Trabalho 1 - Método Numéricos (PEC00141)

1. Dados

Para a estrutura ilustrada na figura abaixo: As áreas das barras A e B são 10E03 mm² e 15E03 mm². A área da barra C pode variar entre 0 e 40E03 mm². Fazer um gráfico indicando a variação de força em cada barra em função da área da barra C. Considerando que o mesmo material é empregado para as três barras e que este apresenta tensão admissível de 140 MPa, para qual valor de área da barra C se apresenta a maior relação entre capacidade de carga e peso próprio. Adotou-se E=200GPa e P=1kN

2. Inserindo os dados

Importação de módulos que serão utilizados

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Há 3 barras e 4 nós e 8 gdl (2 por nó). A numeração dos graus de liberdade é feita nó a nó, onde **a direção x tem o primeiro gdl do nó e a direção y o segundo**, dessa forma, a numeração dos gdls para o nó n é dada por:

$$gdl_h = 2n - 1 \quad (1)$$
$$gdl_v = 2n \quad (2)$$

Importação do excel (as unidades estão em metros e em Newtons):

In [2]:
```
nos = pd.read_excel('dados_de_entrada.xlsx', sheet_name='Nós')
barras = pd.read_excel('dados_de_entrada.xlsx', sheet_name='Barras')

# Acertando os números dos nós e das barras conforme a figura (iniciando em 1)
nos.index += 1
barras.index += 1

# Trocando celulas vazias por zeros
nos.fillna(0, inplace=True)

# Printando os valores na tela
nos
```

Out[2]:

	X	Y	RX	RY	FX	FY	dx	dy
1	0	8.660254	1.0	1.0	0.0	0.0	0.0	0.0
2	5	8.660254	0.0	0.0	0.0	-1000.0	0.0	0.0
3	0	3.660254	1.0	1.0	0.0	0.0	0.0	0.0
4	0	0.000000	1.0	1.0	0.0	0.0	0.0	0.0

In [3]:
```
barras
```

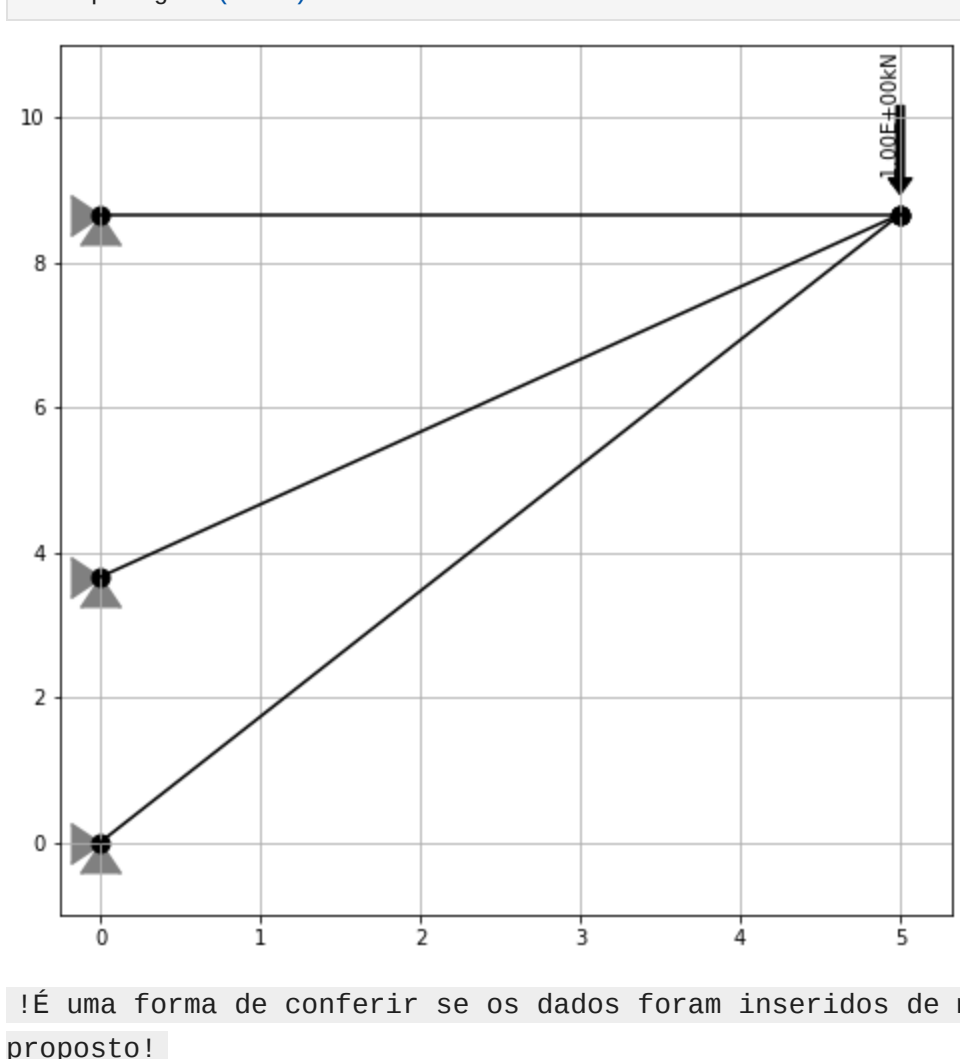
Out[3]:

	N1	N2	A	E
1	1	2	0.010	200000000000
2	2	3	0.015	200000000000
3	2	4	0.040	200000000000

3. Plotagem da estrutura

O código abaixo permite a montagem da estrutura plotando elemento a elemento a partir das matrizes das barras e dos nós.

In [4]:
```
plt.figure(1,figsize=(8,8))
plt.ylim(-1,11)

# Plotagem dos apoios e das forças
for no in nos.index:
    X,Y,RX,RY,Fx,Fy,dx,dy = nos.loc[no]  #percorre os valores de X,Y,RX,RY,Fx,Fy,dx,dy na tabela para cada no do for (1,2,3 e 4)

    # Aplicação dos vínculos (Se RX restrito aplica sobre gl de X)
    if RX == 1:
        plt.scatter(X,Y,400,marker =5,zorder = -2,color ='gray')
    if RY == 1:
        plt.scatter(X,Y,400,marker =6,zorder = -2,color ='gray')

    # Aplicação das forças
    if Fx >0:
        plt.arrow(X-1.5,Y,1,0,width =0.05,color='k')
        plt.text(X-1.5,Y,'{:.2E}kN'.format(Fx/1000),va='bottom')
    if Fx <0:
       plt.arrow(X+1.5,Y,-1,0,width =0.05,color='k')
       plt.text(X+.5,Y,'{:.2E}kN'.format(Fx/1000),va='bottom')
    if Fy >0:
        plt.arrow(X,Y-1.5,0,1,width =0.05,color='k')
        plt.text(X,Y,'{:.2E}kN'.format(Fy/1000),va='bottom',rotation=90)
    if Fy <0:
       plt.arrow(X,Y+1.5,0,-1,width =0.05,color='k')
       plt.text(X,Y+.5,'{:.2E}kN'.format(Fy/1000),ha='right',rotation=90)

    # Aplicação dos deslocamentos
    if dx != 0:
        plt.text(X+0.5,Y-0.5,'Δx={}m'.format(dx),color='black')
    if dy != 0:
        plt.text(X+0.5,Y-0.5,'Δy={}m'.format(dy),color='black')

# Plotagem das barras
for barra in barras.index:
    # Vamos passar os nós para as variáveis N1 e N2
    N1 = barras.loc[barra, 'N1']
    N2 = barras.loc[barra, 'N2']

    # Agora vamos acessar as coordendas de cada um dos nós
    x1, y1 = nos.loc[N1, ['X','Y']]
    x2, y2 = nos.loc[N2, ['X','Y']]
    y = [y1,y2]
    x = [x1,x2]

    plt.plot(x,y,'black')
    plt.scatter(x,y, s=80,marker ='o',color ='black')
    plt.grid(True)
```



`!É uma forma de conferir se os dados foram inseridos de maneira correta, ou seja, a imagem gerada deve ser compatível com a do problema proposto!`

4. Comprimento das barras e cos e sen diretores

Percorre-se o **DataFrame** de `barras` e a cada passo acessa o **DataFrame** de `nos` para obter suas coordendas. Os valores cálculados serão armazenados em **listas** que serão inseridas no **DataFrame** ao fim do processo.

