Table of Contents

In [34]:

```
%matplotlib inline
%run ../starter.py
from PIL import Image #We'll have some fun today ;)
```

0.1 Pràctica 1: Índex de refracció no uniforme

Nota: *En aquesta pràctica s'utilitza el cm com a unitat base de longitud:*

A la següent pràctica de laboratori tindrem com a objectiu el avaluar el comportament d'un feix de làser a través d'un medi amb un índex de refracció no uniforme.

Per a fer-ho utilitzarem el següent set-up experimental:

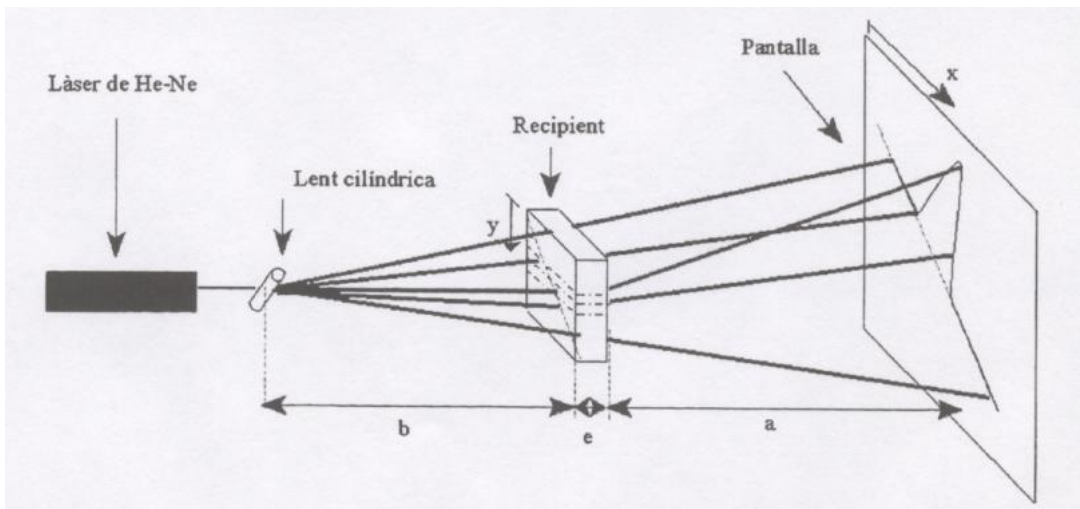


Figura 4. Esquema del muntatge experimental a realitzar

On less variables mostrades a la figura tenen per valor:

In [35]:

```
A = uc.ufloat(120, .1)
B = uc.ufloat( 80, .1)
E = uc.ufloat(2.5, .1)
N_H2O = 1.33
N_ETA = 1.36
```

Captarem les imatges amb PIL, detectarem la figura, en píxels. Aquestes dades després s'hauran de passar a centímetres amb els factors de conversió que trobem.

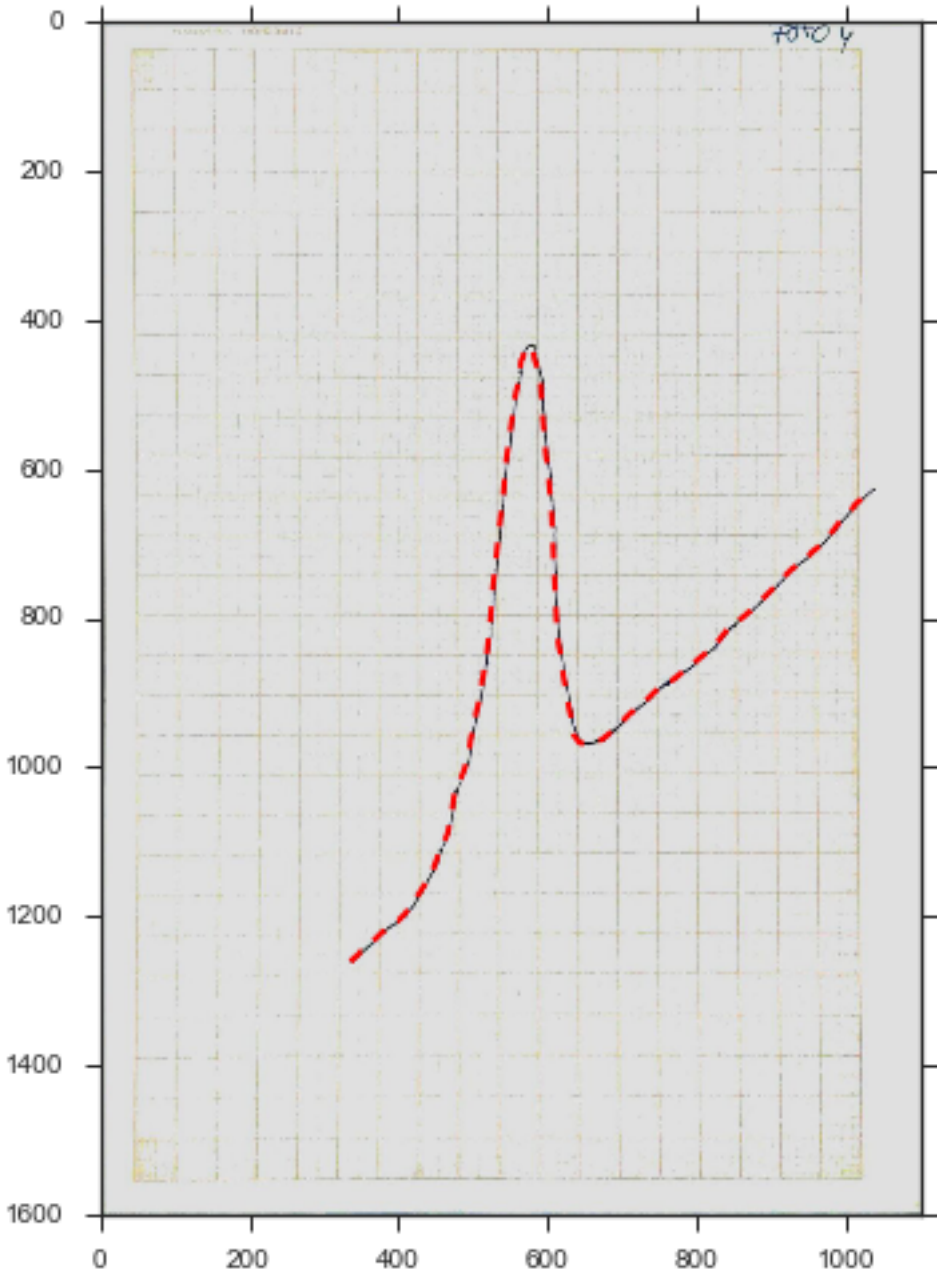
In [36]:

```
def curveFinder(pil_im, k=1, k2=3):
    pil_im = pil_im.split()[0]
```

```
    pil_im = np.asarray(pil_im)
    pil_im = np.array(pil_im)
    pil_im = np.cumsum(pil_im, axis=0)
    pil_im = -pil_im[0:-k,:]+pil_im[k:,:]
    y = np.argmax(pil_im, axis=0)
    selector = np.max(pil_im, axis=0)>5
    selector = selector[k2:]
    y = np.cumsum(y)
    y = (-y[0:-k2]+y[k2:])/k2
    x = np.array(range(0,len(y)))+k2
    return {'X':x[selector][k2:],'Y':y[selector][k2:]}

def photoPlot(k):
    pil_im0=Image.open('IN/IMG/clean{:02d}.jpg'.format(k))
    pil_im1=Image.open('IN/IMG/raw{:02d}.jpg'.format(k))
    plt.imshow(pil_im1, animated=True)
    cf =curveFinder(pil_im0)
    plt.plot(cf['X'], cf['Y'],linestyle='--', color='red', lw=2, alpha=1)

fig=plt.figure(figsize=(11./2,16./2) )
photoPlot(4)
plt.xlim(0,1100)
plt.ylim(1600,0)
```

