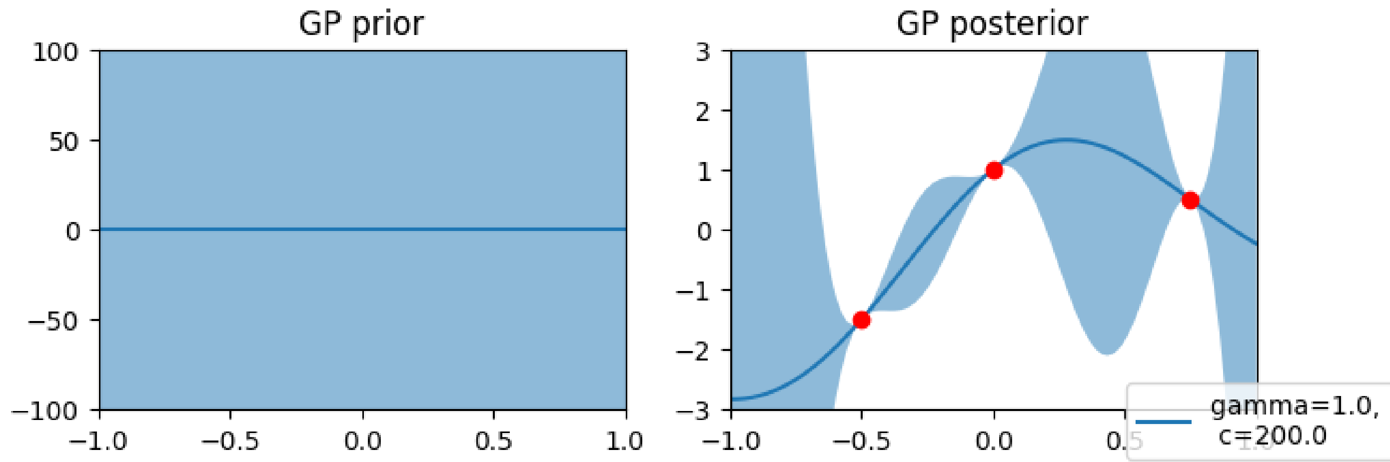
Agora, executando o código para diferentes valores de c mas $gamma$ fixo

In [17]:

```
for gamma, c in [(1.0, 0.1), (1, 5), (1, 10), (1,25), (1, 200.0)]:
    fig, axs = plt.subplots(1, 2, figsize=(8, 2.5))

    K = rbf_kernel(x, x, gamma, c) + torch.eye(x.shape[0])*s2
    mu = torch.zeros_like(x)

    axs[0].plot(x, mu)
    axs[0].fill_between(x.flatten(), mu.flatten()-K.diag(), mu.flatten()+K.diag(), alpha=(
    axs[0].set_xlim([-1, 1])
    axs[0].set_ylim([-100, 100])
    axs[0].set_title('GP prior')

    post_mu, post_cov = posterior_pred(x, xtrain, ytrain, gamma, c)
    axs[1].plot(x, post_mu)
    axs[1].fill_between(x.flatten(), post_mu.flatten()-post_cov.diag(), post_mu.flatten()+
    axs[1].scatter(xtrain, ytrain, color='red', zorder=5)

    axs[1].set_xlim([-1, 1])
    axs[1].set_ylim([-3, 3])
    axs[1].set_title('GP posterior')

    # make legend with gamma and c
    fig.legend(['gamma={:.1f}, \n c={:.1f}'.format(gamma, c)], loc='lower right')
```



GP prior
GP posterior
gamma=1.0, c=0.1
gamma=1.0, c=5.0
gamma=1.0, c=10.0
gamma=1.0, c=25.0



Podemos agora perceber que alterar os valores de c impactam tanto na priori quanto na posteriori. N apriori, o impcto está sobre a área sombreada, o que induz a crer que C é intimimante relacionado ao desvio padrão. Na posteriori, o impacto é sobre a suavidade do sombreamento, onde valores de c muito altos geram menos suavidade do sombreado, enquanto valores muito baixos fazem com que o sombreado se ajuste aos pontos de forma muito brusca e precisa. Para intervalos da função distantes dos dados, o sombreado é largo e se anula ao atingir os dados observados.

2. Durante a aula, discutimos como escolher os hiper-parametros do nosso GP. Estime os parâmetros ótimos para os dados carregados abaixo (acredite, é isso que o código faz). Reporte a evidência obtida e faça um plot similar ao acima. Para o dado de teste, reporte a i) log verossimilhança e ii) o MSE com relação à média. Em caso de dúvidas, recorra a nota de aula e o link adicionado no eclass.