In [5]:
```
# Criação de listas vazias para armazenar as variáves
Ls   = []
sens = []
coss = []

for barra in barras.index:
    # Vamos passar os nós para as variáveis N1 e N2. Por exemplo: quando  barra=4 (no for), N1=3 e N2=4
    N1 = barras.loc[barra, 'N1']
    N2 = barras.loc[barra, 'N2']

    # Agora vamos acessar as coordendas de cada um dos nós. Por exemplo: quando  barra=4 (no for), x1=5, y1=8.66, x2=10, y2=8.66
    x1, y1 = nos.loc[N1, ['X','Y']]
    x2, y2 = nos.loc[N2, ['X','Y']]

    # O comprimento da barra é dado pelo teorema de Pitagoras
    Lx = x2 - x1
    Ly = y2 - y1
    L = np.sqrt(Lx**2 + Ly**2)

    # Os cossenos diretores são então:
    sen = Ly/L
    cos = Lx/L

    # Inserindo nas listas
    Ls.append(L)
    sens.append(sen)
    coss.append(cos)

# Agora que saimos do loop vamos inserir no DataFrame
barras['L']   = Ls
barras['sen'] = sens
barras['cos'] = coss

# Printando novo DataFrame
barras
```

Out[5]:

	N1	N2	A	E	L	sen	cos
1	1	2	0.010	200000000000	5.000000	0.000000	1.000000
2	2	3	0.015	200000000000	7.071068	-0.707107	-0.707107
3	2	4	0.040	200000000000	10.000000	-0.866025	-0.500000

5. Montagem da matriz de rigidez

A matriz de rigidez global é a obtida pela superposição das matrizes de rigidez locais nos respectivos graus de liberdade. Inicialmente a matriz global `K` deve ser pré alocada como uma matriz de zeros.

In [6]:
```
maxgl = 2*len(nos)
K = np.zeros([maxgl,maxgl])
```

A alocação é realizada percorrendo todas as barras novamente, cálculando suas matrizes de rigidez local e alocando suas componentes na matriz global.

In [7]:
```
for barra in barras.index:
    # Vamos importar as propriedades necessárias para construção da matriz local e da matriz de rotação
    L   = barras.loc[barra, 'L']
    sen = barras.loc[barra, 'sen']
    cos = barras.loc[barra, 'cos']
    A   = barras.loc[barra, 'A']
    E   = barras.loc[barra, 'E']
    N1  = barras.loc[barra, 'N1']
    N2  = barras.loc[barra, 'N2']

    # Matriz de rigidez no sistema local
    Kl = E*A/L*np.array([[ 1, 0,-1, 0],
                         [ 0, 0, 0, 0],
                         [-1, 0, 1, 0],
                         [ 0, 0, 0, 0]])

    # Matriz de rotação
    Mrot = np.array([[ cos,  sen,    0,   0],
                     [-sen,  cos,    0,   0],
                     [    0,   0,  cos, sen],
                     [    0,   0, -sen, cos]])

    # Rotação da matriz de coordenadas locais para globais, o np.dot é para multiplicação das matrizes
    Klr = np.dot(np.dot(Mrot.T, Kl), Mrot)

    # Cálculo dos graus de liberdade
    gl1 = 2*N1 - 1
    gl2 = 2*N1
    gl3 = 2*N2 - 1
    gl4 = 2*N2

    # Aloca a matriz local na matriz global
    # Lembrando as propriedades das listas do Python!
    K[gl1-1:gl2, gl1-1:gl2] += Klr[0:2, 0:2]
    K[gl3-1:gl4, gl1-1:gl2] += Klr[2:4, 0:2]
    K[gl1-1:gl2, gl3-1:gl4] += Klr[0:2, 2:4]
    K[gl3-1:gl4, gl3-1:gl4] += Klr[2:4, 2:4]
```

6. Aplicação das condições de contorno

Agora, para solução do problema, a matriz de rigidez global é clonada e as restrições de apoio são impostas zerando as respectivas linhas e colunas.

