2factor-Vasicek model

YiTao Hu, Charles Rambo, Junyu(Kevin) Wu, Jin (Jane) Huangfu

12/02/2020

define a function to compute A and B terms

```
#define a function to compute A,B terms
get_vasicek.AB=function(params,MT){
  alpha=params[1]
  beta=params[2]
  sigma=params[3]
  First_exp_term=((sigma^2/(2*beta^2))-alpha/beta)*MT
  Second_exp_term=(alpha/beta^2-sigma^2/beta^3)*(1-exp(-beta*MT))
  Third_exp_term=(sigma^2/(4*beta^3))*(1-exp(-2*beta*MT))
  A=exp(First_exp_term+Second_exp_term+Third_exp_term)
  B=(1/beta)*(1-exp(beta*MT))
  return(c(A,B))
}
```

define a function to compute X and Y

```
#compute A,B for 0.25 and 10 maturity
ABx0.25=get_vasicek.AB(params = model_coeff[1:3],MT = 0.25)
ABy0.25=get_vasicek.AB(params = c(0,model_coeff[4:5]),MT = 0.25)
ABx10=get_vasicek.AB(params = model_coeff[1:3],MT = 10)
ABy10=get_vasicek.AB(params = c(0,model_coeff[4:5]),MT = 10)
ABs=rbind(ABx0.25,ABy0.25,ABx10,ABy10)
#define a function to solve for X and Y, given A,B's
get_XY=function(ABs,yields){
  Ax0.25=ABs[1,1]
  Bx0.25=ABs[1,2]
  Ay0.25=ABs[2,1]
  By0.25=ABs[2,2]
  Ax10=ABs[3,1]
  Bx10=ABs[3,2]
  Ay10=ABs[4,1]
  By10=ABs[4,2]
  Sol_mat=matrix(c(Bx0.25/0.25,Bx10/10,By0.25/0.25,By10/10),nrow = 2,ncol = 2)
  b_vec=c(yields[1]+log(Ax0.25)/0.25-log(Ay0.25)/0.25,yields[2]+log(Ax10)/10-log(Ay10)/10)
  XY=solve(a = Sol_mat,b = b_vec)
  return(XY)
  }
#define a function to compute X,Y at all time steps
get_all_XY=function(yields,ABs){
  #initilize a matrix to store X,Y at each time point
  t_steps=nrow(yields)
  XY_mat=matrix(0,nrow = t_steps,ncol = 2)
  for (t in 1: t_steps){
    XY_mat[t,]=get_XY(yields = yields[t,],ABs = ABs)
  }
  return(XY_mat)
}
```

```
XY_mat=get_all_XY(yields = Par_Rates[,c(1,6)],ABs = ABs)
```

define a function to compute DT at each time step of each maturity

```
#define a function to compute the DT function
get_DT=function(params,XY_mat,MT){
  ABx=get_vasicek.AB(params = params[1:3],MT = MT)
  ABy=get_vasicek.AB(params = c(0,params[4:5]),MT=MT)
  DT=ABx[1]*ABy[1]*exp(-ABx[2]*XY_mat[,1]-ABy[2]*XY_mat[,2])
  return(DT)
}

#define a function to compute all DT values
get_all_DTs=function(params,XY_mat,MTs){
  #initializa a DT matrix
  DT_mat=data.frame(matrix(0,nrow = nrow(XY_mat),ncol = length(MTs)))
  for (m in 1:length(MTs))
    {
    DT_mat[,m]=get_DT(params = params,XY_mat = XY_mat ,MT = MTs[m])
  }
  colnames(DT_mat)=MTs
  return(DT_mat)
}

DTs=get_all_DTs(params = model_coeff,XY_mat = XY_mat,MTs = seq(0.5,10,0.5))
```

define a function to compute fitted Par_Rates

```
get_fitted_par_rates=function(DTs,MTs){
  #initialize the fitted value matrix
  fitted_yields=data.frame(matrix(0,nrow = nrow(DTs),ncol = length(MTs)))
  colnames(fitted_yields)=MTs
  i=1
  m=2
  for (m in MTs){
    fitted_yields[,i]=2*((1-DTs[,(2*m)])/(rowSums(DTs[,1:(2*m)])))
    i=i+1
  }
  return(fitted_yields)
}
fitted_par_Rates=get_fitted_par_rates(DTs = DTs,MTs = c(2,3,5,7))
```

define a function to compute the RMSE

```
comp_RMSE=function(fitted,observed){
  observed=observed[,c(-1,-ncol(Par_Rates))]
  RMSE=((sum((fitted-observed)^2))^0.5)/4
  return(RMSE)
}
Cost=comp_RMSE(fitted = fitted_par_Rates,observed = Par_Rates)
Cost
```

```
## [1] 0.05898783
```