In [38]:

```
data = np.load(io.BytesIO(base64.b85decode(
```

```
    'P)h>@6aWAK2mk;8Apo$)IktlX005u^000XB6aaK`VQFq(ST1gGc>w?r0H6Z^000000D}Vn000000G-JFQ_}?;
    '2NI?X(A<!XAf&F9gxYZsB1plJNq0ihIRZpQ_$**23=qcCzMEScGRBv|9&{@)AhLg<_iyi4)xOld$%ox21(ae
    '@8X}92%ftnhkr)!{KI3SqaztHTO%1?F#czrlL<L1=1>jXSP9_`ra#1EFBN&dUp#}93U`}z=Uq7W<3m;jT?g?r
    '2>OV&sD>fT|N5`c9=}Oc-;Ux)Kh#5%KK0C}5=2f&UT^G<B|Itrr%Vx`M7!#hh-(FZfcVH+w$FYOzHvTfU7#>v
    '!v-$V<-Dtd`E%k=Xs3rE#nf`py+{YH_eyu}Zl1;=a)XkTsql7Za04@Q26fwFD3`OfxLm?bo9G(@bL+s2Gth#y
    '+KbUaeQpyT1dh$Edi<>I#kEak-MsA!kV)ShvPHNAZV}_Rs^}u9td17*yGFs8ad~gY_ar)B^!hCNgc^wsrl)g6
    'C;jap_g;LHbT)#a(VMt#9tJ9t)!vXuq6qn={f|sQnfIcF9an-Xxcex}zY}Csq9U0wi6sGLmP$Vj62*^s)Hnt9
    'frB|1=vY_4L7yh-ug^wd53_iglQs<lTeG<iat;hn&RT6P7JRVASl@0mg7!vvm?GPVu0}7Nrg9cDLWkVHE$fAr
    'w<vb^QyY$cXQ9a=2f&wBV0_}&4N|+@`oqK=);7=Hp|j1XuDMqk<e`D^gt6@A*VOpYsq*sTO9Z4Fo^1>!x}a9V
    'N(u`b=2&rY#1fFUM&Q*1;RC6!5@Yw1yXmo7=qaT7y4vhOHYq#!dE*%FTX3j)#0ZX}3~rwGKGbt4aW^j#Fnz!2
    'P)h>@6aWAK2mk;8Apk!*Q7DH4005u^000XB6aaK`VQFq(c`j~nc>w?r0H6Z^000000EYtr000000G*KSGaCjF
    'O?L<t%BCWTWlE8&iF)a%+|5NB+ODeRWn$XRgdeh1hY^vq1VIv~=?xX@wI;3Ezp&@q^L%_{k)NK9KC^=$Cgjn(
    'k}^ao+g+5Fl$EmW-u_G`gX-_YpypHmCvWQRlTzoBC(vOzFCY7KAgjDN%<$4duk3PdbIm0DDZgC&UhWh|#(pJ
    '&hcASfrB($S$FR@kY=?lb<(Gib8=If@uL-N#|GT}&W>R30q<aG>0{`8IsfaOu5M&p-|t>h-hh_+8izM_D3Rs(
    'GN9TR5<P>VBG&+cUW?hxvb3mHBNCk}<wU1Z@Y)r6?}N5sP(6F>sn|OTt97sDDBr1pNp>0ITsLF<iDvt`*d9pJ
    '&{rmIjyOC4H&!gSPBEW=x>BadcI?4#oQHRb_!=0WmIPe2tpQ>3cHzDwb91<-w4-mrAX?+aI~%#(5O<|y;$+(-
    '!8q88>=k7BUl9H67Y=0-4aWMYifd-Cg70`CAwAcC50l9;1I`9a=(4diTV{w;hBh51Yhc3eIlD)qg{lq<otU}-
    '7+##?dMTqFcmHBoT$auw@4{zb;h1oxmb@O}G>841YXz;dW9Y_OQJi}*gX%it0Qt5SoL+x2*<|m4wUDAs*C#(i
    'X)G^x&s^R}MAP?#!2`AZxUc-+T$SAjlDi8(c{&awpWPX@*G&f1A*5@R7iNq&xUpH?G6Y-eKBS821<Y?%vKP0%
    'wHnxLtM(DOAO6lwpHeclu;-FFotH2Ql$kZZv%GoC2`jxrXx2bq`#a}@p7#Qm^6{vaGlWM4>+XkiP57GV!8t@X
    'RM*14P)h>@6aWAK2mk;8Apn6>1aKG%001Bm000UA6aaK(b97%=E^csn0RRvHAP@im000007zzLY00000omcx~
    'sKiWZBW<J_t;k_fVx=^dEEYD)XUy5mFx$-O+4<SCqeGF%dtdkG{tMpchwVAs_jO;__xt^vmOS^;u&`H}dCl;e
    'V<IBW_XEdIo*rp^KYh~p=t%2xw?~440{tH99_TmQ@Be(Y^0Hq|EV*^>(jI3UnSRgb-Cx??$c5J5_-`6Hu}_B=
    'WbKQ8H*lTxZG3S9KU}cq@VYGxe0BL}5e3N&G{64)ZUgVI7j9tvj<QZK_HN)_>#7g#XyBUD(L>jrujkC#KBdQw
    '&($+4=$`$1J?oj=uk6)hx75?zeXX{R&yJ4%q}#zdc9~>0Tt~CY?D=&p`Ed8ssgvrs@r$IAql4;bR=d`<juY3n
    'hbw|d9-UiDv-;KXwVda*J0YNdEyvxKbk_*KT87w#@UTsf_W1FPhxKOre|Tt~c6FtPDdsWndpL7^@Z#ZPJbcl7
    'UANRQIo);J@Z=gsTf3O`ZViXzn*!Ic%b7<G<@c(g+08#~YFPYt&R<>5ySdR`$W61WS!r%Y`nUW2hlOt9A(wT]
    'HRsI8dFY1&)f_v|G_IOvw^tIY>67By)Ok`hJ?qcTdwx(gcW*NrtmdlyZ~yvz^J>Nx^?vGbMHOGLcK>%y6?<O7
    'w1$g$9jiFi9*9Z~8TfXM|Cvhqo3{Q@$!pfB;#O9&$drFpC4ZYezi-~?N+y{yJzB}A6>X#Z+f_2u9OrTc%_(Dx
    'd~aB<3I<!Jj%`yxpS#<B``x*6di+g6%V|!3X+t?zVe)I87M3%1STCzo<s31_+*r=QH+J0nU0^v|yqCB!_vUi(
