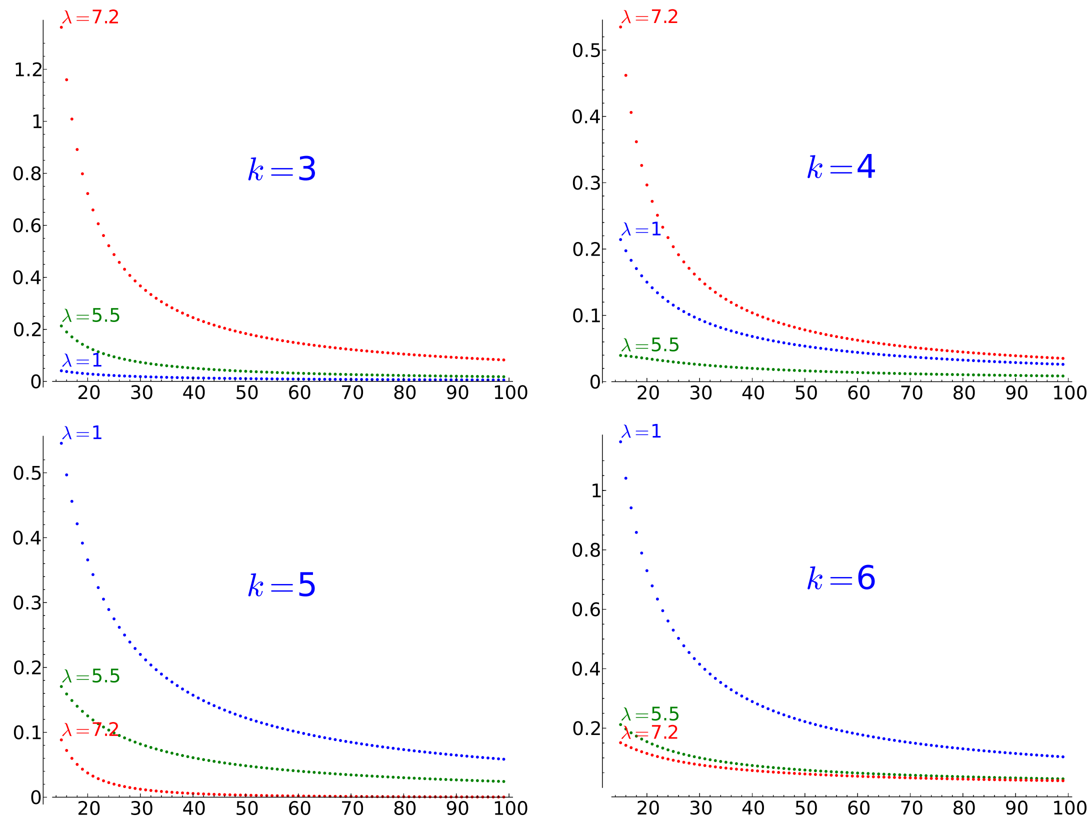
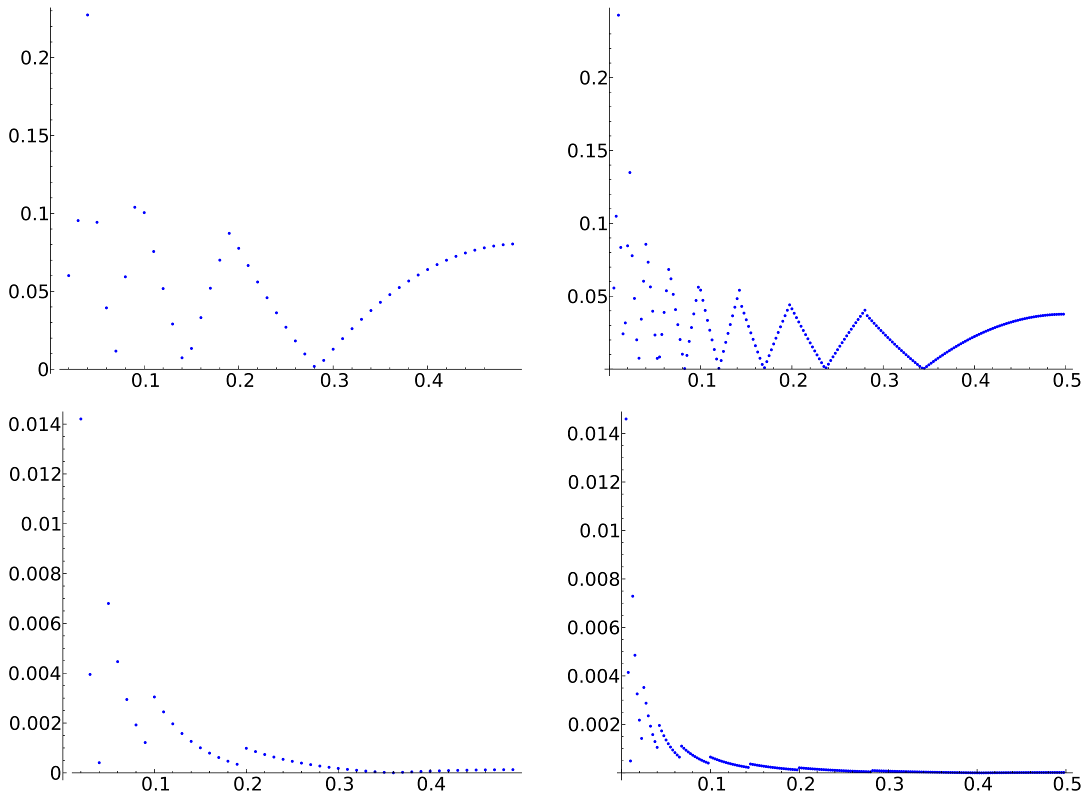
El grado de aproximación de una binomial por una Poisson depende en cierta forma de la relación entre el parámetro y el número de éxitos de los que se calcula la probabilidad. En las siguientes figuras se representa el error relativo al aproximar la probabilidad de k éxitos en una binomial $B(n, p)$ por una Poisson $P(\lambda)$ con $\lambda = np$. En el eje X se indican los valores de n.