Assmble all the functions into a pipeline to get CostFunction:

```
CostFun=function(model_coeff,data){
  #compute A,B for 0.25 and 10 maturity
  ABx0.25=get_vasicek.AB(params = model_coeff[1:3],MT = 0.25)
  ABy0.25=get_vasicek.AB(params = c(0,model_coeff[4:5]),MT = 0.25)
  ABx10=get_vasicek.AB(params = model_coeff[1:3],MT = 10)
  ABy10=get_vasicek.AB(params = c(0,model_coeff[4:5]),MT = 10)
  ABs=rbind(ABx0.25,ABy0.25,ABx10,ABy10)
  #compute XY's
  XY_mat=get_all_XY(yields = data[,c(1,6)],ABs = ABs)
  #compute DT function
  DTs=get_all_DTs(params = model_coeff,XY_mat = XY_mat,MTs = seq(0.5,10,0.5))
  #compute fitted values
  fitted_par_Rates=get_fitted_par_rates(DTs = DTs,MTs = c(2,3,5,7))
  #compute RMSE
  Cost=comp_RMSE(fitted = fitted_par_Rates,observed = data)
  return(Cost)
}
CostFun(model_coeff = model_coeff,data = Par_Rates)
```

```
## [1] 0.05898783
```

Perform the optimization

```
coeff_obj=nlminb(start=model_coeff,objective=CostFun,lower = c(-Inf,-Inf,0,0,0),data=Par_Rates)
coeff_obj
```

```
## $par
## [1] 2.456870e-02 1.573794e-02 9.899637e-03 3.562569e-06 1.088867e-03
##
## $objective
## [1] 0.05530202
##
## $convergence
## [1] 1
##
## $iterations
## [1] 16
##
## $evaluations
## function gradient
##       47       80
##
## $message
## [1] "false convergence (8)"
```

5

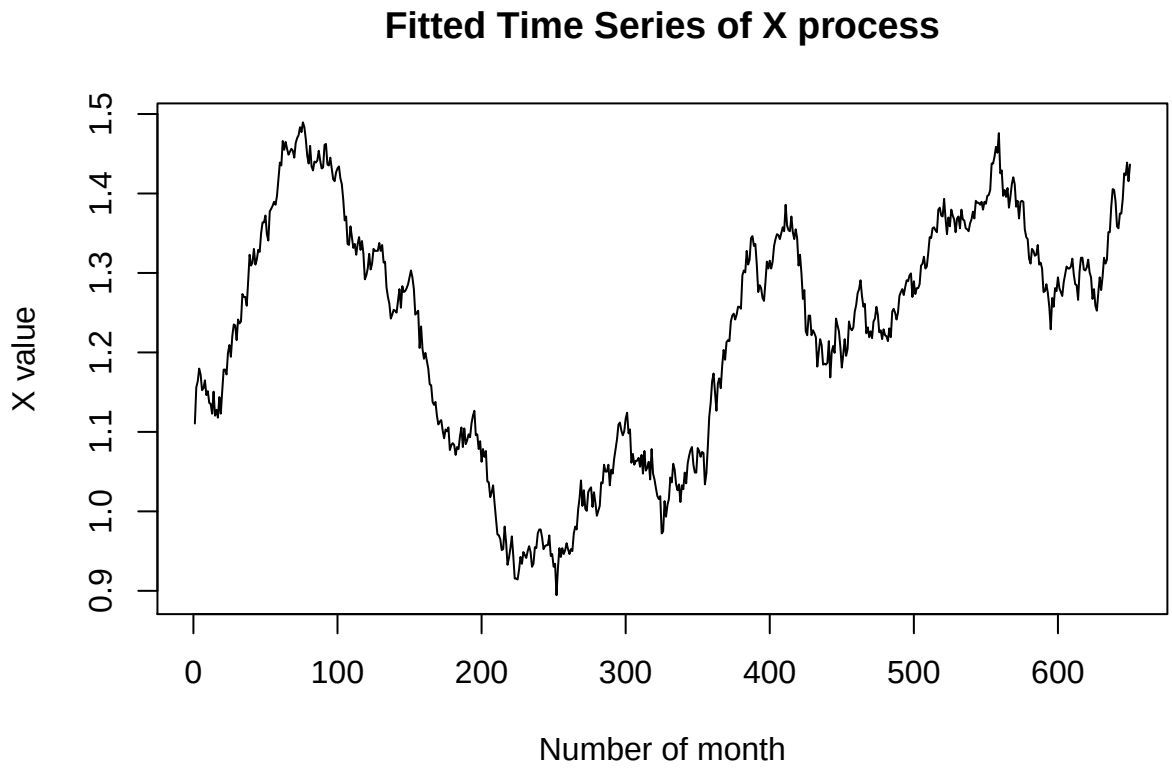
define a function to compute XYs from the optimized coeff

```
Compt_XYs=function(model_coeff,data){
  #compute A,B for 0.25 and 10 maturity
  ABx0.25=get_vasicek.AB(params = model_coeff[1:3],MT = 0.25)
  ABy0.25=get_vasicek.AB(params = c(0,model_coeff[4:5]),MT = 0.25)
  ABx10=get_vasicek.AB(params = model_coeff[1:3],MT = 10)
  ABy10=get_vasicek.AB(params = c(0,model_coeff[4:5]),MT = 10)
```