    '_10J4PAp@m0}nr#JFtvbi%g@+SXh}o>b~Y>OuO(&{j(LNG*z0BQ_6_j&4x<Z@7~itADmN4Q?0+kOL;!zt>b>7
    '@6WGYw8wImi#01wRXsD>#T+v}kGlBH?{8YIbg|*%3C}ERJj*rTottj;lrLN8#^keXCE(Lqq^ZVzhB2+}fo7MU
    'Vr%cT!1%<Hlc(6G@vA8Jic>V`ePYxp#&zA2VU>zOPZ|O_$#PxD<#7fWCpmOk<7?HopJX$;;U!GZy?K)bRZeo7
```

```
    'q4V(pCs-E~{>!5OoZ#xu8;)0I7t^Mv_+on78z|;B0gTZGewjt=tiho@zQILIGUN4UAwy03Hx=@tY0tDmrke5
    'fPFBQ*+4B|a^yI}e0*k(8NdQwTWK>JH*WQ|nCuv%3T)<LdkbBCIviu3ecYp*8(^5|D1)D}nTMWnuUaiV%2gs
    '3pL2+g1!dn`FuakJoqpTd$Z+X4()jE*Ec@SWBSXt{^!mc^4QVPef%LN+QgO1Mnf-gxg2Q^)<I^QagI62ZfiD
    '0Q(qZ#%A0;F1OpdmtFx@*+=f>M7u0|m~XZ{d=Eno8Nlx5>Kp5=$=pqEtI4<S-NoqXX7YBi_$M=IISe&4|3nT
    '$Y%NjMmVw=VgrZtemuT-`+)5%GIG=J52hMg>yyP;1D~L69ApbwCfi0Dg2|xUV)4ulTe;=QRu(k3F!49@W-~`
    'k;Z(R=r%Aq=Ii_4*s-2IhIZ0_<_4Si*D=hLYsHUjbejhMz(qzbKUl*c1N&*;GUbK|AKkfc70+%T@ZX8+zhbC
    '?})SBh108z{U2WX9tpY*Ub+vVS_fWQ7uk9a67*d7X&*??zTnb65vqM-wa$TNIv2uqPGsxcSfn~2TXjK`>I6U
    'AJI&GMUeW84eC2WH3ultTp(I=0&mR?R%?zBqPar8<_wcGckt32V!7rLL7G$S)7&ClbBxPf?YYK6%{jVh?vX7
    'i8SF9Lxfwngk!`C*GLr3Q6Su-qj&&six;p}Jb@d;8weARK(urO)#4fS7VjWIJcNVdCHRV`5Fy?|ns^LXwEzZ-
    'n#8;4D;`Focp1mU)9@2-W3qT08RB&`7tbS1ypOM?18_?h&|5lzInoW}N=M);UBLwD4AP`KxSD3`5C%z?uv9t)
    'x``y|D2k=4=q#PZH0ds~q{Hx*F5@NXG*YD7sFRLkrF0!TrSou0_u=>eIKBXmPk`ea;P?nQz5-418Ms@%1AXN
    'uvoqbTji5*M!pF*%15Dxd=<jvvoJ%x3oGTruv5MaZuvCaCf|nM@^Kg~Uxx(wJgk-PL#})v&dV3VS3VK_<Qow
    'kttsbmwYZ-$oC>pJ{ZI0i{bcWIKCN<kA~x`;rMJgz8j7YhvUn!Rz4lM^6fY$ACI>3^>BPXBINrqUpWA2$^|&
    'L7;LAhAYP)PPqo(DCc08at}PpLGV#7La=fY#wa&oj&c-Im8+1aoCPO$p`CIVpj-wgry)tX4Ni_jv2q=poCkm
    '6swg>;p9}*D!0PPu?SYKg_CnJSGgB!m4o5rVq8#8hOcrnoE(iP<!U4;XT!<eC{_-KlgsfxP)h>@6aWAK2mk;
    '0RRvHAP@im00000Xb%7Y00000otODL6pYu#QzY9%;*q6HlD$%t@|06(AuY6^WF1>b3ME7#Yg!SNL|L+veK{C
    '_j!Ki;^}jy9BVw+c*_%QZ`s_G*CfavvO6HJLXfv}z3G0_>biyNO``4p@#9v`w`^DWEeER`wyV9uK2=o}!X8B
    '1!i+3Yw&nQBV`;e6~vf=L)=m<2Rj~9&#YkvoUntn23KZI#M80=gCl4pe!wf}DYn@@4sS$qq>|!N(JAL25j~D
    '31)(yfi%JTPY5n_r9Y>+B%r3iJCEY*7Z@8V&$w$+ieJAG_>vT(QEM_i_i9HZXzkiAwJ9eVv@3jH1|3a?HkpZ;
    'YrzU4wz+WS^6Qrzv~bwA!9#Z2ND1~fTK-I4euY(EsRxt$Nf^i})^pak5U<dSRJf^Gn6~qF<01KcoRc{JNA7S
    'ey>B^^d~p;Rw_R3ZD-rYv>=7hS}^M910sHA;_vE;G5*F#XJSh(-tLoFE}*0$_2?7jR_iKUN<iNtg$T&;>-Xl
    'tTCPbgmPc1P%ym^2|<5VBpsVEeXdqvqBS3b4&O=S`bNS7_MV@1N93UJ+sPSy!7h{tzw}@ylZrAy(-S>@C3xt
    '$d{z^kj|Nhm7Moyhhj+hftAx3yR#6FXKa5|x1$<2$NM~7RINenD1-L}2T2elr`E1uSA)xAdfa(CYVnD8X%IS)
    'WtM|`#Vv~)w2F}+IPxif#2!Qgg=dq_<zYkMo!=F@tr&IRU@K{{6^Yi{8l9=h5ISGl>#~i6qGAU<OHMXpuIHD
    'KH`_tD?<Zwd#B+M5}xO5zbw#RjjeYc?aFVe#pP^6-pGGb(MQEr<3f5Ijz?*BSw^KIIUw}!{I+U*{p1iU<{TM
    'lc39hkdRJpfDO8GN8-6+z&WPz$imNDywEOD-Na}FOEvbl(~5MM^Rm7~lxqaGf};ufY(1#Ftv6bZpg?!*?%)T
    '8#TkdaHlhrNCy0YqoKz4>%jSsP2G1PIyjY2$ha)i;2zI~2u)uKD7PQbc@I6H;N~UM7SsiDp(z^<ST%s^Be!c;
    'e}yyK!N+WJ^A3$ps3hrh&d|GoU=^})JE;wR>2`>}Rd0a6L_ekJOEj?Ee6Wanx*iU5aa=AI>V^h3duzovDy-A
    't}5=H=1&^TYG;nTe>4crk+vo0g(&dY)s&H%MuW&2+fZI%8Z-wc9hP`Xf!Y0}z}F)*$OuSCo#CcKhK*xQOI{C