Out[36]:

(1600, 0)



In [37]:

```
images=[Image.open('IN/IMG/clean{:02d}.jpg'.format(k)) for k in range(1,4+1)]
curve_data = [curveFinder(im) for im in images]
```

Per a passar els píxels a centímetres creem un array 2-dimensional. Cada fila de l'array correspondrà a una mesura, i contindrà les coordenades X dels extrems del gràfic i les coordenades Y dels extrems del gràfic. A partir d'això utilitzant que la part del gràfic fa 18 x 28 cm, trobarem la correspondència:

In [38]:

```
#Conversió de les dades:
N = 4; #Number of measures
Axis_size = {'X':18, 'Y':28} #Les mides dels eixos son 18, 28
Axis_px = [x for x in range(N)] #llista amb 4 elements, fem 4 mesures

Axis_px[0] = {'X':[40. ,1017.],
              'Y':[1569.,60.]}
Axis_px[1] = {'X':[44. ,1018.],
              'Y':[1568.,57.]}
Axis_px[2] = {'X':[111. ,1084.],
              'Y':[1554.,41.]}
Axis_px[3] = {'X':[46. ,1015.],
              'Y':[1554.,41.]}
```

Llegim les dades preses i fem la transformació esmentada abans. En el procés transformem les dades en valors amb incertesa, amb incertesa de 1 mm (por que si, al lab ho analitzem millor)

In [39]:

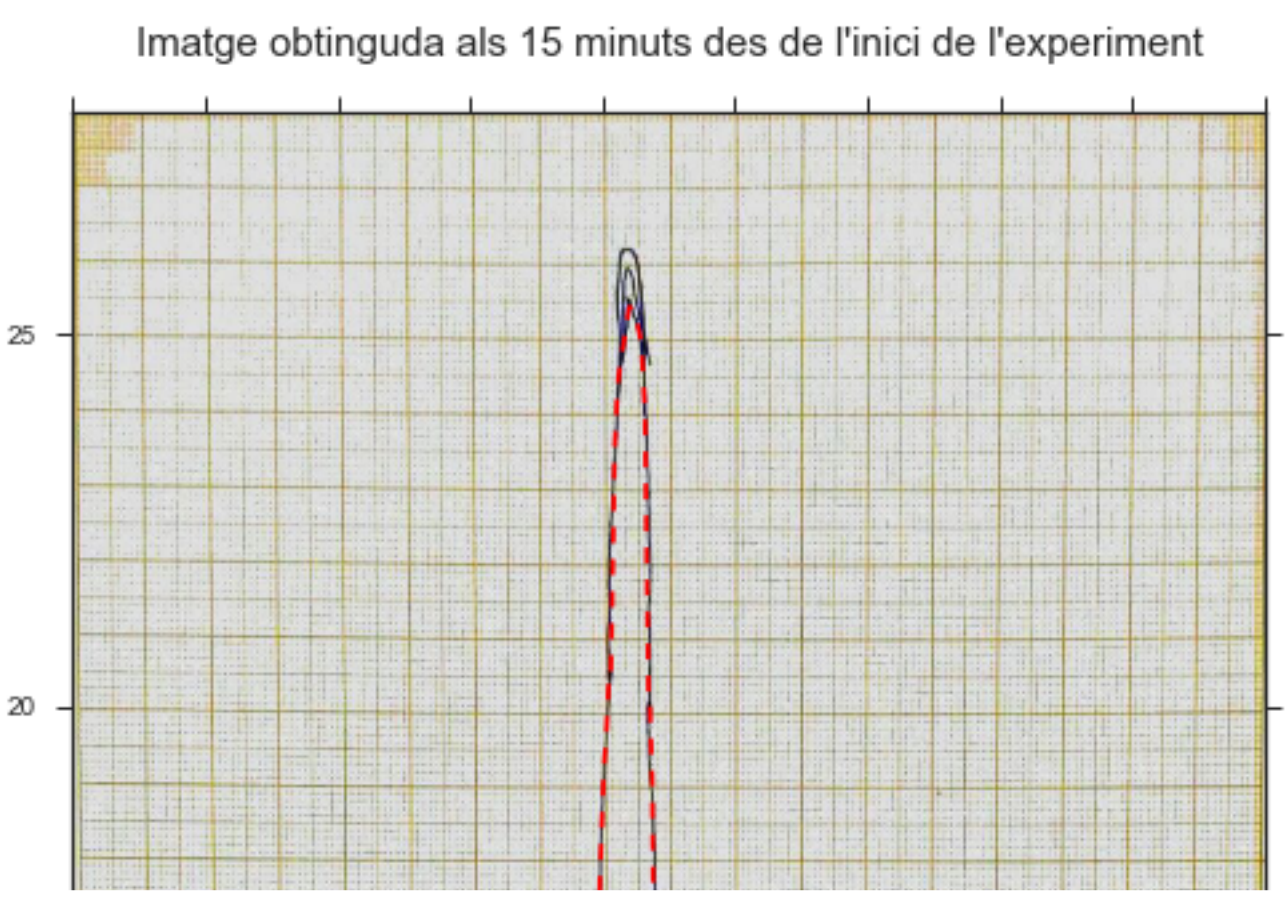
```
data=curve_data

for measure,i in zip(data, range(N)):
    for coord in measure:
        measure[coord] = Axis_size[coord] * (measure[coord]-Axis_px[i][coord][0])

Rdata=data

unc_data = 0.1

data = [
        np.array([
          np.array(data[k]['X']),
           np.array(data[k]['Y'])])
                for k in range(N)]

data= [ unp.uarray(m, 0.1) for m in data]
```

In [40]:

```
data = [x[:,x[0]>3.7] for x in data]
```

In [41]:

```
for k in range(4):

    f=plt.figure(figsize=(8,14))
    plt.xlim([0,18])
    plt.ylim([0,28])



    d=data[k]
    plt.plot(unp.nominal_values(d[0]), unp.nominal_values(d[1]),
                         color='red', ls='--')



    img = Image.open('IN/IMG/raw{:02d}.jpg'.format(k+1))
    img = img.crop(np.array(
                    [Axis_px[k]['X'][0],
                    Axis_px[k]['Y'][1],
                    Axis_px[k]['X'][1],
                    Axis_px[k]['Y'][0]
                   ]).astype(int))
    plt.title("Imatge obtinguda als {:d} minuts des de l'inici de l'experiment\n"
    plt.imshow(img, zorder=0, extent=[0,18,0,28], aspect='auto')
    plt.xlabel('x (cm)')
    plt.ylabel('y (cm)')
    plt.savefig('OUT/FIG/mixed_{:02d}.png'.format(k+1))
    plt.figure()
```



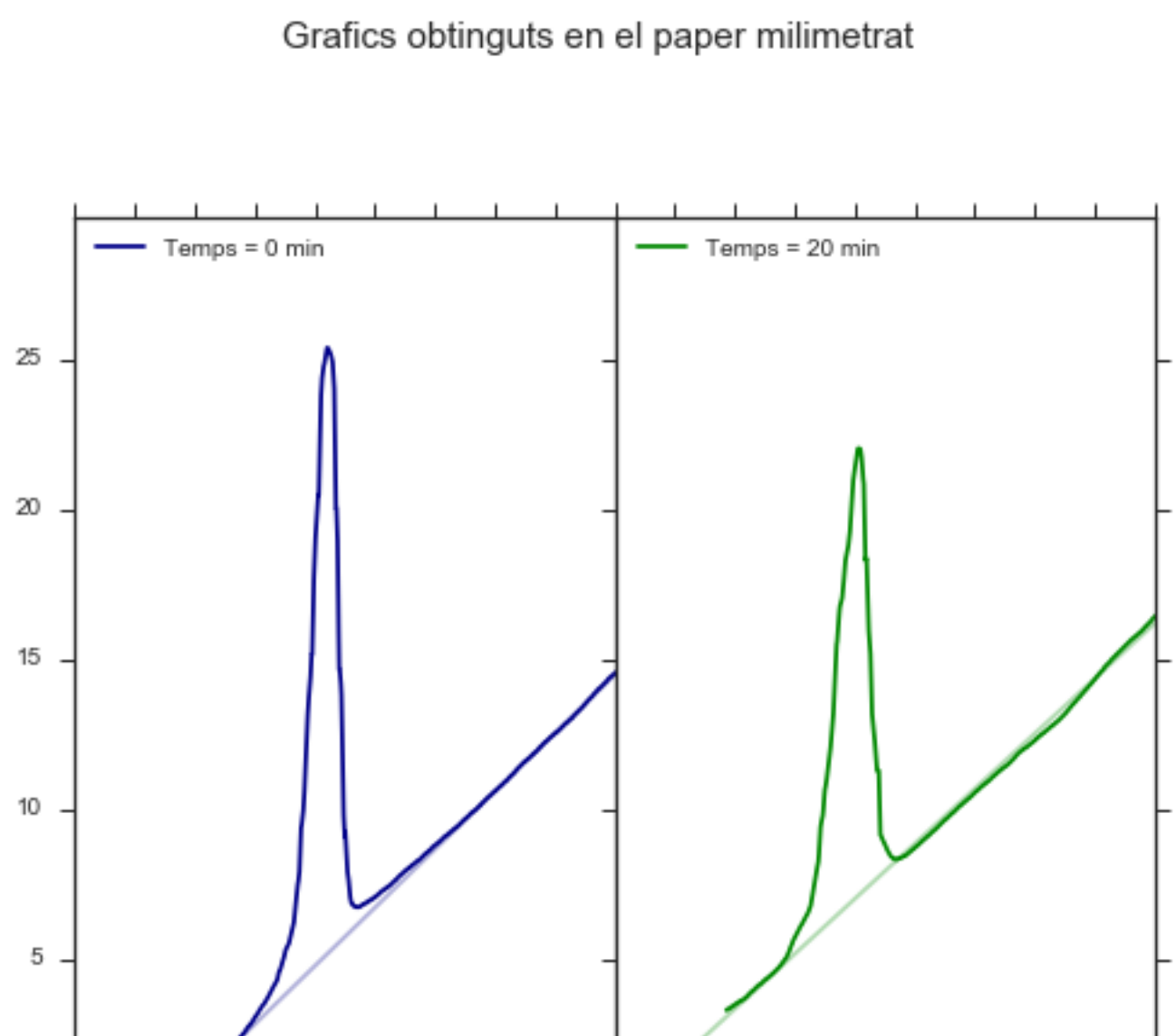
Ara, per a tots els càlculs, ens caldrà calcular la recta base de les gràfiques. Per a fer-ho, assenyalem els punts allunyats del pic i hi fem una recta de regressió:

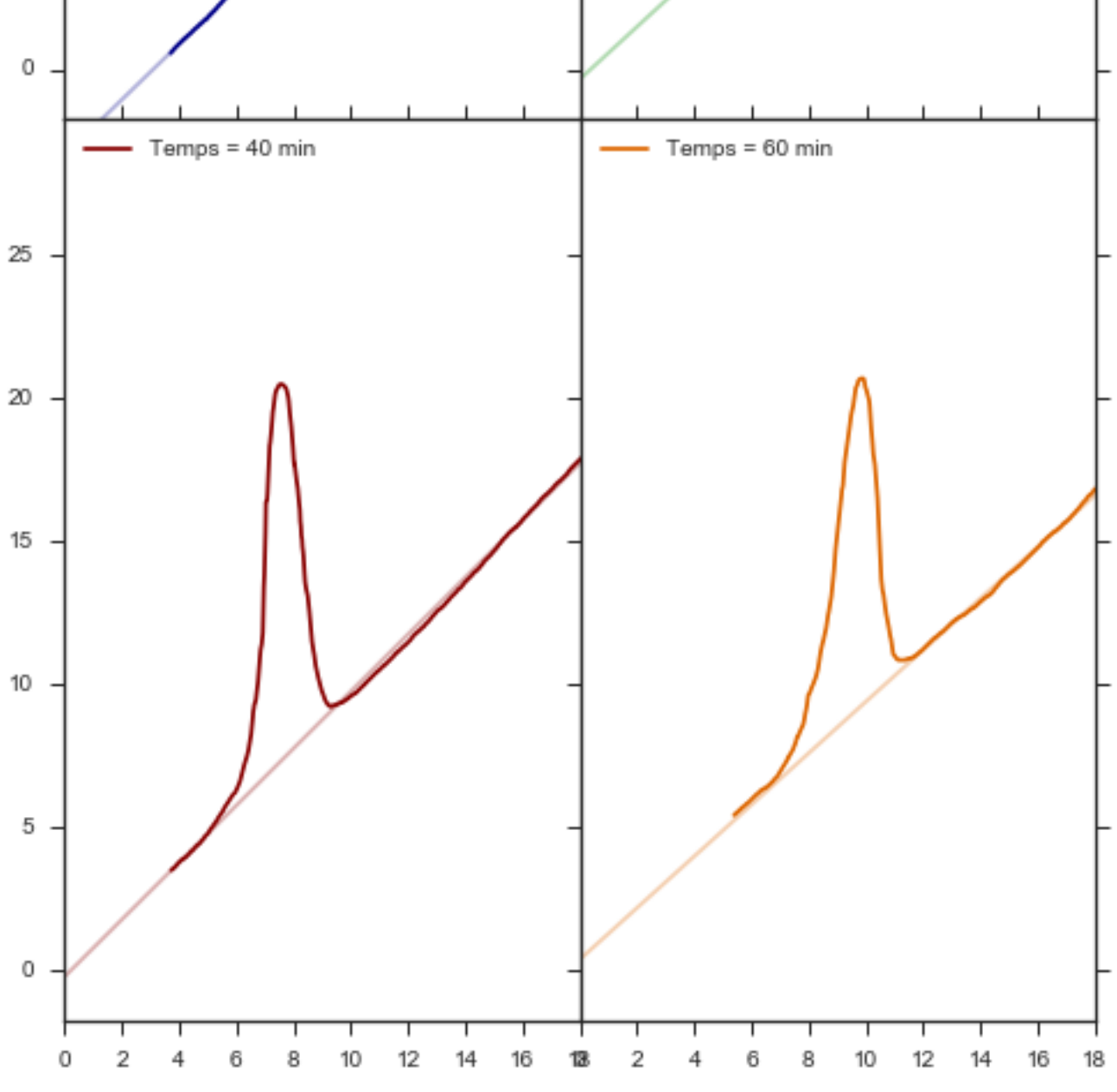
In [42]:

```
reg_selectors=[np.logical_or(c[0]<5.7, c[0]>13) for c in data]

linear=lambda x,a,b:a+b*x

parameters = [scp.optimize.error_curve_fit(linear, c[0][sel], c[1][sel]) for c,se
```