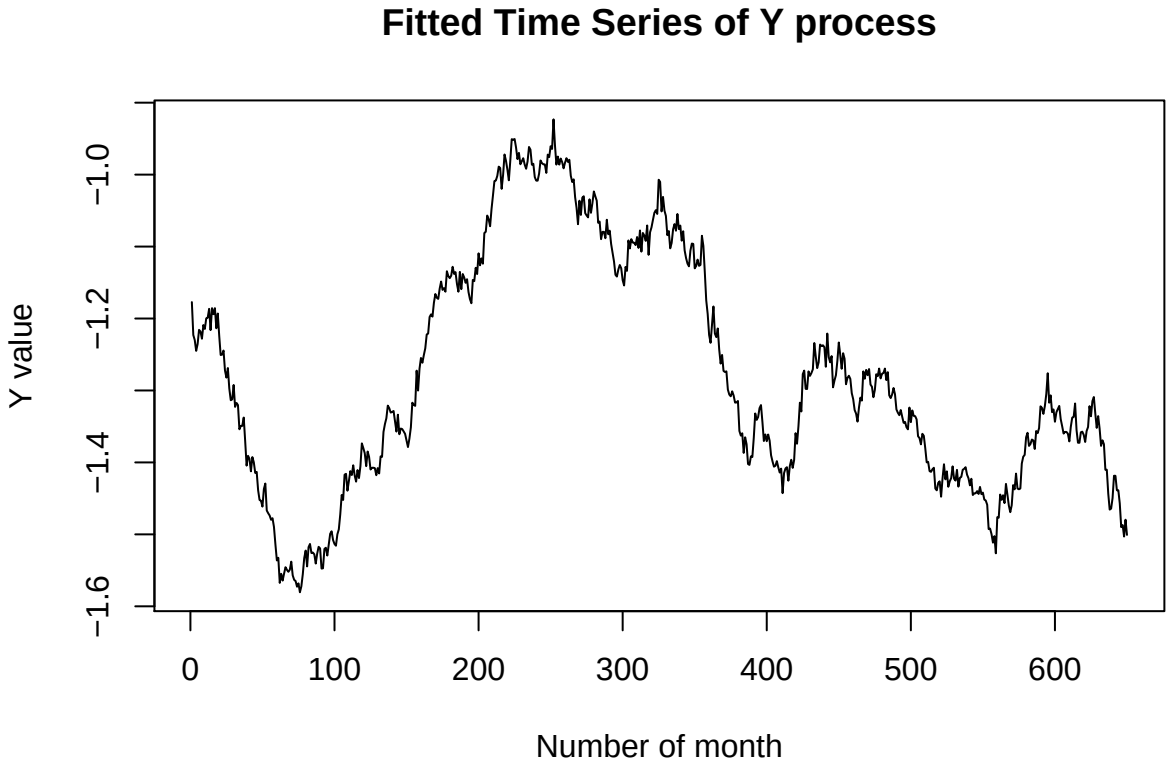
```
  ABs=rbind(ABx0.25,ABy0.25,ABx10,ABy10)
  #compute XY's
  XY_mat=get_all_XY(yields = data[,c(1,6)],ABs = ABs)
  return(XY_mat)}
#compute XY with the fitted parameters
coeff_obj$XYs=Compt_XYs(model_coeff = coeff_obj$par,data=Par_Rates)
```