Para $X \sim B(n,p)$ se tiene $\mu = np$ y $\sigma = \sqrt{np(1-p)}$. Las dos primeras gráficas muestran el error relativo para $p \in [2/n, 1/2]$ cuando se aproxima el la probabilidad de que el número de éxitos esté en $[\mu-\sigma, \mu+\sigma]$ a través de $F(\mu+\sigma) - F(\mu-\sigma)$ con F la función de distribución de una normal $N(\mu, \sigma)$ (en términos de la estándar, esto es $F_E(1) - F_E(-1) = 0.682689$). Como es lógico, la aproximación, en términos generales no es muy fiable si tomamos p muy pequeño.



Las dos gráficas siguientes muestran una mejora muy notable de las aproximaciones simplemente teniendo en cuenta la “aritmética”: Si calculamos $P(a \leq X \leq b)$ con la binomial, en realidad estamos calculando $P(\lceil a \rceil \leq X \leq \lfloor b \rfloor)$ donde $\lceil a \rceil$ y $\lfloor b \rfloor$ son los números enteros más cercanos por arriba y por abajo a a y b. Al usar $F(b) - F(a)$ con F la función de distribución de una normal $N(\mu, \sigma)$, no tenemos en cuenta esta situación. Es natural pensar que $P(X = k)$ está aproximado por $F(k + 1/2) - F(k - 1/2)$, asociando a k el intervalo de longitud 1 que lo rodea. De esta forma, es de esperar que $F(\lfloor b \rfloor + 1/2) - F(\lceil a \rceil - 1/2)$ dé lugar a una aproximación mejor y eso es lo que representan las dos últimas gráficas.

El código SAGE para las figuras es:

```
def cdfn(mu, sigma, x):
        W = RealDistribution('gaussian',sigma)
        return W.cum_distribution_function(x-mu)

def bin_int(nn,p, a, b):
        re = 0
        for k in srange( ceil(a), floor(b)+1 ):
                re += ( binomial(nn,k)*p^k*(1-p)^(nn-k) ).n()
        return re

def bin_poi(l,k, col='blue'):
        L = []
        for nn in srange(15,100):
                p = l/nn
                re = (exp(-l)*l^k/gamma(k+1)).n()
                re = abs(re/binomial(nn,k)/p^k/(1-p)^(nn-k)-1)
                L.append( (nn,re.n()) )
        P = list_plot(L,plotjoined=False, rgbcolor=col)
        P += text('$\lambda = $'+str(l)[0:3]+'\n',L[0], fontsize=20,
            horizontal_alignment='left', rgbcolor=col)
        return P

def bin_norm(nn, l, arith=0):
        L = []
        for p in srange(2/nn,1/2,1/nn):
                mu = nn*p
                sig = sqrt(nn*p*(1-p))
                a = mu-l*sig
                b = mu+l*sig
                if arith==1:
                        a = ceil(a)
                        b = floor(b)
                re = abs(( cdfn(mu, sig, b+arith/2)-cdfn(mu, sig, a-arith/2)
                    )/bin_int(nn,p, a, b)-1)
                L.append( (p,re) )
        return list_plot(L,plotjoined=False)


def tri_k( k):
        P = bin_poi(1, k, 'blue')
        P += bin_poi(5.5, k, 'green')
        P += bin_poi(7.2, k, 'red')
        d = P.get_axes_range()
        P += text('$k = $'+str(k),( (d['xmin']+d['xmax'])/2,0.4*d['ymin'
            ]+0.6*d['ymax']), fontsize=35)
        return P

P = tri_k( 3 )
P.fontsize(20)
P.save('../images/bin_poi3.eps')
P = tri_k( 4 )
P.fontsize(20)
```

```
P.save('../images/bin_poi4.eps')
P = tri_k( 5 )
P.fontsize(20)
P.save('../images/bin_poi5.eps')
P = tri_k( 6 )
P.fontsize(20)
P.save('../images/bin_poi6.eps')

nn = 100
l = 1
P = bin_norm(nn,l,1)
P.fontsize(20)
P.save('../images/bin_nor100a.eps')
P = bin_norm(nn,l,0)
P.fontsize(20)
P.save('../images/bin_nor100.eps')

nn = 400
P = bin_norm(nn,l,1)
P.fontsize(20)
P.save('../images/bin_nor400a.eps')
P = bin_norm(nn,l,0)
P.fontsize(20)
P.save('../images/bin_nor400.eps')
```