In [43]:

```
f, axis = plt.subplots(2, 2, sharex=True, sharey=True, figsize=(8,14))

axis= [x for y in axis for x in y ]
colors=['#000088', '#008800', '#880000', '#dd6600']

sp = np.linspace(0,18,5)

for x, ax, color in zip(range(N), axis, colors ):
    ax.set_aspect('equal', 'datalim')
    ax.plot(sp, linear(sp, *unp.nominal_values(parameters[x])), color=color, alph
    d=data[x]
    ax.plot(unp.nominal_values(d[0]), unp.nominal_values(d[1]),
                    label='Temps = {} min'.format(20*x),
                    color=color)
    ax.legend(loc=2)
    ax.set_xlim([0,18])
    ax.set_ylim([0,28])

plt.suptitle('Grafics obtinguts en el paper milimetrat', fontsize=15)
plt.savefig('OUT/FIG/grafics_obtinguts.png')
f.subplots_adjust(hspace=0, wspace=0)
```





També podem representar ara els residus de la regressió, la diferència entre la baseline i el mesurat:

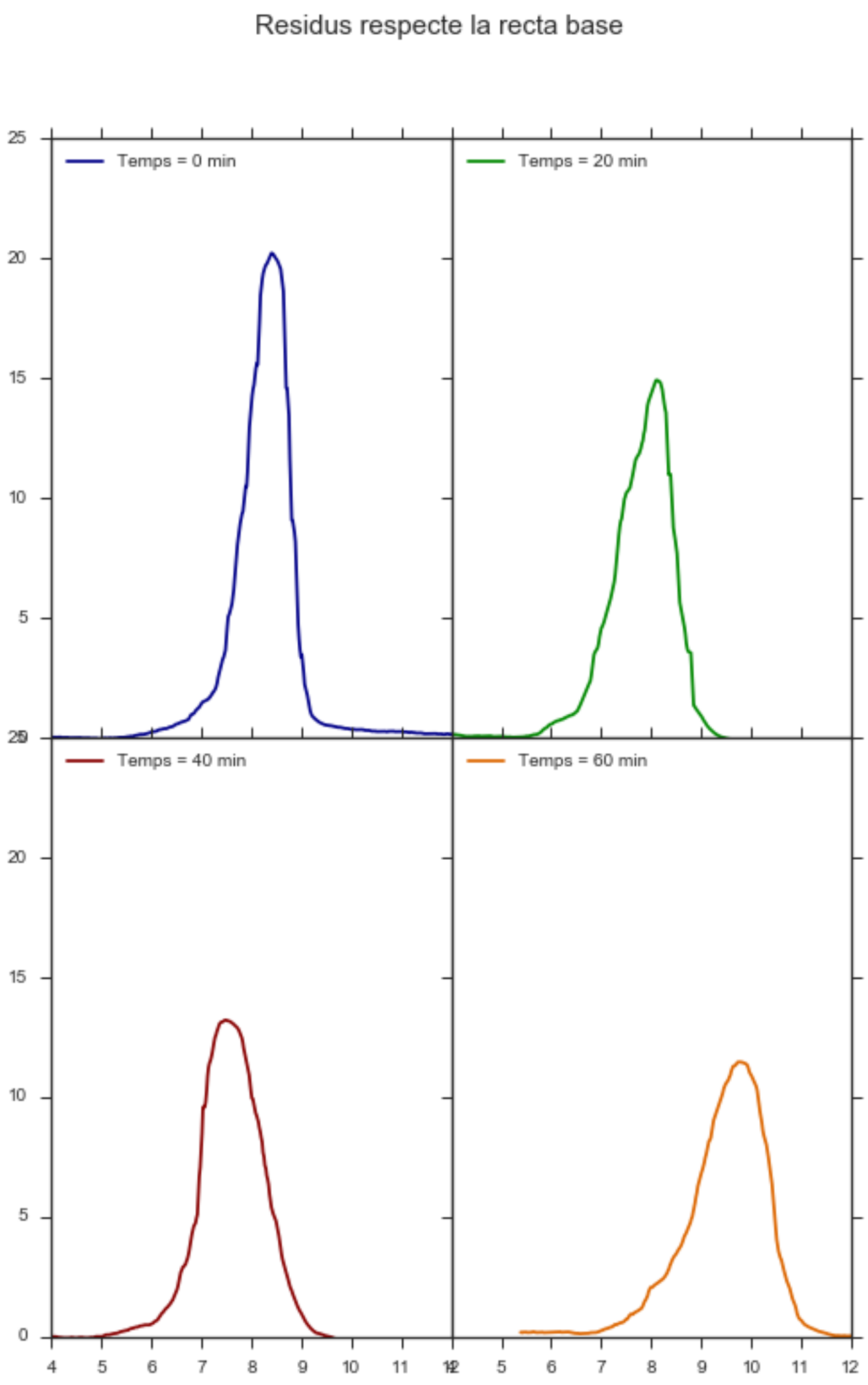
In [44]:

```
residuals = [Y - a-b*X for (X,Y), (a,b) in zip(data, parameters) ]
```