plot the XY series

```
plot(x = seq(1,650),y=coeff_obj$XYs[,1],type='l',main='Fitted Time Series of X process',xlab = 'Number
```




```
plot(x = seq(1,650),y=coeff_obj$XYs[,2],type='l',main='Fitted Time Series of Y process',xlab = 'Number
```



compute the empirical and theoretical moment of X and Y

```
XY_moments=data.frame(matrix(0,nrow = 4,ncol = 2))
rownames(XY_moments)=c('EMMean','EMVariance','TheroMean','TheroVariance')
colnames(XY_moments)=c('X','Y')
XY_moments[1,]=colMeans(coeff_obj$XYs)
XY_moments[2,]=apply(coeff_obj$XYs, 2, FUN=var)
XY_moments[3,1]=coeff_obj$par[1]/coeff_obj$par[2]
XY_moments[3,2]=0
XY_moments[4,1]=coeff_obj$par[3]^2/(2*coeff_obj$par[2])
XY_moments[4,2]=coeff_obj$par[5]^2/abs(2*coeff_obj$par[4])
XY_moments
```

```
##                          X           Y
## EMMean         1.224534881 -1.28056554
## EMVariance     0.022355063  0.02621541
## TheroMean      1.561113367  0.00000000
## TheroVariance 0.003113586  0.16640124
```

6

```
#define a function to compute the fitted rates
get_fitted_rates=function(model_coeff,data,MTs){
  #compute A,B for 0.25 and 10 maturity
  ABx0.25=get_vasicek.AB(params = model_coeff[1:3],MT = 0.25)
  ABy0.25=get_vasicek.AB(params = c(0,model_coeff[4:5]),MT = 0.25)
  ABx10=get_vasicek.AB(params = model_coeff[1:3],MT = 10)
  ABy10=get_vasicek.AB(params = c(0,model_coeff[4:5]),MT = 10)
  ABs=rbind(ABx0.25,ABy0.25,ABx10,ABy10)
  #compute XY's
```

```
  XY_mat=get_all_XY(yields = data[,c(1,6)],ABs = ABs)
  #compute DT function
  DTs=get_all_DTs(params = model_coeff,XY_mat = XY_mat,MTs = seq(0.5,max(MTs),0.5))
  #compute fitted values
  fitted_par_Rates=get_fitted_par_rates(DTs = DTs,MTs = MTs)
  return(fitted_par_Rates)
}
#computed the fitted rates
OPT_fitted_rates=get_fitted_rates(model_coeff = coeff_obj$par,data = Par_Rates,MTs=c(2,3,5,7))
#compute the deviation b/w the observation and the fitted values
Dev_rates=Par_Rates[,c(-1,-ncol(Par_Rates))]-OPT_fitted_rates
```

fit an AR models on the series

```
AR_cmt2=arima(x = Dev_rates$cmt2,order = c(3,0,0))
print(AR_cmt2)
```

```
##
## Call:
## arima(x = Dev_rates$cmt2, order = c(3, 0, 0))
##
## Coefficients:
##          ar1     ar2     ar3  intercept
##       0.7601  0.1366  0.0779     0.0005
## s.e.  0.0392  0.0491  0.0393     0.0016
##
## sigma^2 estimated as 1.223e-06:  log likelihood = 3500.87,  aic = -6991.74
```

AR on cmt3 series

```
AR_cmt3=arima(x = Dev_rates$cmt3,order = c(2,0,0))
print(AR_cmt3)
```

```
##
## Call:
## arima(x = Dev_rates$cmt3, order = c(2, 0, 0))
##
## Coefficients:
##          ar1     ar2  intercept
##       0.8030  0.1755     0.0001
## s.e.  0.0388  0.0388     0.0017
##
## sigma^2 estimated as 1.041e-06:  log likelihood = 3553.22,  aic = -7098.44
```

AR on cmt5 series

```
AR_cmt5=arima(x = Dev_rates$cmt5,order = c(2,0,0))
print(AR_cmt5)
```

```
##
## Call:
## arima(x = Dev_rates$cmt5, order = c(2, 0, 0))
##
## Coefficients:
##          ar1     ar2  intercept
##       0.8222  0.1642     0.0000
## s.e.  0.0389  0.0389     0.0021
```

```
##
## sigma^2 estimated as 6.59e-07:  log likelihood = 3701.51,  aic = -7395.02
```