In [8]:
```
Kcompleta = K.copy()    # Alocando a matriz em outro espaço de memória, pois vou mudar o K,  as será "conservado" em Kcompleta

for no in nos.index:
    RX, RY, dx, dy = nos.loc[no, ['RX','RY','dx','dy']]
    # Se RX restrito aplica sobre gl de X
    if RX == 1:
        gl = 2*no - 1
        K[:,    gl-1] = 0
        K[gl-1,    :] = 0
        K[gl-1, gl-1] = 1
        print('Aplicando restrição horizontal no nó {:d}.'.format(no))

    if RY == 1:
        gl = 2*no
        K[:,    gl-1] = 0
        K[gl-1,    :] = 0
        K[gl-1, gl-1] = 1
        print('Aplicando restrição vertical no nó {:d}.'.format(no))

    # Restrições de deslocamento
    if dx != 0:
        gl = 2*no - 1
        K[:,    gl-1] = 0
        K[gl-1,    :] = 0
        K[gl-1, gl-1] = 1
        print('Aplicando restrição horizontal devido ao deslocamento no nó {:d}.'.format(no))

    if dy != 0:
        gl = 2*no
        K[:,    gl-1] = 0
        K[gl-1,    :] = 0
        K[gl-1, gl-1] = 1
        print('Aplicando restrição vertical devido ao deslocamento no nó {:d}.'.format(no))
```

```
Aplicando restrição horizontal no nó 1.
Aplicando restrição vertical no nó 1.
Aplicando restrição horizontal no nó 3.
Aplicando restrição vertical no nó 3.
Aplicando restrição horizontal no nó 4.
Aplicando restrição vertical no nó 4.
```

In [9]:
```
#print (Kcompleta,'\n')
#print (K)
```

6.1 Montagem do vetor de forças

Com a matriz global restringida vamos montar o vetor de forças aplicadas. Este vetor tem como dimensão o número de graus de liberdade.

In [10]:
```
F = np.zeros(maxgl)
Fauxf = np.zeros(maxgl)  #vetor das forças aplicadas
Fauxd = np.zeros(maxgl)  #vetor das forças devido aos deslocamentos sofridos
Fauxdx = np.zeros(maxgl)  #para contar
Fauxdy = np.zeros(maxgl)  #para contar

for no in nos.index:
    RX, RY, FX, FY, dx, dy = nos.loc[no, ['RX','RY','FX','FY','dx','dy']]

    gl1 = 2*no - 1
    gl2 = 2*no

    if FX != 0:
        Fauxf[gl1-1] = FX
    if FY != 0:
        Fauxf[gl2-1] = FY

    if dx != 0:
        Fauxdx = -Kcompleta[gl1-1, :]*dx
        Fauxdx[gl1-1]=dx

    if dy != 0:
        Fauxdy = -Kcompleta[gl2-1, :]*dy
        Fauxdy[gl2-1]=dy

    Fauxd =(Fauxdx+Fauxdy)

    if RX == 1:
        Fauxd[gl1-1]=0
    if RY == 1:
        Fauxd[gl2-1]=0

F=Fauxf+Fauxd
print(F)
```

```
[    0.     0.     0. -1000.     0.     0.     0.     0.]
```