In [45]:

```
f, axis = plt.subplots(2, 2, sharex=True, sharey=True, figsize=(8,12))

axis= [x for y in axis for x in y ]
colors=['#000088', '#008800', '#880000', '#dd6600']

sp = np.linspace(0,18,5)

for x, ax, color in zip(range(N), axis, colors ):

    ax.plot(*unp.nominal_values([data[x][0], residuals[x]]),
                    label='Temps = {} min'.format(20*x),
                    color=color)
    ax.legend(loc=2)
    ax.set_xlim([4,12])
    ax.set_ylim([-0,25])

plt.suptitle('Residus respecte la recta base', fontsize=15)

f.subplots_adjust(hspace=0, wspace=0)
```

```
plt.savefig('OUT/FIG/residus_resp_base.png')
```



Així mateix ens poden interessar les distribucions cumulatives. També podem veure que totes sumen el mateix:

In [46]:

```
cum=[np.cumsum(np.diff(X)*(R[1:]+R[:-1])/2) for (X,Y),R in zip(data, residuals)]
```

In [47]:

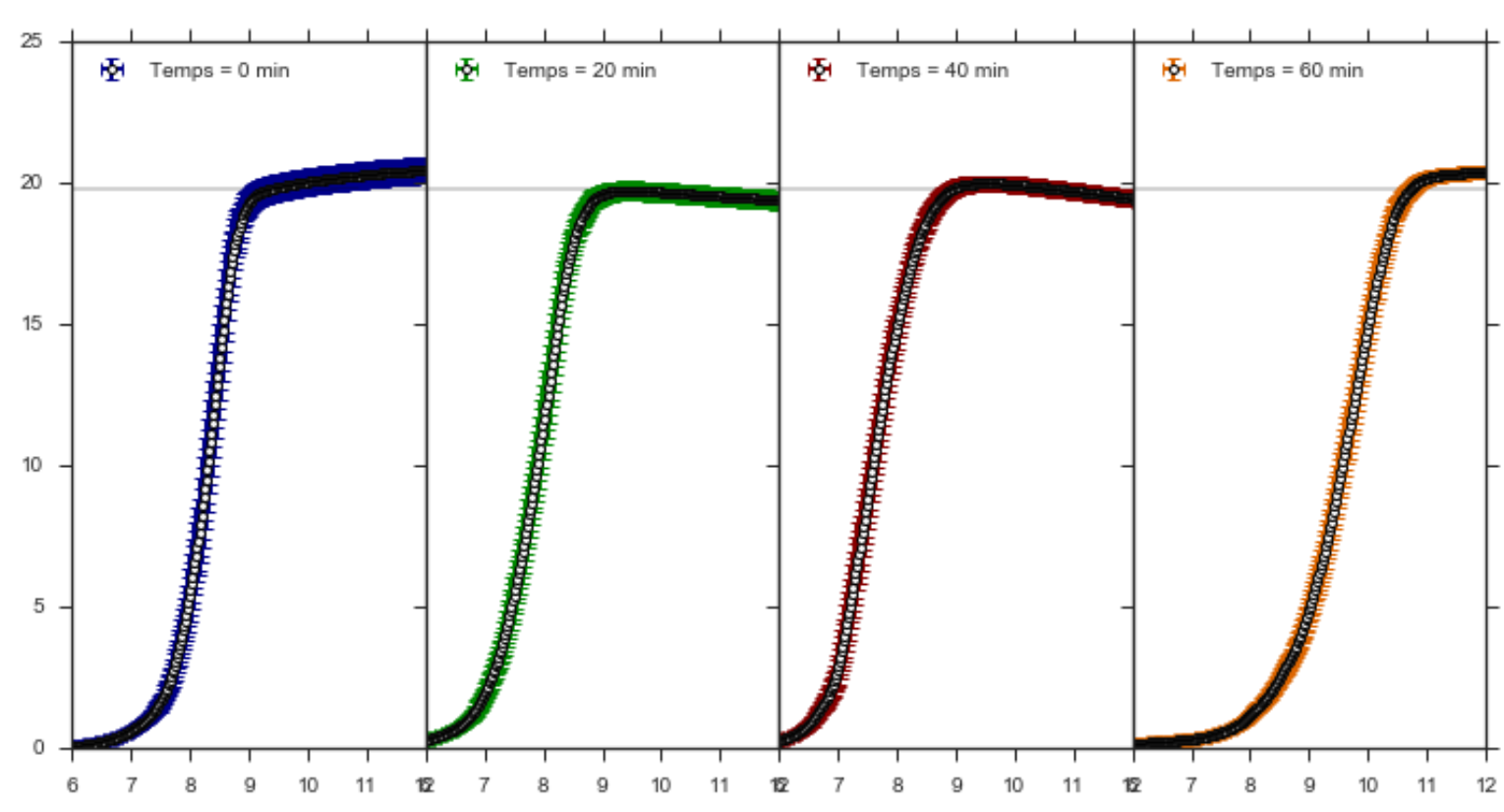
```
cum[0]*=0.9
```

In [48]:

```
areamean=np.mean(unp.nominal_values([k[-1] for k in cum]))
```

In [49]:

```
f, axis = plt.subplots(1, 4, sharex=True, sharey=True, figsize=(12,6))

axis= [x for x in axis ]
colors=['#000088', '#008800', '#880000', '#dd6600']

sp = np.linspace(0,18,5)

for x, ax, color in zip(range(N), axis, colors ):
    ax.plot([0,20], [areamean]*2, color='black', alpha=0.2)
    plt.errorScatter(*[ (data[x][0][1:]+data[x][0][:-1])/2, cum[x]],
                     target=ax,
                     label='Temps = {} min'.format(20*x),
                     color=color)
    ax.legend(loc=2)
    ax.set_xlim([6,12])
    ax.set_ylim([-0,25])

plt.suptitle('Integrals acumulades respecte la recta base', fontsize=15)

f.subplots_adjust(hspace=0, wspace=0)

plt.savefig('OUT/FIG/integrals_acumulades.png')
```