AR on cmt7 series

```
AR_cmt7=arima(x = Dev_rates$cmt7,order = c(2,0,0))
print(AR_cmt7)
```

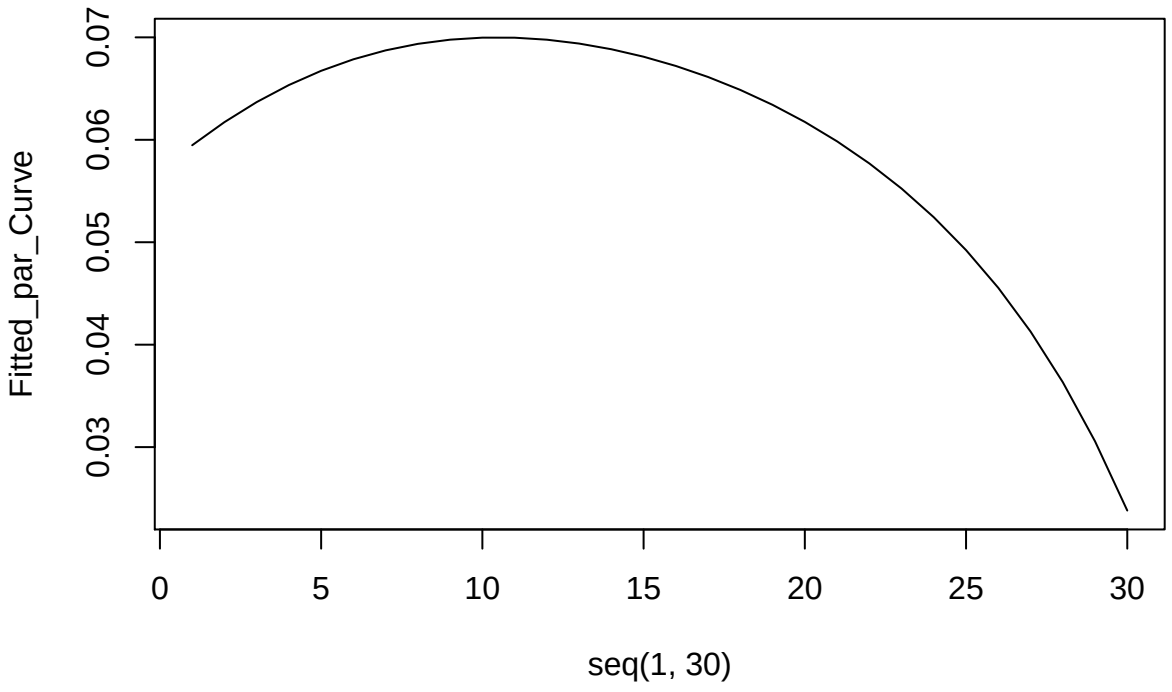
```
##
## Call:
## arima(x = Dev_rates$cmt7, order = c(2, 0, 0))
##
## Coefficients:
##          ar1     ar2  intercept
##       0.8165  0.1722     0.0000
## s.e.  0.0388  0.0389     0.0017
##
## sigma^2 estimated as 3e-07:  log likelihood = 3957.17,  aic = -7906.34
```

For all of the deviations in the four series, we can fit a AR(2) or AR(3) model with statistcially significant coefficients, which is equivalently to say there is autocorrelation in the residuals of our two-factor Vasicek model. This result implies that there are still some predictive power from lagged series that two-factor Vasicek model does not extrasct. Based on this result, we recommend to include another time-series component in the orginial model to increased the predictive power.

7

compute fitted par curves for T=1:30, we take the mean of the fitted values as estimated yields for par bond with maturity from 1 to 30 years

```
#compute fitted par curves for T=1:30, we take the mean of the fitted values as estimated yields for pa
Fitted_par_Curve=colMeans(get_fitted_rates(model_coeff = coeff_obj$par,data = Par_Rates,MTs = seq(1,30))
plot(x=seq(1,30),y=Fitted_par_Curve,type='l')
```



```
bond_data=data.frame(matrix(data = 0,nrow = 30,ncol = 3))
rownames(bond_data)=seq(1,30)
colnames(bond_data)=c('Coupon','Maturity','Yield')
#note the corp bond paid coupon semi-annually but the 10-yr bond paid coupon annully
bond_data$Coupon=Fitted_par_Curve*100
bond_data$Maturity=seq(1,30)
bond_data$Yield=Fitted_par_Curve
```

