Evaluación 3

integrantes: Fabian Pacheco (20.376.575-4),

Sebastián Saldias (20.307.660-6),

Benjamin Navarrete (20.169.687-9),

Felipe Nazar (20.350.625-2).

Asignatura: Computación numérica

Docente: Juan Lopez Díaz

18 de nov. de 2021

Índice

Desarrollo

Actividad 1

Cuando el valor de una acción en la Bolsa de valores tiene una fuerte alza y sostenida (es persistente), bajo ciertas condiciones del mercado, al cabo de un tiempo típicamente sufre una baja. Este fenómeno suele modelarse mediante una función de la forma:

$$V(t) = V_o + a_1 t + a_2 t^2 + .. + a_n t^n \tag{4}$$

donde $V(t)$ es el valor de la acción en el día t, V_0 es un valor de referencia, y a_n son parámetros desconocidos.

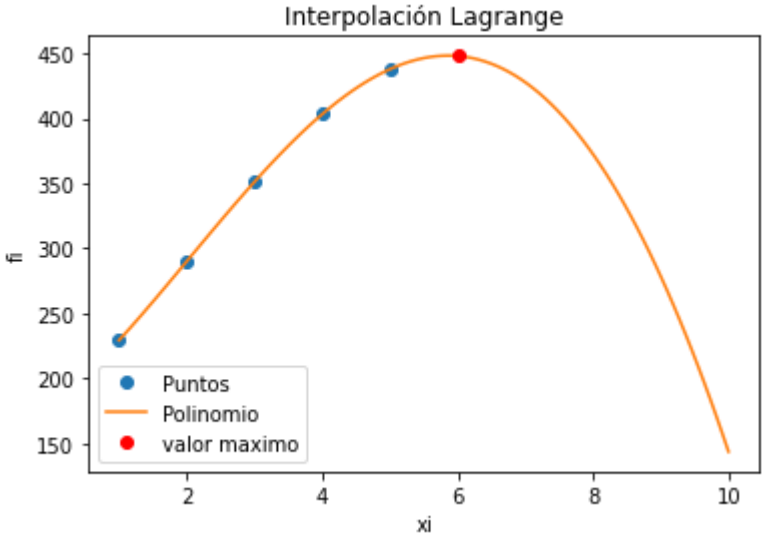
La siguiente tabla da los valores de una acción en alza durante 5 días consecutivos:

Días	1	2	3	4	5
Valores	229.30	290.05	351.20	403.25	437.90

Determine los valores de los parámetros del modelo, el valor máximo V_{max} que alcanzarán esas acciones y el día t_{max} en el que conviene venderlas. Utilice la interpolación mediante polinomios de Lagrange y compare sus resultados al utilizar un polinomio de interpolación de segundo orden (Curvas de Interpolación.ipynb).

Para esta actividad se necesita crear dos métodos para realizar la predicción de la función que se obtiene mediante la interpolación de los puntos, primero con la interpolación de lagrange se obtiene el siguiente resultado:

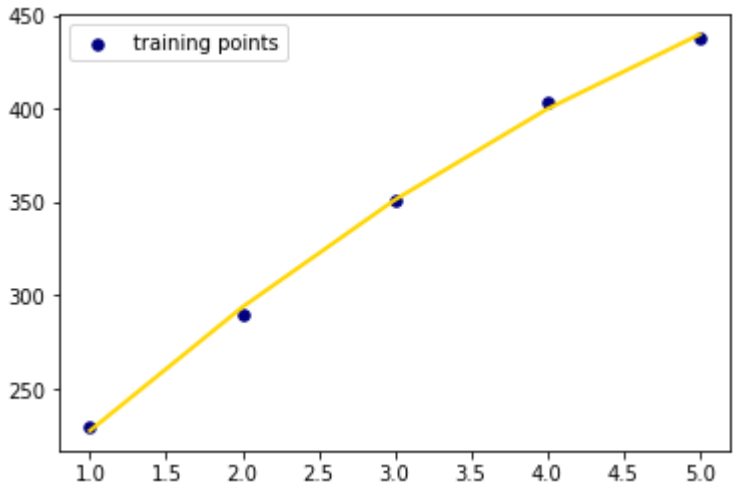
```python
# simplifica el polinomio
polisimple = polinomio.expand()

# para evaluación numérica
px = sym.lambdify(x,polisimple)

# Puntos para la gráfica
muestras = 10
a = np.min(xi)
#! se extiende el espacio muestral a 10 dias
b = 10
#! para graficar
pxi = np.linspace(a,b,101)
pfi = px(pxi)

#! para conocer los valores maximos
pxi_2 = np.linspace(a,b,muestras)
pfi_gr = px(pxi_2)


pfi_2 = list(pfi_gr)

valor_maximo = max(pfi_2)
idx = pfi_2.index(max(pfi_2))

dia_maximo = pxi_2[idx]

#! se obtiene el maximo de los evaluaciones de la funcion
print(f"el maximo valor es: {valor_maximo}
 y se obtiene en el dia {dia_maximo}\n")

# SALIDA
print('Polinomio de Lagrange: ')
print(polisimple)
```