Ara tant sols ens fa falta renormalitzar les dades per a que expressin l'index de refracció en funció de l'alçada:

In [50]:

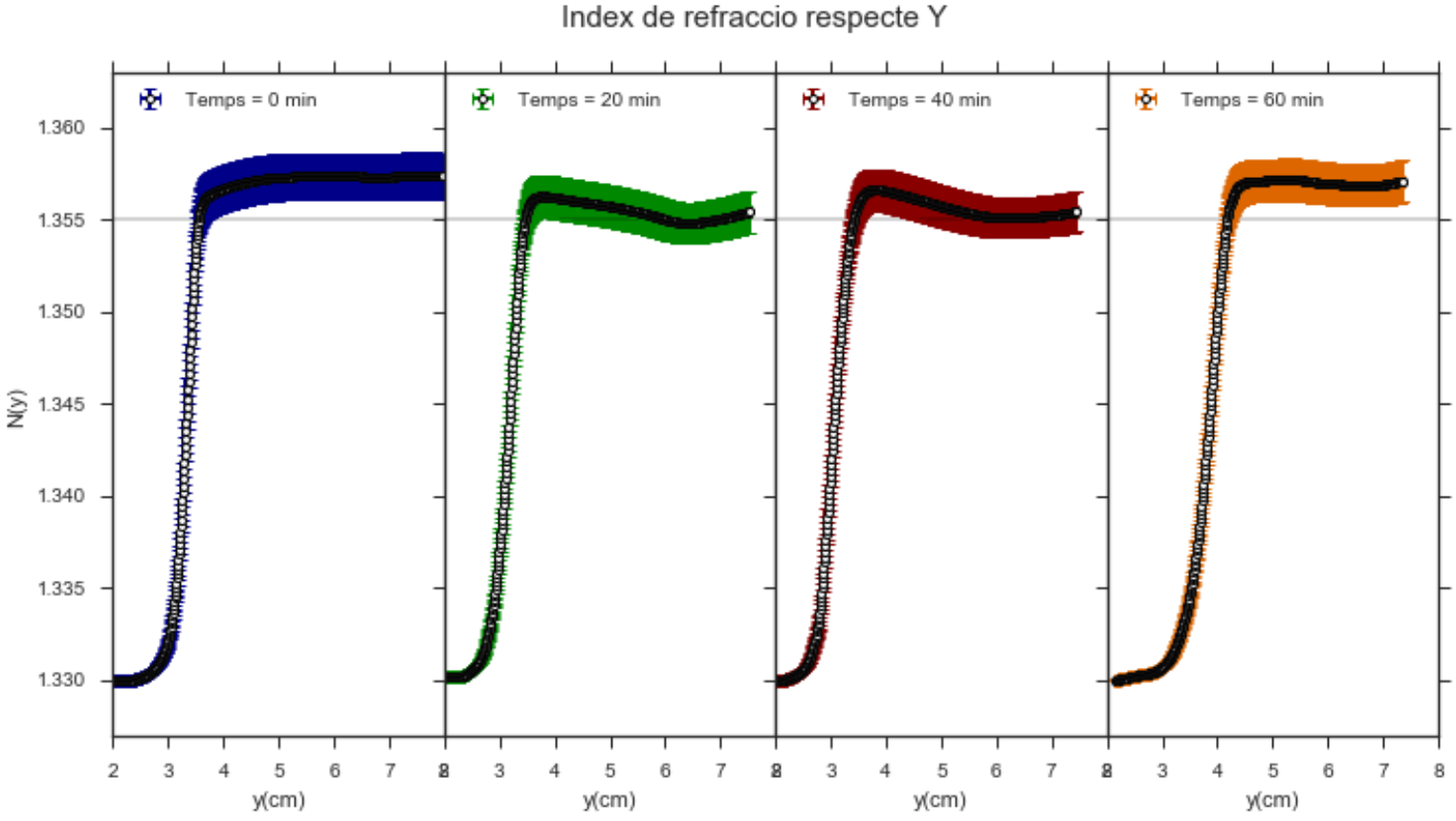
```
Y_cubeta = [B/(A+B)*x for (x,y) in data]
Y_cubeta = [(y[1:]+y[:-1])/2 for y in Y_cubeta]
```

In [51]:

```
n = [B/(A*(A+B)*E)*c+N_H2O for c in cum]
```

In [52]:

```
f, axis = plt.subplots(1, 4, sharex=True, sharey=True, figsize=(12,6))

axis= [x for x in axis ]
colors=['#000088', '#008800', '#880000', '#dd6600']

sp = np.linspace(0,18,5)

axis[0].set_ylabel('N(y)')

for x, ax, color in zip(range(N), axis, colors ):
    ax.plot([0,20], [1.355]*2, color='black', alpha=0.2)
    plt.errorScatter(Y_cubeta[x], n[x],
                     target=ax,
                     label='Temps = {} min'.format(20*x),
                     color=color)
    ax.legend(loc=2)
    ax.set_xlim([2,8])
    ax.set_ylim([1.327,1.363])
    ax.set_xlabel('y(cm)')

plt.suptitle('Index de refraccio respecte Y', fontsize=15)

f.subplots_adjust(hspace=0, wspace=0)
plt.savefig('OUT/FIG/index_de_refraccio.png')
```



In [53]:

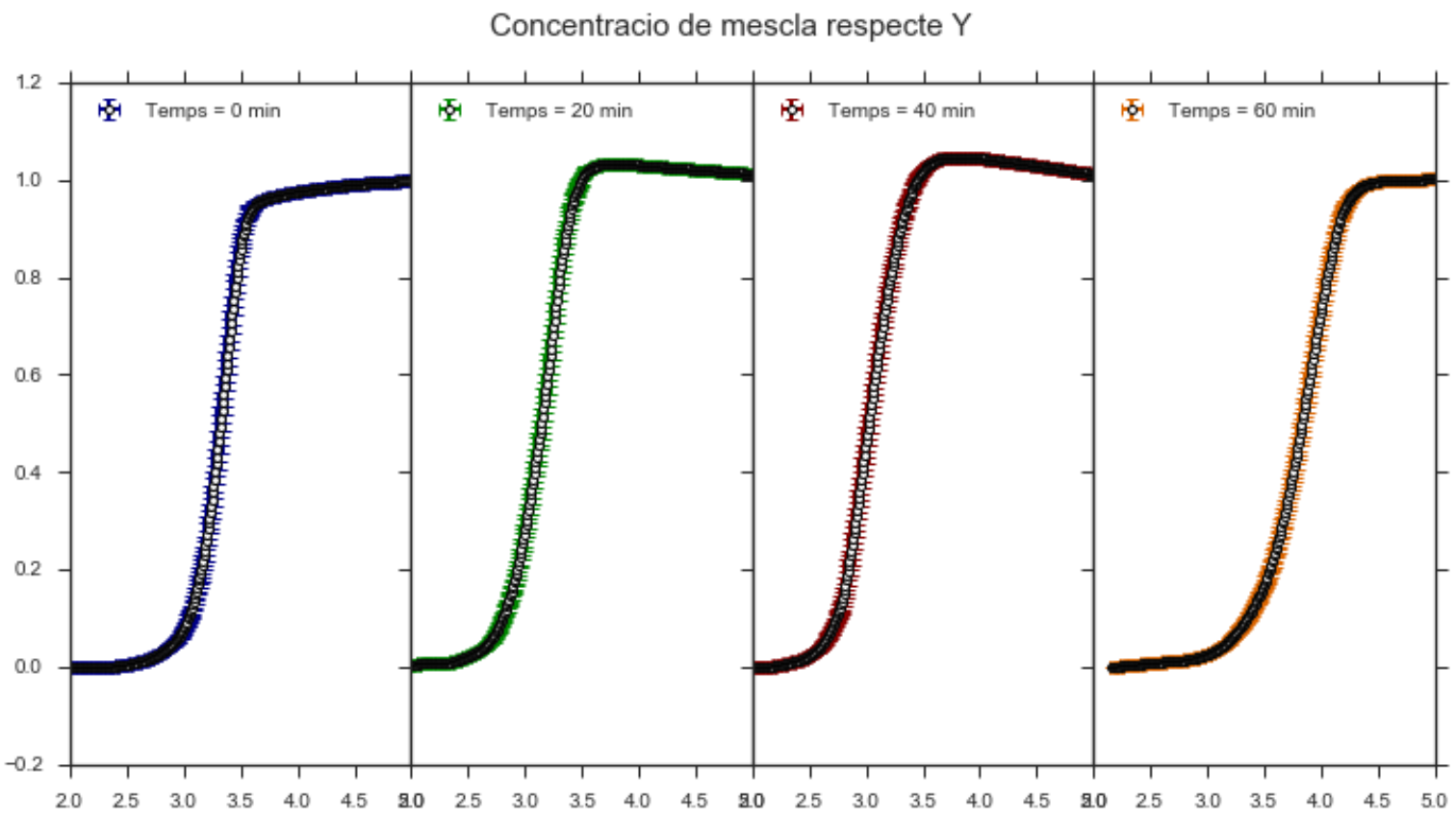
```
concentr=[ (ni-ni[0])/(ni[-1]-ni[0]) for ni in n ]
```

In [54]:

```
gaussian=lambda x,m,s,n : n*1/(2*3.1415*s**2)**.5 *np.exp(-(x-m)**2/(2*s**2))
erf= lambda x,m,s,n: .5*n*(1+scp.special.erf( (x-m)/(2*s**2)) )

p0s= [scp.optimize.error_curve_fit(gaussian,
                                   X, r,
                                   p0=[9,2,15])
      for (X,Y),r in zip(data, residuals)]


#erfp = [scp.optimize.error_curve_fit(erf,
#                                     Y,
#                                     c,
#                                     epsfcn=1E-13,
#                                     maxfev=1000000,
#                                     factor=0.1,
#                                     p0=[(p[0]*B/(A+B)).n, 2*(p[1]*B/(A+B)).n ,1]
#      for Y,c,p in zip(Y_cubeta[0:1], concentr, p0s)]#

erfp = [ [(p[0]*B/(A+B)), 1.8*(p[1]*B/(A+B)) ,1]   for p in  p0s]
```

In [55]:

```
f, axis = plt.subplots(1, 4, sharex=True, sharey=True, figsize=(12,6))

axis= [x for x in axis ]
colors=['#000088', '#008800', '#880000', '#dd6600']

sp = np.linspace(2,8,100)

for x, ax, color in zip(range(N), axis, colors ):
    plt.errorScatter(Y_cubeta[x], concentr[x],
                     target=ax,
                     label='Temps = {} min'.format(20*x),
                     color=color)
    ax.legend(loc=2)
    ax.set_xlim([2,5])

plt.suptitle('Concentracio de mescla respecte Y', fontsize=15)
plt.savefig('OUT/FIG/concentr_mescla.png')
f.subplots_adjust(hspace=0, wspace=0)
```



In [56]:

```
erfp = [ [(p[0]*B/(A+B)), 2*(p[1]*B/(A+B)) ,1]   for p in  p0s]
```

In [ ]:

```

```