    '9hmHO0k$%Y!0KI@E7+R|$)Wza+d5jHYhH26TCWM7OLL0kakfCfIdLK8N+F165BlUYy5MZ_{HO4O8ek?GA2AL
    '6fXk>?IJtr&}5kVdc5pq799rLYIX(iXJNy$U)%-1L%}JqN6EDP6I9$>*>^U#9yXnJ>rdd!hl9Q9cdKRVA!i8
    'OsNT8B7u(1j&IV>szFp%H<y@Iij_Ldu?@vJP`7>ffZkXewAXDMtemTXlZ3hfi>VjjB`V;cOlbzDYyb1Tb7e4
    ';lCpRiK)bLxYM7q)#P9?s9SOcD;tsEF(HIDD$)Y(@2{J2f2D$($6FP<#wNI?9NBwSsT<m3U86Nrx`6E{FGRd
    'NA$pVVj!`2g#wnFgX}JB?t|<*FXc_N`yqE%JbSm$5XcqpX|cM?0-{o4y-~<8)aaXZx!a6Gfv-zh-jgB7NH)%
    'zO=FH5FC(fYDj%I3Ww8necKZ|0b$Y>E0v$8!2BDl%1~wsq&Mj74*W0$?>-(J+WmYIUSWh-fX@s>w^<16>6nG
    '2DaK*tbE8A2gUPA2N@Alus1dP4RK-tisNMjx4i!XTF3Rb^^j*E#<riAe|28}QqqrOp|h~8!yL~zH4n``AzC6
    '!Y|A&f@?bU$TEE%m{Or<o!`zvyVUaC_Qlmb<du3ct3`OSXrxL`orgn|D!mo^8K_zmKP$-p6{e+FVO>w=!J?Q
    '&6Z=LZyth$xzd9K=787R`C5+X6fCUmeIa5r3y--<@}32*>S`L+Ichlr4OTMU<IHIYGv|qVkvR^0n>uMf-Nqo
    'afo-%j$L^>1qtT+lgT&=GhW9Ag_g$Qd#n1x?3--J8PFhfDvpEljMCJ_%?t>XsQke%(hstO>3g@|U-diBgBb1
    'E|CGH_TI$}jr3K0%p>OHZr~2oaR$Q<Fh6m0K~k0q7P4Y4W$x9$<L<*$fAj&?a10ZbU#r7)lVh4sZOdR^n`;K
    'FYX?ELACum{qi0DCvg1f(N|UI|M0!WjescBuhZ1}N2mtPMQzS)Unb+74a&LObMldv&a6Hj-i?38jE?H^uyBI
    'nMZn$US##)gDWkAJuZEy(YIXYzlMdsLfV2QCI`^1H)l_U-4GhaKULD@8NjG$cK5!wL#V?eyi68lqcqQKbn%H
    '=s1e$YP1ro^Ph1@|NM5V&Jpx->I)8J4Wn>|9Lc711YhNJWS6)zF`<ax@?g;@+6SwWPfm<u$^iR$Ui>hwYltI
    'U=!UiiTgMN53p3nQ0;a5@V=L$=qYbB0PjaIbeqX!C20h$mA|f-wGUx9Y0H^=%PjoC5Z{=oJBAnfzwCOd$-)g
    'XnuWXQ&uYrgB433XS(&_$sofjuLL^2=8|#~KGTH-&R^~LWCrofq6fF*sX-)Fa3>Yz_Tr)4!hV+%htbSOd;gy
    'RsK-X(&xo`ZI>~;8(#Xe&0-jR-rOREuk<2s!~!~O>%>y;_#^6!9^@#rQB_as!PpZU?F~G7@R-o<sLJLcJbfZ)
    'B|DWHQjoaTpyv~*6IDOnbMo_Mpq^ZvbM7HJ$_f}C-|~GB<<~C1*y7oRl5F1%g!T1!NcUG`ms|^~n-?8zNoV4
    '$5%;9j+aHd(bPhY9O=-4Q?qfO&)BW<&Ln1NT`LNGeEa=gGX+ES6eaxk^kCE6q~(BNH=eGNI>))E4rkV%D)U;
    '$Hzb&zWBK6L;ZO7_ugZ?>pJm~D|>g29Ru}?1LTb&hA`emyY+8xCq|sqWV3fuP`GT1w|9RP#%O3g8-3i3OTif
    'IWAbq#6k~ab%k0M4u+3T`<BzNCeyk2@G%zhI|YefUPr~k{nB&q2>rO<j~bP!(S?gUK3{($(2EnVrd#4h+A+d
    'Jo>hFP}P%;DQg`C$DeeduC<D$kPjKv&4k?h3Mu#?OU#Q;rw@-<{?I<fL&1~MBJ+)5badu%6bmt-V~fF$!xoB
    'qZ)KN^r6J}B^iV7oVk$|-G*zq7YDKuTe1CY?c)oT9T-cC___TS1?$A7g1h+Ikmxi#$q=c-Zn4_YZ^a+sC+{K
    'W%{zZ+fY6E;JIyuwfK-=w6u3MU$1_bX0Z+wJS{!h^gB5g4HRfw1}<g-S8lLCy<`pi`y}_AJbw{-mUE_W{gsL
    'H4GQi$q8F`y++$lzclr3B!akj_?r;xMo>%I8e}5T00h)Gn*8F6(RWL9Uwo|wBfg9GCGQtPjEF)2(W4oznr7{
    'Ti=Ll>VaR#RoYcG1N3=1zZx`BAo<tbEn;$2aHM+D$&^wJh5WI__M4jE!4g$a<zFhaPU}68&}@YAgtm`C&fV~
    'fcx?Ox9mt=plUe6URr7Zm+aFY<<_?YqunPv+`0pj`Qjg4+fV?+v+I6D&h^0hN7m0q)aW2`Yi}{kP@!}xIzro
    'jBFh+J)W;S5l4mT2^SuM;A%d;oua0d)CK<oP)h*<6ay3h000O8001EXu*W&Jg988npaTE^3jhEB000000000
    '0000X06#iWD2D?80H6Z^01E&B00000000000Du9&0{{SYa$#w1UwJNWaCuNm0Rj{Q6aWAK2mk;8Apn6>1aKG
    'Usx_~aCuNm0Rj{Q6aWAK2mk;8ApnKqayn=a001Bm000UA000000000004ji*bx8#bY*jNUwJNWaCuNm1qJ{
)))
```


```
train_X, train_y = data['train_X'], data['train_y']
test_X, test_y = data['test_X'], data['test_y']
```