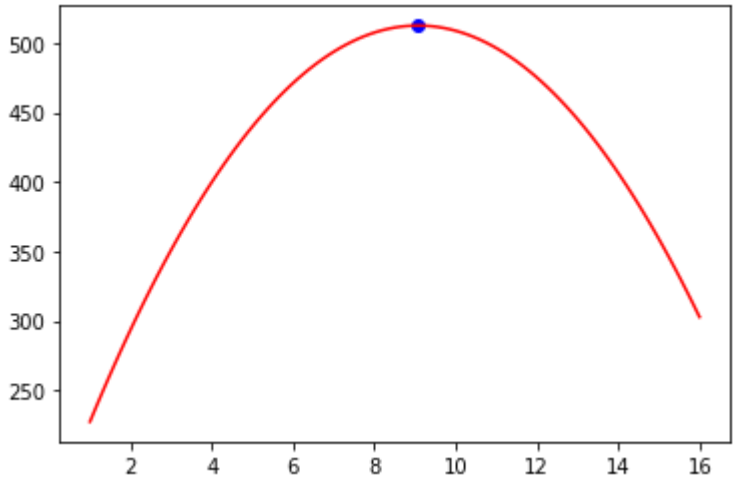


el cual muestra que el mejor día para vender acciones es el día 6, con un total de 448 de valor de la acción.

Luego dados los puntos de entrenamiento se realiza la interpolación usando un polinomio de grado 2, obteniendo los siguientes resultados:

```
x = symbols("x")
fun = -4.37*x**2 + 79.31*x + 152.57
fun_l = sym.lambdify(x,fun)

nums = np.linspace(1,16,101)

nums_eval = list(map(fun_l,nums))

coe_a = -4.37
coe_b = 79.31
maximo_x = -coe_b/(2*coe_a)
maximo_y = max(nums_eval)

print(f"dia maximo:{int(maximo_x)}")
print(f"acciones maximas: {maximo_y}")
```





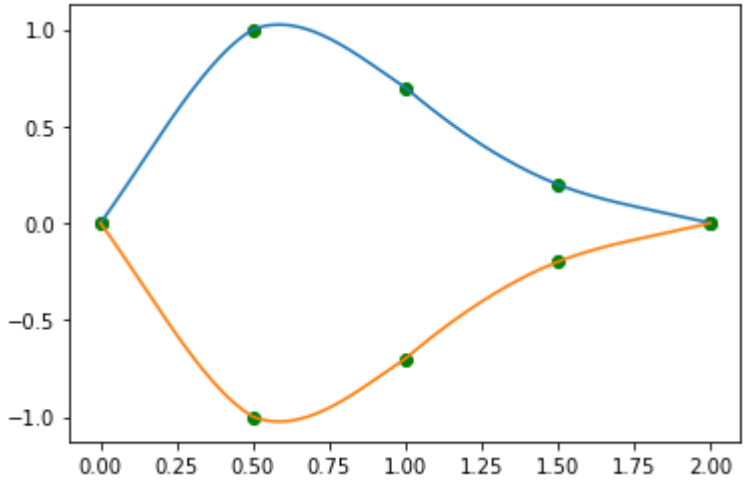
siendo el punto azul el valor máximo de ventas de acciones con una estimación de 512 en el día 9.