define a function to compute duration and convexity

```
get_price_ModiD_Conv=function(df)
  {for (i in 1:nrow(df))
    {
    #compute bond price
    df$Price[i]=bondpv(coupon = df$Coupon[i],mat = df$Maturity[i],yield = df$Yield[i],principal = 100,f
    #compute bond modified duration
    df$ModiD[i]=duration(price = df$Price[i],coupon = df$Coupon[i],mat = df$Maturity[i],principal = 100
    #compute bond convexity
    df$Conv[i]=convexity(price = df$Price[i],coupon = df$Coupon[i],mat = df$Maturity[i],principal = 100
  }
  return (df)}
bond_data=get_price_ModiD_Conv(bond_data)
```

define a function to compute the duration and conveixty hedging ratio, with two and ten year par bonds. To get the hedge ratio, we need to solve this two system of equation in matrix form

$$\begin{pmatrix} -D_2^* & -D_{10}^* \\ C_2 & C_{10} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -D_t^* \\ C_t \end{pmatrix}$$

where the vector $\begin{pmatrix} x \\ y \end{pmatrix}$ is the weights we want solve for bond with different maturity.

```
get_hedge_weights=function(df){
  #initialize a weightes matrix
  weights_mat=data.frame(matrix(0,nrow = nrow(df),ncol = 2))
  colnames(weights_mat)=c('Two year bond','Ten year bond')
    # construct A matrix and b vector
  A_mtx=as.matrix(df[c(2,10),5:6])
  A_mtx=t(A_mtx)
  A_mtx[1,]=-A_mtx[1,]
  for (i in 1: nrow(weights_mat)){
    b=c(-df$ModiD[i],df$Conv[i])
    weights_mat[i,]=solve(a = A_mtx,b = b)
  }

  return(weights_mat)
}

DurConvHege_weights=get_hedge_weights(df=bond_data)
```

For XY hedge, we first define a function to compute the gradients on X and Y

```
#first we define a function to compute the Bs
get_all_Bs=function(model_coeff,MTs)
{
  Bs=data.frame(matrix(0,nrow = 30,ncol = 2))
```

```
  colnames(Bs)=c('Bx','By')
  #compute Bs
  for (t in 1:length(MTs))
    {
    Bs[t,1]=get_vasicek.AB(params = model_coeff[1:3],MT = t)[2]
    Bs[t,2]=get_vasicek.AB(params =c(0,model_coeff[4:5]),MT = t)[2]
  }
  return(Bs)
}
#compute Bx By
Bxy=get_all_Bs(model_coeff = coeff_obj$par,MTs = seq(0.5,30,0.5))
#define a function to compute DTs from the fitted coeffs
get_DTs=function(model_coeff,data,MTs){
  #compute A,B for 0.25 and 10 maturity
  ABx0.25=get_vasicek.AB(params = model_coeff[1:3],MT = 0.25)
  ABy0.25=get_vasicek.AB(params = c(0,model_coeff[4:5]),MT = 0.25)
  ABx10=get_vasicek.AB(params = model_coeff[1:3],MT = 10)
  ABy10=get_vasicek.AB(params = c(0,model_coeff[4:5]),MT = 10)
  ABs=rbind(ABx0.25,ABy0.25,ABx10,ABy10)
  #compute XY's
  XY_mat=get_all_XY(yields = data[,c(1,6)],ABs = ABs)
  #compute DT function
  DTs=get_all_DTs(params = model_coeff,XY_mat = XY_mat,MTs = seq(0.5,max(MTs),0.5))
  DTs=colMeans(DTs)
  return(DTs)
}
DTs=get_DTs(model_coeff = coeff_obj$par,data = Par_Rates,MTs = seq(1,30))
#define the function to compute the gradients
compute_XY_gradients=function(bond_data,DTs,Bxy)
{
  bond_data$Xgrad=0
  bond_data$Ygrad=0
  for(i in 1:nrow(bond_data)){
    bond_data$Xgrad[i]=-(bond_data$Coupon[i]/2)*sum((DTs[1:(2*i)]*Bxy[1:(2*i),1]))-DTs[(2*i)]*Bxy[(2*i)
    bond_data$Ygrad[i]=-(bond_data$Coupon[i]/2)*sum((DTs[1:(2*i)]*Bxy[1:(2*i),2]))-DTs[(2*i)]*Bxy[(2*i)
  }
 return(bond_data)
}
#compute the gradients
bond_data=compute_XY_gradients(bond_data = bond_data,DTs = DTs,Bxy = Bxy)
```