Vamos utilizar a eurística "A Practical Guide to Gaussian Processes" by Marc Deisenroth et al." referenciado pelo eClass para encontrar os hiperparâmetros ótimos. O código abaixo faz isso para nós.

In [39]:

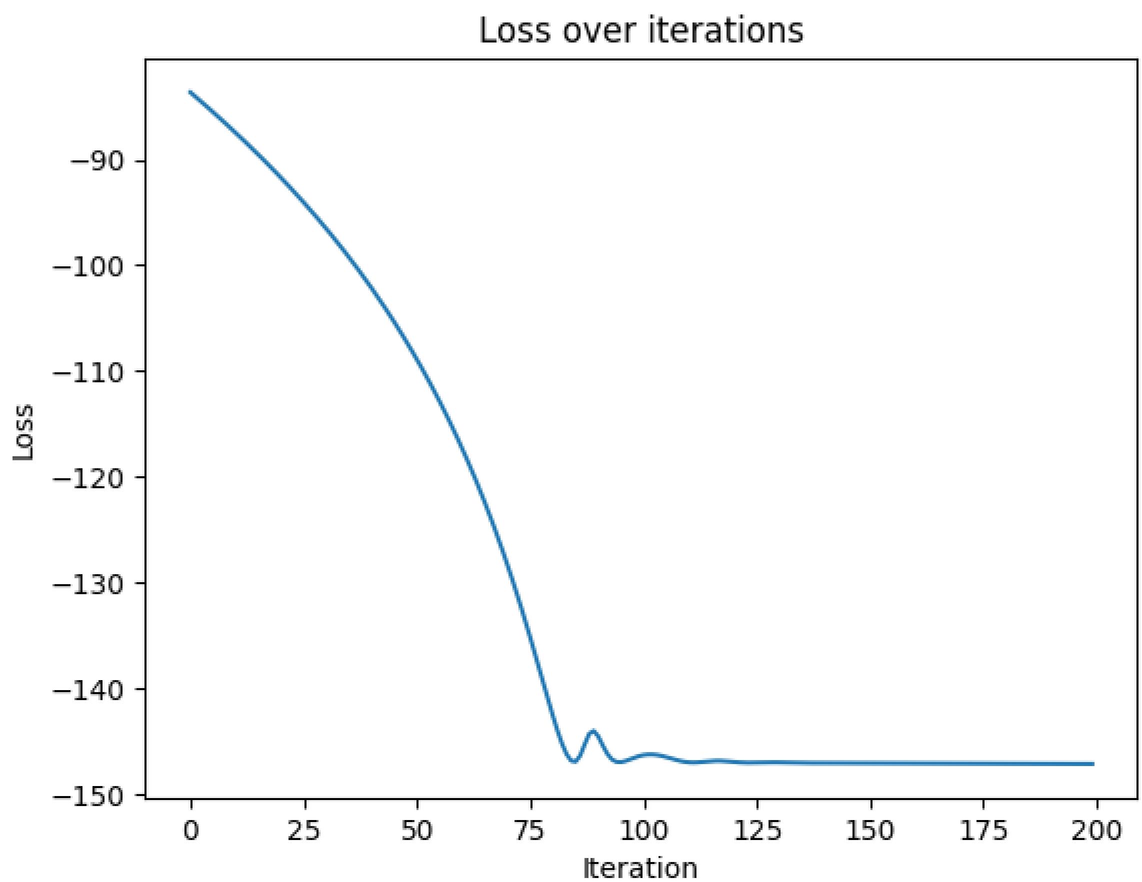
```
# define initial parameters with heuristics
gamma_values = torch.linspace(1, 10, 10) # gamma should vary from being divided by 2 to 1(
sigma2_n_values = torch.linspace(2, 100, 15) # sigma_n should vary from being divided by 2
initial_sigma2_f = torch.var(torch.tensor(train_y))
best_loss = np.inf

# rbf kernel
def rbf_kernel(x1, x2, gamma, sigma2_f):
    x1 = torch.tensor(x1)
    x2 = torch.tensor(x2)
    return (-gamma*(torch.cdist(x1, x2)**2)).exp()*sigma2_f

# loss function
def loss(X, y, gamma, sigma2_f, sigma2_n):
    X = torch.tensor(X)
    y = torch.tensor(y)
    K = rbf_kernel(X, X, gamma, sigma2_f)
    return -0.5*torch.logdet(K + (sigma2_n*torch.eye(len(X)))) - 0.5*(y.T @ torch.inverse

# train model
def optimizer(X, y, gamma, sigma2_f, sigma2_n):
    opt = torch.optim.Adam([gamma, sigma2_f, sigma2_n], lr=0.001)
    losses = []
    gammas = []
    sigma2_ns = []
    sigma2_fs = []

    for i in range(200):
        gammas.append(gamma.item())
        sigma2_ns.append(sigma2_n.item())
        sigma2_fs.append(sigma2_f.item())
        opt.zero_grad()
        updated_loss = -1 * loss(X, y, gamma, sigma2_f, sigma2_n)
        losses.append(updated_loss.item())

        updated_loss.backward()
        opt.step()

    return losses, gammas, sigma2_ns, sigma2_fs

initial_sigma2_f.requires_grad = True

# train model using multiple starting points for gamma and sigma2_n
for gamma_value in gamma_values:
    for sigma2_n_value in sigma2_n_values:

        # initialize parameters
        gamma = gamma_value/(2 * torch.std(torch.tensor(train_X)))
        gamma.requires_grad = True
        sigma2_n = torch.sqrt(initial_sigma2_f).detach()/sigma2_n_value
        sigma2_n.requires_grad = True

        # apply optimizer
        losses, gammas, sigma2_ns, sigma2_fs = optimizer(train_X, train_y, gamma, initial_

        # check if loss is better than previous best
        if losses[-1] < best_loss:
            best_loss = losses[-1]
            best_loss_vector = losses

            # save best parameters vectors
            best_gammas = gammas
```

```
            best_sigma2_ns = sigma2_ns
            best_sigma2_fs = sigma2_fs
```

```
C:\Users\Daniel\AppData\Local\Temp\ipykernel_11892\4226813858.py:9: UserWarning: To copy c
onstruct from a tensor, it is recommended to use sourceTensor.clone().detach() or sourceTe
nsor.clone().detach().requires_grad_(True), rather than torch.tensor(sourceTensor).
  x1 = torch.tensor(x1)
C:\Users\Daniel\AppData\Local\Temp\ipykernel_11892\4226813858.py:10: UserWarning: To copy
construct from a tensor, it is recommended to use sourceTensor.clone().detach() or sourceT
ensor.clone().detach().requires_grad_(True), rather than torch.tensor(sourceTensor).
  x2 = torch.tensor(x2)
```

In [40]:

```
#plot loss
plt.plot(best_loss_vector)
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.title('Loss over iterations')
plt.show()
```



In [41]:

```
# evidence
evidence_result = loss(train_X, train_y, best_gammas[-1], best_sigma2_fs[-1], best_sigma2_
print('Evidence: ', float(evidence_result.detach()))
```

```
Evidence:  147.17627492591882
```

```
C:\Users\Daniel\AppData\Local\Temp\ipykernel_11892\4226813858.py:9: UserWarning: To copy c
onstruct from a tensor, it is recommended to use sourceTensor.clone().detach() or sourceTe
nsor.clone().detach().requires_grad_(True), rather than torch.tensor(sourceTensor).
  x1 = torch.tensor(x1)
C:\Users\Daniel\AppData\Local\Temp\ipykernel_11892\4226813858.py:10: UserWarning: To copy
construct from a tensor, it is recommended to use sourceTensor.clone().detach() or sourceT
ensor.clone().detach().requires_grad_(True), rather than torch.tensor(sourceTensor).
  x2 = torch.tensor(x2)
```