Actividad 2

Escriba un programa en Python que dibuje una curva suave que pase por los siguientes puntos:

x	2.0	1.5	1.0	0.5	0.0	0.5	1.0	1.5	2.0
y	0.0	0.2	0.7	1.0	0.0	-1.0	-0.7	-0.2	0.0

```python
def cubic_interp1d(x0, x, y):
    x = np.asfarray(x)
    y = np.asfarray(y)

    # remove non finite values
    # indexes = np.isfinite(x)
    # x = x[indexes]
    # y = y[indexes]

    # check if sorted
    if np.any(np.diff(x) < 0):
        indexes = np.argsort(x)
        x = x[indexes]
        y = y[indexes]

    size = len(x)

    xdiff = np.diff(x)
    ydiff = np.diff(y)

    # allocate buffer matrices
    Li = np.empty(size)
    Li_1 = np.empty(size-1)
    z = np.empty(size)

    # fill diagonals Li and Li-1 and solve [L][y] = [B]
    Li[0] = sqrt(2*xdiff[0])
    Li_1[0] = 0.0
    B0 = 0.0 # natural boundary
    z[0] = B0 / Li[0]

    for i in range(1, size-1, 1):
        Li_1[i] = xdiff[i-1] / Li[i-1]
        Li[i] = sqrt(2*(xdiff[i-1]+xdiff[i]) - Li_1[i-1] *
Li_1[i-1])
        Bi = 6*(ydiff[i]/xdiff[i] - ydiff[i-1]/xdiff[i-1])
        z[i] = (Bi - Li_1[i-1]*z[i-1])/Li[i]

    i = size - 1
    Li_1[i-1] = xdiff[-1] / Li[i-1]
    Li[i] = sqrt(2*xdiff[-1] - Li_1[i-1] * Li_1[i-1])
    Bi = 0.0 # natural boundary
    z[i] = (Bi - Li_1[i-1]*z[i-1])/Li[i]

    # solve [L.T][x] = [y]
    i = size-1
    z[i] = z[i] / Li[i]
    for i in range(size-2, -1, -1):
        z[i] = (z[i] - Li_1[i-1]*z[i+1])/Li[i]

    # find index
    index = x.searchsorted(x0)
    np.clip(index, 1, size-1, index)

    xi1, xi0 = x[index], x[index-1]
    yi1, yi0 = y[index], y[index-1]
    zi1, zi0 = z[index], z[index-1]
    hi1 = xi1 - xi0

    # calculate cubic
    f0 = zi0/(6*hi1)*(xi1-x0)**3 + \
         zi1/(6*hi1)*(x0-xi0)**3 + \
         (yi1/hi1 - zi1*hi1/6)*(x0-xi0) + \
         (yi0/hi1 - zi0*hi1/6)*(xi1-x0)
    return f0

#x = 2, 1.5, 1, 0.5, 0, 0.5, 1, 1.5,2
#y = 0, 0.2, 0.7, 1, 0, -1, -0.7, -0.2, 0

x = np.array([2, 1.5, 1, 0.5, 0])
y = np.array([0, 0.2, 0.7, 1, 0])

xx = np.array([0,0.5, 1, 1.5, 2])
yy = np.array([0,-1, -0.7, -0.2, 0])
plt.scatter(x, y,color="green")
plt.scatter(xx,yy,color="green")

x_new = np.linspace(0, 2, 201)
plt.plot(x_new, cubic_interp1d(x_new, x, y))
plt.plot(x_new, cubic_interp1d(x_new, xx, yy))
```



- ¿Qué tipo de interpolación, polinomial o spline, recomienda utilizar en este caso para evitar el riesgo de oscilaciones?

Para este caso se recomienda la interpolación spline cúbica, debido a la cantidad de datos y además existen dos puntos (x,y) que son iguales.

- Dado que el primer y último punto de la tabla coinciden: ¿Es posible utilizar interpolación mediante spline?. Justifique su respuesta.

No, puesto que al coincidir el primer y último punto de la tabla no es posible realizar la interpolación spline cúbica, ahora, si esta se divide en dos si es posible su implementación.

.