Finally, define a function to compute the duration and conveixty hedging ratio, with two and ten year par bonds. To get the hedge ratio, we need to solve this two system of equation in matrix form

$$\begin{pmatrix} Px_2 & Px_{10} \\ Py_2 & Py_{10} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} Px_t \\ Py_t \end{pmatrix}$$

where the vector $\begin{pmatrix} w1 \\ w2 \end{pmatrix}$ is the weights we want solve for bond with different maturity.

```
get_grad_hedge_weights=function(df){
  #initialize a weightes matrix
  weights_mat=data.frame(matrix(0,nrow = nrow(df),ncol = 2))
  colnames(weights_mat)=c('Two year bond','Ten year bond')
    # construct A matrix and b vector
```

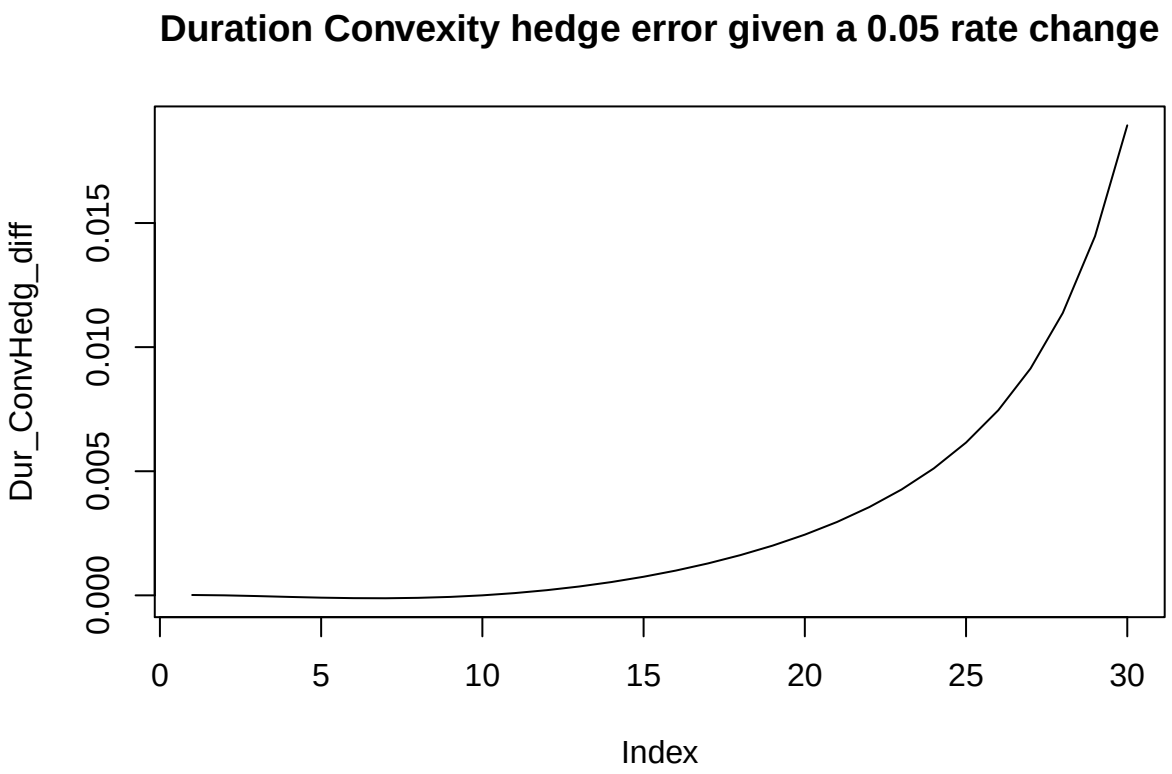
```
  A_mtx=as.matrix(df[c(2,10),c('Xgrad','Ygrad')])
  A_mtx=t(A_mtx)
  for (i in 1: nrow(weights_mat)){
    b=c(df$Xgrad[i],df$Ygrad[i])
    weights_mat[i,]=solve(a = A_mtx,b = b)
  }
  return(weights_mat)
}
grad_hedge=get_grad_hedge_weights(df=bond_data)
```