In [45]:

```
gamma = best_gammas[-1]
var_noise = best_sigma2_ns[-1]
c = best_sigma2_fs[-1]
loss_vec = best_loss_vector

print('gamma: ', gamma)
print('c: ', c)
print('variância do ruído: ', var_noise)
print('loss: ', loss_vec)
```

```
gamma:  10.381698097500038
c:  0.8327156143957126
variância do ruído:  0.010380541912552676
loss:  [-83.63070172523106, -83.99908089313182, -84.37053792272528, -84.74517694902094, -8
5.12310234979364, -85.50442723503849, -85.88926239181797, -86.27772480976239, -86.66993213
183439, -87.0660143898122, -87.46609385404486, -87.87030554867798, -88.27878277187692, -8
8.69166915186645, -89.10910378845739, -89.53123959005343, -89.95822517629013, -90.39022357
628164, -90.8273969946832, -91.26990997129519, -91.71793922025064, -92.17166438586403, -9
2.63126817342628, -93.09694652737944, -93.56889577235299, -94.04731944540771, -94.53243211
035473, -95.02445643566898, -95.52362007878413, -96.03015950001527, -96.54432403287187, -9
7.06636927527582, -97.59656479756259, -98.13518743660856, -98.68252924095765, -99.23889449
990162, -99.8046004256562, -100.37997788035563, -100.96537215085918, -101.5611478058178, -
102.16768575943469, -102.7853842255252, -103.4146597339495, -104.05595246968652, -104.7097
2341119912, -105.37646214524024, -106.05667978366827, -106.75091022763758, -107.4597275374
7864, -108.1837253133227, -108.92353896127739, -109.67983159304295, -110.45331262531934, -
111.24472086402571, -112.05484612414774, -112.88451948790168, -113.73461535782667, -114.60
606142160692, -115.49983574766743, -116.41697134768896, -117.35854931989904, -118.32571049
301013, -119.31964681328412, -120.3416022180318, -121.39285945491446, -122.47474143458855,
-123.5885933450073, -124.73575770818854, -125.91754298665805, -127.13516872539765, -128.38
971326249938, -129.68199697135066, -131.0124434450092, -132.38086864113112, -133.786173448
23374, -135.2258948079761, -136.69555360347903, -138.18767276704963, -139.69031500075283,
-141.18491029180421, -142.64296677722325, -144.021235316613, -145.25482180353123, -146.248
55292596393, -146.87018786998746, -146.96067592244114, -146.40597362320955, -145.336933041
7026, -144.32961306851854, -144.05120764398464, -144.56703023050437, -145.43383315896108,
-146.2239664206164, -146.74346700744286, -146.98132065081754, -147.00711665235, -146.90478
987318593, -146.74474306282457, -146.5772625011971, -146.43442728040083, -146.334198110844
37, -146.28427429876461, -146.28507420048018, -146.331856810134, -146.41619964022607, -14
6.527062174813, -146.65166824117077, -146.77640035254151, -146.88788456710938, -146.974392
53170646, -147.02756470829547, -147.04423993672148, -147.02785011541357, -146.988554964919
64, -146.94135365597302, -146.90214152940172, -146.88292343141177, -146.8882994719156, -14
6.9149236282749, -146.95396277194666, -146.99500444072854, -147.02948655293275, -147.05252
492131572, -147.06308592779652, -147.06308082396572, -147.05606765019746, -147.04604421086
086, -147.0365606926893, -147.03019766671358, -147.02835983657448, -147.0313066941913, -14
7.0383426871311, -147.04810816431075, -147.0589166867895, -147.06909954414354, -147.077310
70320034, -147.08275278230985, -147.08528434436522, -147.08538670487343, -147.083995286490
1, -147.08223492641258, -147.08112896749012, -147.08136218360735, -147.0831608007418, -14
7.08631046320028, -147.09028999298374, -147.09446163430692, -147.09825505678995, -147.1012
9453036922, -147.10344996140705, -147.1048158976991, -147.10564241270492, -147.10624574647
153, -147.1069240848867, -147.1078953536543, -147.10926558618274, -147.11102690479865, -14
7.11308054531338, -147.11527504227377, -147.1174492381293, -147.1194707653232, -147.121262
25796817, -147.12281087790652, -147.1241615172744, -147.12539706083567, -147.126612212579,
-147.12788832836526, -147.1292753974264, -147.13078500604703, -147.13239424906692, -147.13
40577857819, -147.13572356358557, -147.1373473992879, -147.13890271199625, -147.1403839814
918, -147.14180423829185, -147.14318834484422, -147.14456463621673, -147.1459572537202, -1
47.14738101689727, -147.14883941392605, -147.15032590398738, -147.15182734408603, -147.153
3284755489, -147.15481623881556, -147.15628269725187, -147.1577262266082, -147.15915085117
13, -147.1605642416482, -147.1619751131975, -147.1633907538851, -147.16481538434198, -147.
16624958320463, -147.16769081829224, -147.1691346800888, -147.17057641946485, -147.1720122
712316, -147.1734403075096, -147.17486061843894, -147.17627492591882]
```

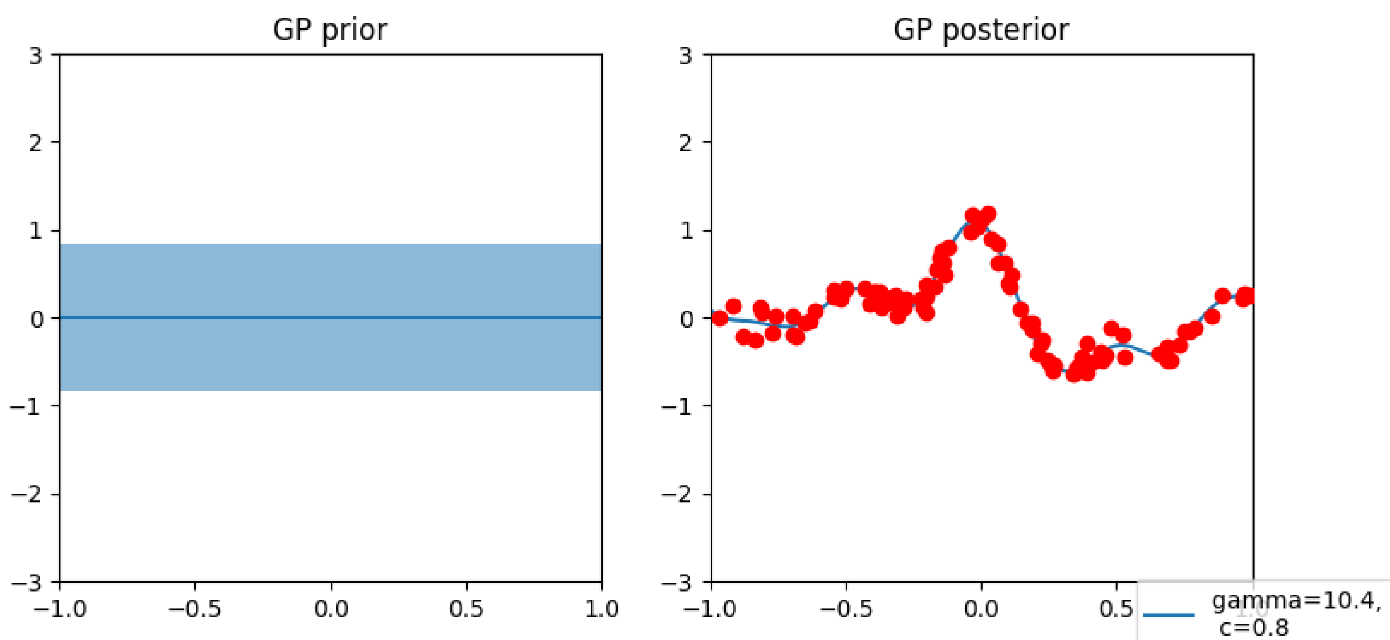
In [49]:

```
# plot prior and posterior
x = torch.linspace(-1, 1, 100)[:, None]
fig, axs = plt.subplots(1, 2, figsize=(9, 4))

K = rbf_kernel(x, x, gamma, c) + torch.eye(x.shape[0])*var_noise
```

```
mu = torch.zeros_like(x)

axs[0].plot(x, mu)
axs[0].fill_between(x.flatten(), mu.flatten()-K.diag().detach().numpy(), mu.flatten()+K.di
# axs[0].fill_between(x.flatten(), mu.flatten()-K.diag(), mu.flatten()+K.diag(), alpha=0.
axs[0].set_xlim([-1, 1])
axs[0].set_ylim([-3, 3])
axs[0].set_title('GP prior')

post_mu,post_cov = posterior_pred(x, torch.tensor(train_X, dtype=torch.float32), torch.te
# axs[1].plot(x, post_mu)
axs[1].plot(x, post_mu.detach().numpy())

# axs[1].fill_between(x.flatten(), post_mu.flatten()-post_cov.diag(), post_mu.flatten()+po
axs[1].fill_between(x.flatten(), post_mu.detach().numpy().flatten() - post_cov.diag().deta

axs[1].scatter(train_X, train_y, color='red', zorder=5)

axs[1].set_xlim([-1, 1])
axs[1].set_ylim([-3, 3])
axs[1].set_title('GP posterior')

# make legend with gamma and c
fig.legend(['gamma={:.1f}, \n c={:.1f}'.format(gamma, c)], loc='lower right')
```

```
C:\Users\Daniel\AppData\Local\Temp\ipykernel_11892\4226813858.py:9: UserWarning: To copy c
onstruct from a tensor, it is recommended to use sourceTensor.clone().detach() or sourceTe
nsor.clone().detach().requires_grad_(True), rather than torch.tensor(sourceTensor).
  x1 = torch.tensor(x1)
C:\Users\Daniel\AppData\Local\Temp\ipykernel_11892\4226813858.py:10: UserWarning: To copy
construct from a tensor, it is recommended to use sourceTensor.clone().detach() or sourceT
ensor.clone().detach().requires_grad_(True), rather than torch.tensor(sourceTensor).
  x2 = torch.tensor(x2)
```