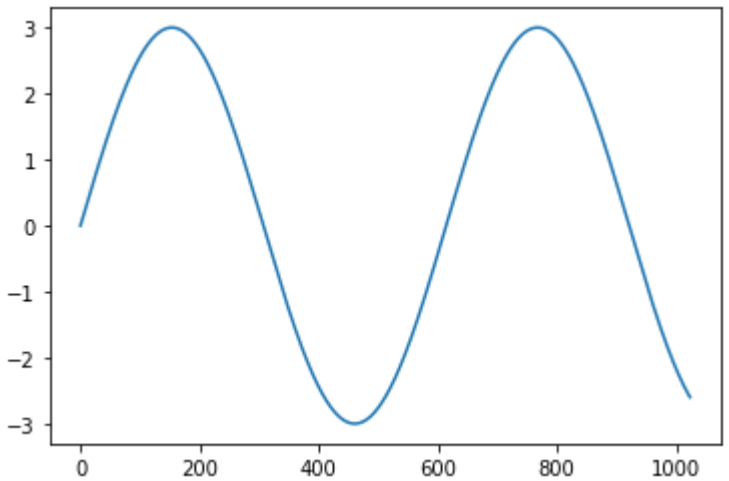
Actividad 3

Se desea utilizar la técnica de interpolación como herramienta de filtrado. Para comprender la utilidad de la interpolación en este proceso, se requiere que realice las siguientes actividades:

1. Genere una función sinusoidal de amplitud $A = 3$, frecuencia $f = \frac{2\pi t}{3}$ $(0 < t < 5\pi)$ y longitud de 2^{10} datos.
2. Genere un ruido gaussiano de 2^{10} datos, con media cero, y desviación típica de 1 (noise = np.random.normal(0, 1, 2^{10}))
3. Sume ambas series de tiempo.
4. Grafique simultáneamente los tres casos anteriores.
5. Considere la serie resultante (sinusoidal + ruido). Mediante ventanas móviles de 32 datos sin traslape, ajuste un polinomio de orden 3 para cada ventana y guarde los datos de interpolación en un nuevo vector. A partir de lo anterior, Ud. obtendrá una nueva serie de tiempo la cual deberá utilizar para **filtrar** la serie resultante **sinusoidal + ruido** mediante una simple resta.
6. Grafique la serie (sinusoidal + ruido) y la obtenida por interpolación y compare los valores promedios y desviaciones típicas para cada una de las series utilizadas en el estudio.

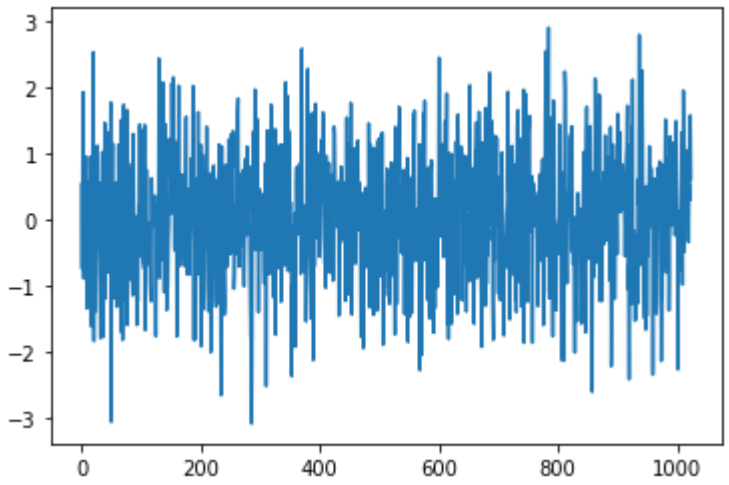
1.

```
import numpy as np
from pylab import *
import matplotlib.pyplot as plt
from sympy.solvers import solve
from sympy import*
x = np.linspace(0, 5, 2**10,endpoint=True)
print("x =",x)
seno = 3*np.sin((2*np.pi*x)/3)
plt.plot(seno)
show()
```



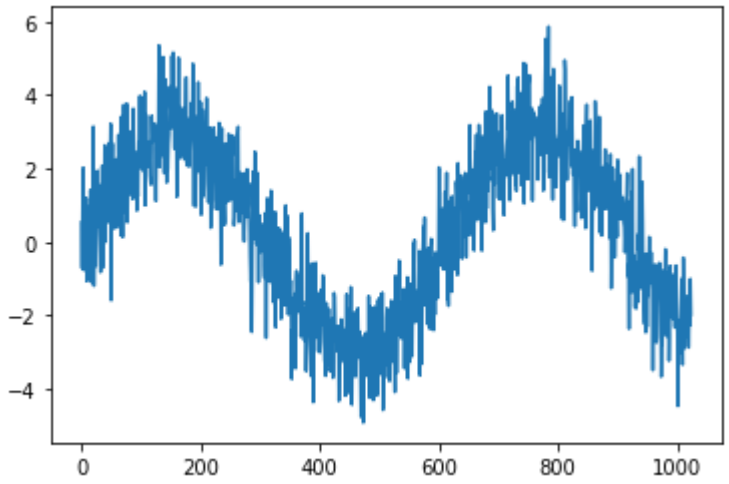
2.

```
noise = np.random.normal(0,1,2**10)
plt.plot(noise)
show()
```