To compare the effect of these two hedge approaches, we need to define a function to compute the value of hedge protfolio after a rate change and compare the result.

```
compute_Port_value=function(hedge_ratio,bond_data){
    hedge_ratio=cbind(hedge_ratio,-1)
    ini_Port_value=as.matrix(hedge_ratio)%*%rep(100,3)
    up_bond=bond_data[,1:3]
    up_bond$Yield=bond_data$Yield+0.005
    up_bond=get_price_ModiD_Conv(df=up_bond)
    #compute the value difference in portfolio
    for (t in 1:nrow(bond_data))
      {
    up_bond$after[t]=as.matrix(hedge_ratio[t,])%*%c(up_bond$Price[2],up_bond$Price[10],up_bond$Price[t]
    }
    up_diff=up_bond$after-ini_Port_value
    return(up_diff)
}
```

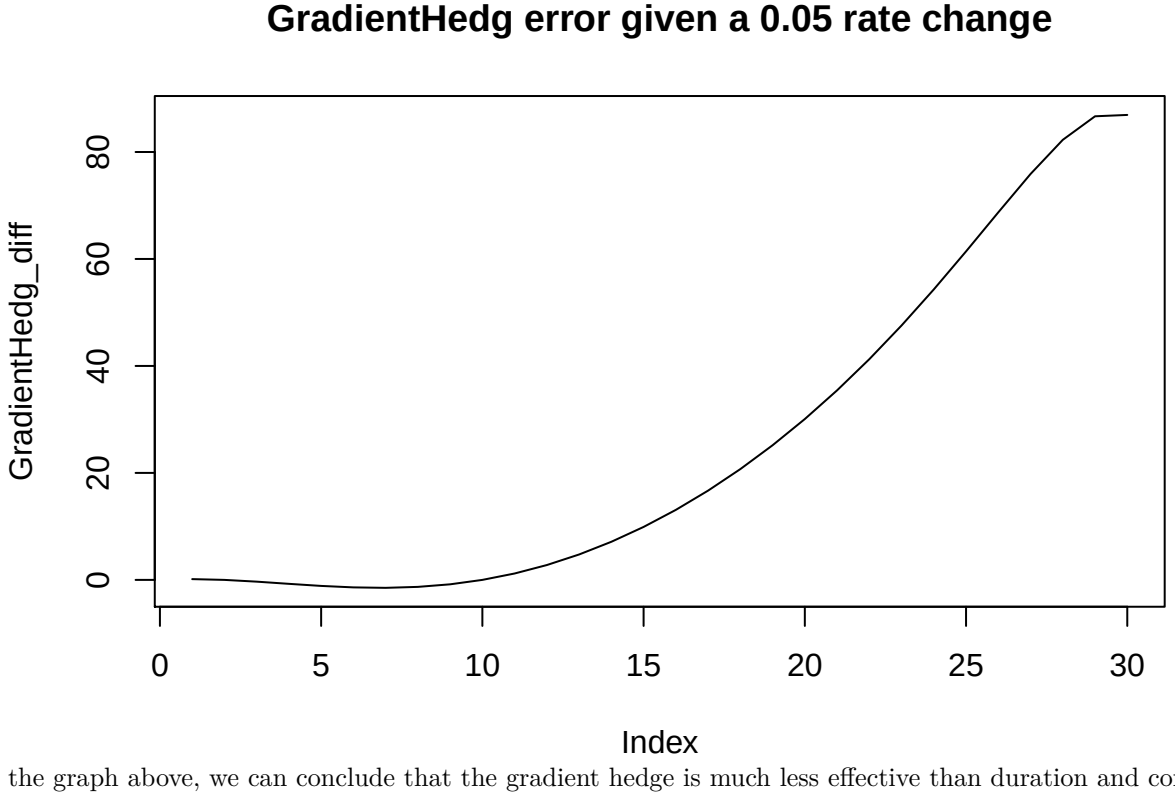
Compute and plot the Duration Convexity Hedging Error

```
Dur_ConvHedg_diff=compute_Port_value(hedge_ratio = DurConvHege_weights,bond_data = bond_data)
plot(Dur_ConvHedg_diff,type='l',main='Duration Convexity hedge error given a 0.05 rate change')
```



Compute and plot the X,Y gradients Hedging Error

```
GradientHedg_diff=compute_Port_value(hedge_ratio = grad_hedge,bond_data = bond_data)
plot(GradientHedg_diff,type='l',main='GradientHedg error given a 0.05 rate change')
```



From the graph above, we can conclude that the gradient hedge is much less effective than duration and convexity hedge.