7. Resolução da equação de equilíbrio

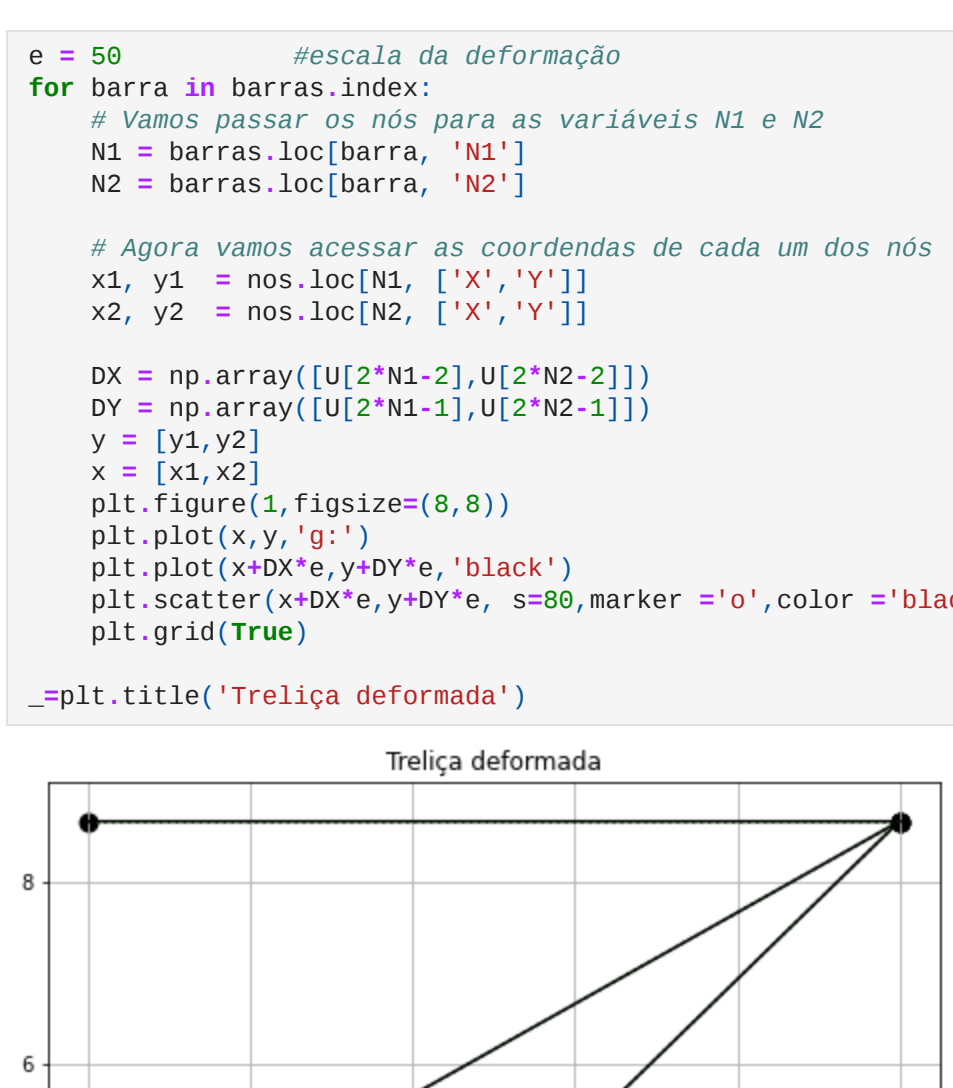
In [11]:
```
U = np.linalg.solve(K, F)
print (U)
```

```
[ 0.00000000e+00  0.00000000e+00  1.60690386e-06 -2.33647182e-06
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
```

8. Plotagem da treliça deformada

Em posse dos valores de deslocamento, é possível realizar a plotagem da treliça deslocada, junto com suas reações de apoio.

In [12]:
```
e = 50          #escala da deformação
for barra in barras.index:
    # Vamos passar os nós para as variáveis N1 e N2
    N1 = barras.loc[barra, 'N1']
    N2 = barras.loc[barra, 'N2']

    # Agora vamos acessar as coordendas de cada um dos nós
    x1, y1  = nos.loc[N1, ['X','Y']]
    x2, y2  = nos.loc[N2, ['X','Y']]

    DX = np.array([U[2*N1-2],U[2*N2-2]])
    DY = np.array([U[2*N1-1],U[2*N2-1]])
    y = [y1,y2]
    x = [x1,x2]
    plt.figure(1,figsize=(8,8))
    plt.plot(x,y,'g:')
    plt.plot(x+DX*e,y+DY*e,'black')
    plt.scatter(x+DX*e,y+DY*e, s=80,marker ='o',color ='black')
    plt.grid(True)

_=plt.title('Treliça deformada')
```