Out[49]: `<matplotlib.legend.Legend at 0x2aa8a2effd0>`



In [51]:

```
# log marginal likelihood
log_marginal_likelihood = loss(test_X, test_y, gamma, c, var_noise)
print('Log marginal likelihood: ', log_marginal_likelihood.item())

# mse
test_mean, test_cov = posterior_pred(torch.tensor(test_X, dtype=torch.float32), torch.tens
mse = torch.mean((test_mean - torch.tensor(test_y, dtype=torch.float32))**2)
print('MSE: ', mse.item())
```

```
Log marginal likelihood:  871.5346305656243
MSE:  0.010259582661092281
```

```
C:\Users\Daniel\AppData\Local\Temp\ipykernel_11892\4226813858.py:9: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().requires_grad_(True), rather than torch.tensor(sourceTensor).
  x1 = torch.tensor(x1)
C:\Users\Daniel\AppData\Local\Temp\ipykernel_11892\4226813858.py:10: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().requires_grad_(True), rather than torch.tensor(sourceTensor).
  x2 = torch.tensor(x2)
```

Exercício de "papel e caneta"

1. Na nota de aula, derivamos a posteriori preditiva $p(y_\star|x_\star, x_1, y_1, \dots, x_N, y_N)$. Por simplicidade, deduzimos a priori preditiva $p(y_\star, y_1, \dots, y_N|x_\star, x_1, \dots, x_N)$ e as condicionamos nas saídas $y_1, \dots, y_N$ observadas no conjunto de treino. No entanto, também é possível obter o mesmo resultado calculando a posteriori $p(f_\star, f_1, \dots, f_N|x_\star, x_1, y_1, \dots, x_N, y_N)$ e, então, calculando o valor esperado de $p(y_\star|x_\star, f_\star)$ sob essa posteriori. Deduza novamente a posteriori preditiva seguindo esse outro procedimento.

(Dica: você pode calcular a conjunta $p(f^*, f_1, \dots, f_N, y_1, \dots, y_N|x_*, x_1, \dots, x_N)$, que também será Gaussiana.)

Seja $\mathbf{f} = (f_1, \dots, f_N)$, $\mathbf{y} = (y_1, \dots, y_N)$ e $\mathbf{x} = (x_1, \dots, x_N)$. Queremos calcular a conjunta $p(f_*, \mathbf{f}, \mathbf{y}|x_*, \mathbf{x})$, que pode ser escrita como

$$p(f_*|\mathbf{y}, \mathbf{x}, x_*)p(\mathbf{f}|\mathbf{x})p(\mathbf{y}|\mathbf{f})$$

onde cada uma dessas distribuições é gaussiana. Dessa forma, temos

$$f_* \sim \mathcal{N}(\mu(f_*), \Sigma) = (0, \Sigma)$$

$$f \sim \mathcal{N}(\mu(f), \Sigma) = (0, \Sigma)$$

$$y \sim \mathcal{N}(\mu(u), \Sigma) = (0, \Sigma)$$

Em que Σ é a matriz de covariâncias

$$\Sigma = \begin{bmatrix} cov(f_*, f_*) & cov(f_*, f) & cov(f_*, y) \\ cov(f, f_*) & cov(f, f) & cov(f, y) \\ cov(y, f_*) & cov(y, f) & cov(y, y) \end{bmatrix} = \begin{bmatrix} k(x_*, x_*) & k(x_*, \mathbf{x}) & cov(f_*, y) \\ k(\mathbf{x}, x_*) & k(\mathbf{x}, \mathbf{x}) & cov(f, y) \\ cov(\mathbf{y}, f_*) & cov(\mathbf{y}, \mathbf{f}) & var(y) \end{bmatrix}$$

Vamos marginalizar f:

$$p(f_*|\mathbf{y}, \mathbf{x}, x_*) = \int p(f_*, f, \mathbf{y}|\mathbf{x}, x_*)df$$

Assim eliminmos os termos que dependem de $\mathbf{f}$ na matriz de covariâncias resultando em

$$f_* \sim \mathcal{N}(0, \Sigma')$$

$$y \sim \mathcal{N}(0, \Sigma')$$

onde Σ' é a matriz de covariâncias resultante da eliminação dos termos que dependem de $\mathbf{f}$ na matriz de covariâncias original. $$ \Sigma' =

$$\begin{bmatrix} k(x_*, x_*) & cov(f_*, y) \\ cov(y, f_*) & var(y) \end{bmatrix}$$

Queremos ainda condicionar essa distribuição em $\mathbf{y}$. Para isso vamos tentar simplificar as entradas.