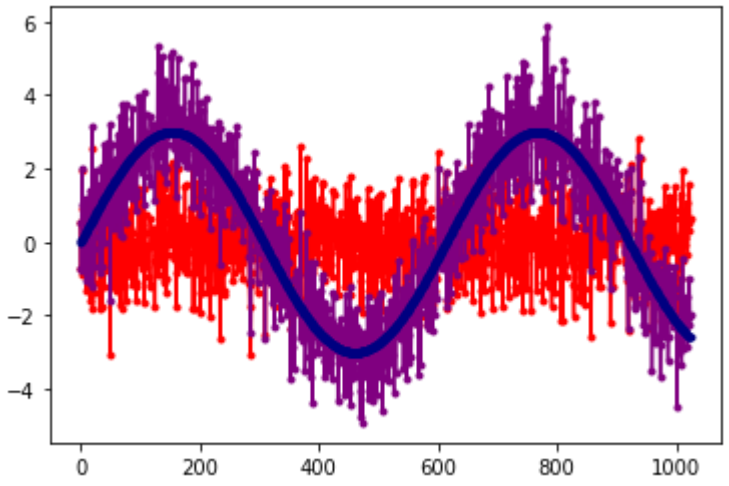
3.

```
suma = seno + noise
plt.plot(suma)
show()
```



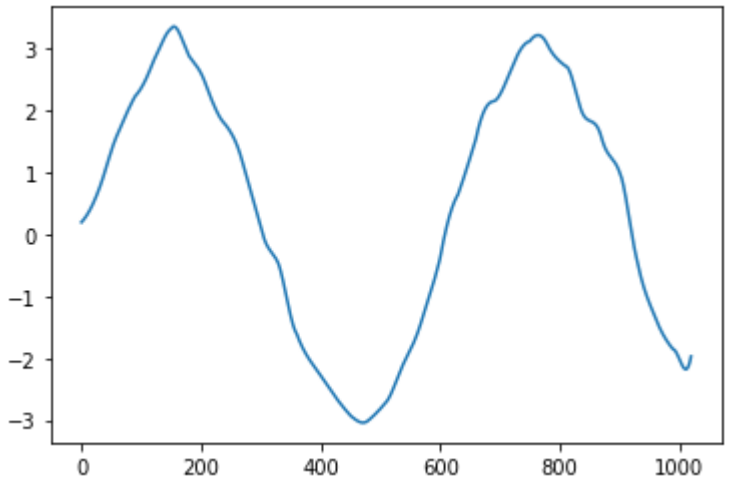
4.

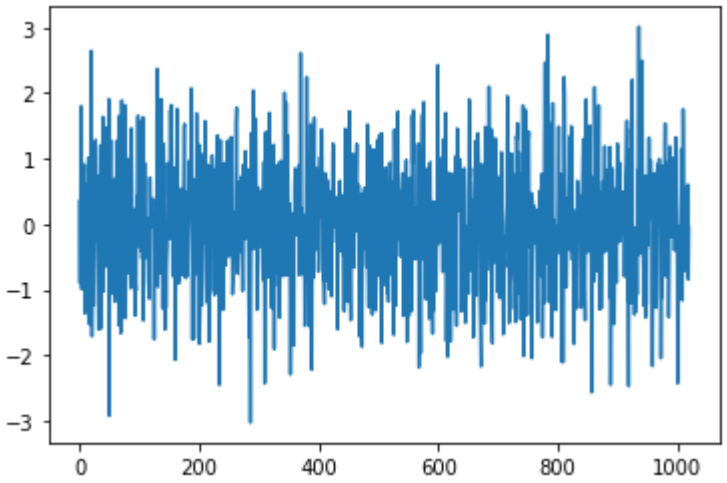
```
plt.plot(noise,color='red', marker='.', label='training points')
plt.plot(suma,color='purple', marker='.', label='training points')
plt.plot(seno,color='navy', marker='.', label='training points')
show()
```



5.