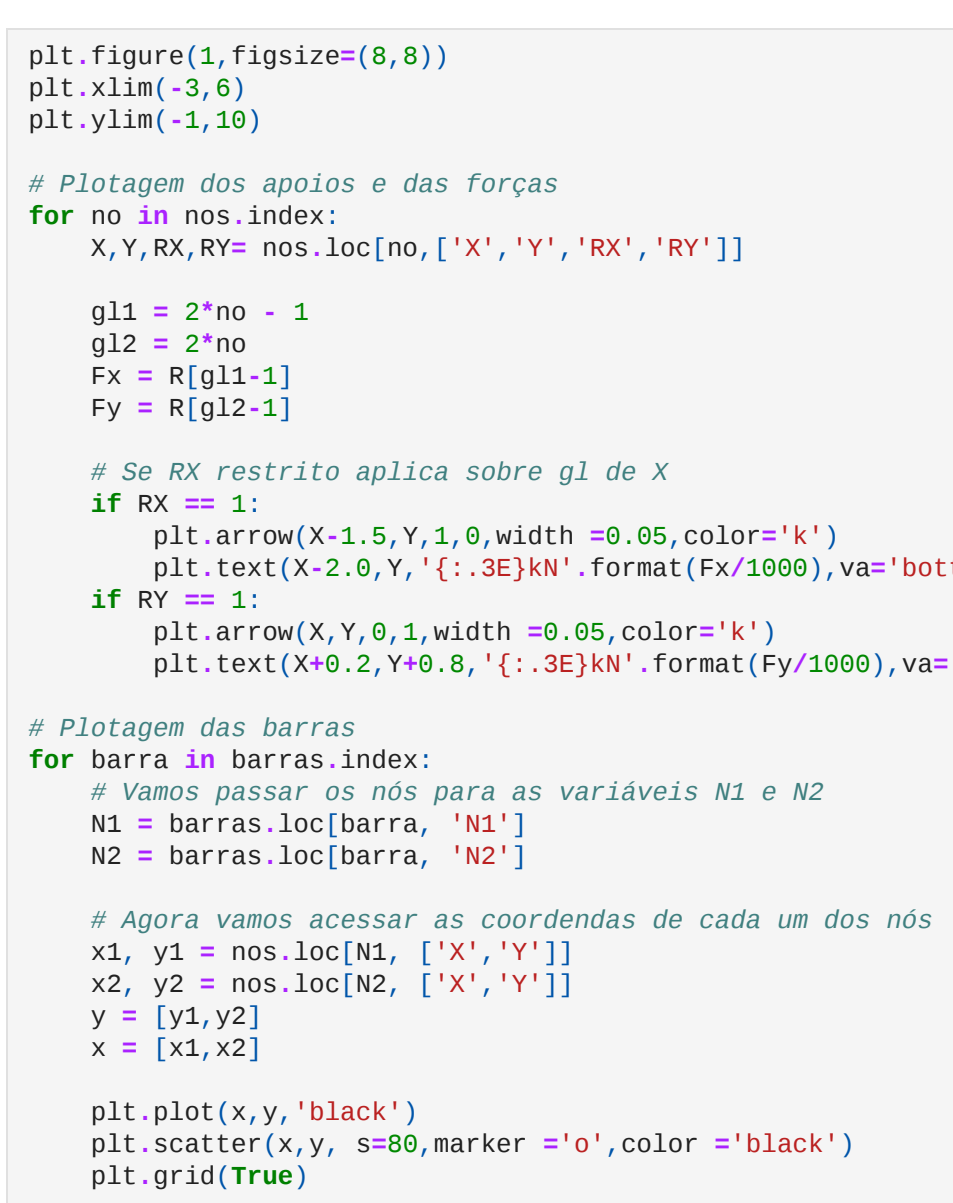
O cálculo das reações de apoio é feito a partir do produto da matriz de rigidez pelos deslocamentos.

In [13]:
```
R = np.dot(Kcompleta,U)
print(R)
```

```
[ -642.76154288     0.             0.         -1000.
   154.76473525   154.76473525   487.99680763   845.23526475]
```

Agora, para a realização da plotagem, é necessário escolher apenas os graus de liberdade que são restritos.