Como $cov(f_*, y) = \mathbb{E}_{f_*,y}[(f_* - [f_*])(y - [y]^T)]$

COmo $y = f_\epsilon$ podemos manipular a expressão acima de modo a obter $\mathbb{E}_{f_*,f+\epsilon}[(f_* f)] = k(x_*, x)$

Daí, temos que $cov(f_*, y) = k(x_*, x)$

$$f_* \sim \mathcal{N}(0, \Sigma'')$$

$$y \sim \mathcal{N}(0, \Sigma'')$$

onde

$$\Sigma'' = \begin{bmatrix} k(x_*, x_*) & k(x_*, x) \\ k(x, x_*) & var(y) \end{bmatrix}$$

$$

Agora, para termos $p(f_*|y, x, x_*)$ vamos considerar que y são observações, para então termos $p(f_*|y, x_*, x) \sim (\mu_*, \Sigma''')$

Com $\mu_* = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(y - \mu_2) = k(x_*, x)\Sigma(y)^{-1}y$ e $\Sigma''' = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} = k(x_*, x_*) - k(x_*, x)\Sigma(y)^{-1}k(x, x_*)$

Assim, temos a distribuição $p(f_*|y, x, x_*)$ sendo uma normal com média μ_* e variância Σ'''. Sabendo que

$$p(y_*|f_*, x, x_*) \approx p(f_* y_*, x, x_*)$$

(pois f_* é a média de y^*). Com isso obtemos o produto de duas normais avaliadas na mesma variável f_* o que nos permite usar a fórmula

$$\mathcal{N}_x(m_1, \Sigma_1)\mathcal{N}_x(m_2, \Sigma_2) = \mathcal{N}_{m_1}(m_2, \Sigma_1 + \Sigma_2)\mathcal{N}_x(m_c, \Sigma_c)$$

Aplicando ao nosso problema teremos como resultdo $\mathcal{N}_x(m_c, \Sigma_c)$ uma normal em função de f_* onde a integral resultante é 1. Daí, precisamos obter $\mathcal{N}_{m_1}(m_2, \Sigma_1 + \Sigma_2)$, que é a distribuição de y_* condicionada em x_*, x e y.

$$p(y_*|y, x, x_*) = \mathcal{N}_{m_1}(m_2, \Sigma_1 + \Sigma_2)$$

onde m_1 e Σ_1 são, respectivamente, média e variância de $p(f_*|y_*, x, x_*)$ e m_2, respectivamente; Σ_2 de $p(f_*|y, x, x_*)$.

Para encontrar o valor de Σ_1 precisamos lembrar que $p(f_*|y_*, x, x_*)$ é equivalente a $p(y_*|f_*, x, x_*)$, mas que com f_* dado a variância é dependente exclusivamente do ruído. Daí, $\Sigma_1 = var(\epsilon) = \sigma^2$.

Como conclusão, temos que

$$\mu^@ = m_2 = k(x_*, x)(k(x, x) + \sigma^2 I)^{-1}y$$

$\Sigma^@ = \Sigma_1 + \Sigma_2 = \sigma^2 + k(x_*, x_*) - k(x_*, x)(k(x, x) + \sigma^2 I)^{-1}k(x, x_*)$

Gerando a posteriori preditiva

$$p(y_*|y, x, x_*) = \mathcal{N}(\mu^@, \Sigma^@)$$

2. Quando trocamos a verossimilhança Gaussiana por uma Bernoulli (i.e., no caso de classificação binária), a posteriori para nosso GP não possui fórmula fechada. Mais especificamente, a verossimilhança para esse modelo é dada por $y|x \sim \mathrm{Ber}(\sigma(f(x))$ onde σ é a função sigmoide. Em resposta à falta de uma solução analítica, podemos aproximar a posteriori sobre f para qualquer conjunto de pontos de entrada usando as técnicas de inferência aproximada que vimos anteriormente. Discuta como usar a aproximação de laplace nesse caso, incluindo as fórmulas para os termos da Hessiana. Além disse, discuta como usar o resultado desse procedimento para aproximar a posteriori preditiva.

Queremos fazer uma classificação binária usando uma Bernoulli ao invés de uma verossimilhança Gaussiana. Assim, a verossimilhança é dada por $p(y|f) = \sigma(f)^y(1-\sigma(f))^{1-y}$, onde σ é a função sigmoide e $y \in \{0; 1\}$. Essa será nossa verossimilhança. Podemos avaliar de forma arbitrária $p(y_* = 1|x, x_*, y)$ que é

$$p(y_* = 1|x, x_*, y) = \int p(y_* = 1|f_*, x, x_*)p(f_*|x, x_*, y)df_*$$

Vamos então aproximar essa posteriori para um conjunto dado de entrada usando a técnica de Laplace. Para, isso vamos analisar $p(f_*|x, x_*, y)$

$$p(f_*|x, x_*, y) = \int p(f_*|x, x_*, f)p(f|x_*, x, y)df$$

Nessa integral, temos a que o primeiro fator é uma normal obtida utilizando processos gaussianos para a regressão. Manipulando, atingimos

$$p(f_*|f) \approx \mathbf{N(0, k(x_*, x_*))}$$

Sabemos ainda que podemos escrever

$$p(f, f_*|x_*, x) = \mathcal{N}\left(\begin{bmatrix} f \\ f_* \end{bmatrix} \middle| \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} k(x,x) + \sigma^2 I & k(x, x_*) \\ k(x_*, x) & k(x_*, x_*) \end{bmatrix}\right)$$

Então a posteriri é dada por uma normal de média μ_* e variância Σ_*, onde

$$\mu_* = k(x_*, x)(k(x,x) + \sigma_f^2 I)^{-1}y$$

$$\Sigma_* = k(x_*, x_*) - k(x_*, x)(k(x,x) + \sigma_f^2 I)^{-1}k(x, x_*)$$

Vamos agora analisar o segundo fator da integral. Podemos também realizar uma aproximação de Laplace:

$$p(f|x_*, x, y) = p(f)p(y|x_*, x, f)$$

Onde $p(f) = \mathcal{N}(0, (k(x,x) + \sigma_f^2 I)^{-1})$ e $p(y|x_*, x, f) = \prod_{i=1}^N \sigma(f_i)^{y_i}(1-\sigma(f_i))^{1-y_i} = \prod e^{f_i y_i}\sigma(-f_i)$. Daí podemos realizar a aproximação de Laplace utilizando o logaritmo da posteriori.

$$\psi(f) = \ln p(f) + p(y|x_*, x, f)$$

Desenvolvendo a expressão acima temos que

$$\nabla\psi(f) = y - \sigma(f) - (k(x,x) + \sigma_f^2 I)^{-1}f$$

e a Hessiana é dada por

$$H = \nabla^2\psi(f)$$

$$H = \sigma(f)(1-\sigma(f)) + (k(x,x) + \sigma_f^2 I)^{-1}$$

Concluímos daí que a aproximação de $p(f|x_*, x, y)$ é $q(f) = (f|f_*, H^{-1})$ onde conhecemos a Hessiana. Para atingir a posteriori preditiva, utilizamos algum algoritmo de otimização. Assim, temos duas aproximações como gaussianas e podemos utilizar a aproximação de $p(f_*|x_*, x, y)$ como $\mathcal{N}(f_*|u_f, \sigma_f^2)$ onde

$$u_f = k(x_*, x)(y - \sigma_f)$$

$$\sigma_f^2 = k(x_*, x_*) - k(x_*, x)((\sigma(f)(1 - \sigma(f)))^{-1} + (k(x, x) + \sigma_f^2 I))^{-1}$$

Por fim, concluimos que a posteriori preditiva é dada por

$$p(y_* = 1|x, x_*, y) = \sigma(\mu_{f_*} + \sigma_{f_*}(1 + \pi\sigma_{f_*}^2/8)^{1/2})$$