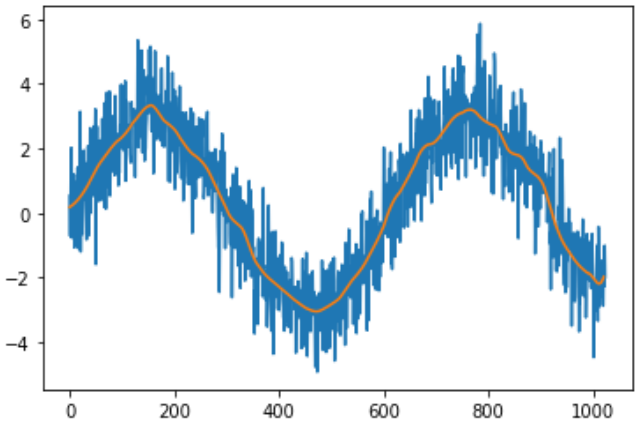
```
import numpy.polynomial as P
n = 30
K = 2
x = suma
Nx = len(x)
yc = []
Nw = 2*n+1 # window size
No = n+1 # overlap

# compute weights
wa = []
wb = []
for i in range(0,No,1): # [0,...,64]
    wa.append(1-((i+1)-1)/n)
    wb.append(((i+1)-1)/n)

# compute the trend data
idNw = np.array(range(1,Nw+1,1))
#print("longitud idNw =",len(idNw))
idNwx = np.array(range(0,Nw,1))
#print("longitud idNwx =",len(idNwx))
p,coef = P.polynomial.polyfit(idNw, x[idNwx],K,full=True)
ya = P.polynomial.polyval(idNw,p)
#print("longitud ya =",len(ya))
yc[0:n] = ya[0:n]
for i in range(Nw-1,Nx,n):
    #print("[i] =",i)
    ni = n+i
    #print("ni =",ni)
    m = np.amin(np.array([ni,Nx-1]))
    #print("m =",m)
    idx = np.array(range(i-No+1,m+1,1))
    #print("idx =",idx)
    p,coef = P.polynomial.polyfit(idx,x[idx],K,full=True)
    yb = P.polynomial.polyval(idx,p)
    yc[i-No+1:i+1] = wa*ya[n:Nw]+wb*yb[0:n+1]
    #print("longitud yc =",len(yc))
    ya = yb
idyc = np.array(range(i,m+1,1))
p,coef = P.polynomial.polyfit(idyc, x[idyc],K,full=True)
yx = P.polynomial.polyval(idyc,p)
yc[i+1:m+1]=yx[i:m+1]
```





6.

```
Detrend = np.array(suma[:-3]) - np.array(yc)
plt.plot(Detrend)
```



El promedio de la gráfica “suma”(sinusoidal + ruido) es de **0.4517288181153033** y además el promedio de la gráfica obtenida por la interpolación es de **0.4509996566988013,** siendo estos valores alusivos a una exactitud obtenida de la limpieza de la función sinusoidal+ruido que se limpió.

Las desviaciones típicas de cada gráfica corresponden respectivamente a **2.241387365386033** y **2.023632350757495** los cuales tienen una diferencia de 0.2 décimas aproximadamente

Conclusión

Se puede ver que los gráficos de la actividad 1 obtienen estimaciones distintas debido al grado de exactitud de cada método y como cada uno trabaja con los datos dados.

Con respecto a la actividad 2, no siempre un método es efectivo para todos los casos, se pudo apreciar que existen inconvenientes para realizar una curva cuando los puntos se repiten en ambos ejes, por lo que se requiere usar métodos que sean más óptimos para este tipo de casos.

La actividad 3 nos presentó un caso de limpieza de datos, lo cual sirve para obtener y crear modelos que se ajusten a las funciones obtenidas cuando a estos datos se les quita el ruido con el que provienen.

En general dependiendo del caso es más factible implementar un método que otro, como por ejemplo, dependiendo de la cantidad de datos ya no resulta factible implementar interpolación si hay más de 6 datos.