In [14]:
```
plt.figure(1,figsize=(8,8))
plt.xlim(-3,6)
plt.ylim(-1,10)

# Plotagem dos apoios e das forças
for no in nos.index:
    X,Y,RX,RY= nos.loc[no,['X','Y','RX','RY']]

    gl1 = 2*no - 1
    gl2 = 2*no
    Fx = R[gl1-1]
    Fy = R[gl2-1]

    # Se RX restrito aplica sobre gl de X
    if RX == 1:
        plt.arrow(X-1.5,Y,1,0,width =0.05,color='k')
        plt.text(X-2.0,Y,'{:.3E}kN'.format(Fx/1000),va='bottom')
    if RY == 1:
        plt.arrow(X,Y,0,1,width =0.05,color='k')
        plt.text(X+0.2,Y+0.8,'{:.3E}kN'.format(Fy/1000),va='bottom')

# Plotagem das barras
for barra in barras.index:
    # Vamos passar os nós para as variáveis N1 e N2
    N1 = barras.loc[barra, 'N1']
    N2 = barras.loc[barra, 'N2']

    # Agora vamos acessar as coordendas de cada um dos nós
    x1, y1 = nos.loc[N1, ['X','Y']]
    x2, y2 = nos.loc[N2, ['X','Y']]
    y = [y1,y2]
    x = [x1,x2]

    plt.plot(x,y,'black')
    plt.scatter(x,y, s=80,marker ='o',color ='black')
    plt.grid(True)
```



9. Determinação dos esforços

Para finalizar a análise, é interessante o cálculo dos esforços atuantes em cada elemento, para isso, realiza-se o script abaixo:

In [15]:
```
Esf = []

for barra in barras.index:
    # Vamos importar as propriedades necessárias para construção da matriz local e da matriz de rotação
    L   = barras.loc[barra, 'L']
    sen = barras.loc[barra, 'sen']
    cos = barras.loc[barra, 'cos']
    A   = barras.loc[barra, 'A']
    E   = barras.loc[barra, 'E']
    N1  = barras.loc[barra, 'N1']
    N2  = barras.loc[barra, 'N2']

    # Matriz de rigidez no sistema local
    Kl = E*A/L*np.array([[ 1, 0,-1, 0],
                         [ 0, 0, 0, 0],
                         [-1, 0, 1, 0],
                         [ 0, 0, 0, 0]])

    # Matriz de rotação
    Mrot = np.array([[ cos,  sen,    0,   0],
                     [-sen,  cos,    0,   0],
                     [    0,   0,  cos, sen],
                     [    0,   0, -sen, cos]])
    # Recebendo os deslocamentos referentes ao elemento em análise
    Ul = np.zeros(4)
    Ul[0] = U[2*N1-2]
    Ul[1] = U[2*N1-1]
    Ul[2] = U[2*N2-2]
    Ul[3] = U[2*N2-1]

    # Realizando o equilíbrio local

    F = np.dot(Kl,np.dot(Mrot,Ul))

    # Salvando o terceiro valor do vetor de forças por convenção de sinais.
    Esf.append(F[2])
print (Esf)
```

```
[642.7615428823119, -218.87038756875157, -975.9936152630395]
```

Agora, vamos plotar os esforços, sinalizando as barras comprimidas pela cor azul e as tracionadas pela cor vermelha.

In [16]:
```
for barra in barras.index:
    # Vamos passar os nós para as variáveis N1 e N2
    N1 = barras.loc[barra, 'N1']
    N2 = barras.loc[barra, 'N2']
    ax = Esf[barra-1]
    cos = barras.loc[barra, 'cos']
    sen = barras.loc[barra, 'sen']
    tg  = sen/cos
    # Agora vamos acessar as coordendas de cada um dos nós
    x1, y1  = nos.loc[N1, ['X','Y']]
    x2, y2  = nos.loc[N2, ['X','Y']]
    y = [y1,y2]
    x = [x1,x2]

    plt.figure(1,figsize=(8,8))
    if ax>0:
        cor = 'r'
    elif ax == 0:
        cor = 'k'
    else:
        cor = 'b'
    plt.plot(x,y,cor,zorder = -1)

    plt.text(np.mean(x),np.mean(y),'{:.2f}kN'.format(ax/1000),rotation =180*np.arctan(tg)/np.pi,
            horizontalalignment='center',
            verticalalignment='center',
            size = 16,
            weight = 'bold')

    plt.scatter(x,y, s=80,marker ='o',color ='black')

_=plt.title('Esforços atuantes')
plt.grid(True)
```