Proyecto 2, Modelos de Gestión Financiera

Sebastian Puerto

25 de octubre de 2019

Punto 2

In [1]:
```
from simulador_S import grafico_valor_activo

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
```

Queremos ver cómo el uso de una opción call ventana puede ofrecer seguridad en un portafolio, analizar lo que sucede cuando se hacen portafolios conteniendo combinaciones de un activo S, una opción call ventana asociada a este activo C y una inversión libre de riesgo.

Primero creamos la función que hace el cálculo (detérministico) de una opción Call Ventana.

Función de Cálculo del valor de la opcíon Call Ventana

In [31]:
```
def d1(S, t, E, r, sig, T):
    res = np.log(E/S) - (r + sig**2/2)*(T-t)
    res = res/( sig * np.sqrt(T-t) )

    return res

def d2(S, t, E, r, sig, T):
    res = np.log(E/S) - (r - sig**2/2)*(T-t)
    res = res/( sig * np.sqrt(T-t) )

    return res

def callV(S, t, E1, E2, r, sig, T):

    d1E1 = d1(S, T, E1, r, sig, T)
    d1E2 = d1(S, T, E2, r, sig, T)
    d2E1 = d2(S, T, E1, r, sig, T)
    d2E2 = d2(S, T, E2, r, sig, T)

    p1 = norm.cdf(d1E2) - norm.cdf(d1E1)

    p2 = norm.cdf(d2E2)- norm.cdf(d2E1)
    return S * p1  -  E1*np.exp(-r*(T-t)) * p2

def deri_C_S(Ss, Cs):
    dSs = Ss[1:] - Ss[:-1]
    dCs = Cs[1:] - Cs[:-1]
    return dCs/dSs
```

Uso de la Teoría de la Cartera para encontrar portafolios Óptimos

Usaremos el portafolio óptimo, en el sentido de tener mínimo riesgo medido en desviación estándar, que sea posible de la combinación de un activo S y una opción call ventana C.

Funciones para Cálculo de Portafolio Óptimo

Funciones Auxiliares

In [3]:
```
def portafolio_de(x, y, z):
    return  np.asarray([[x], [y], [z]])

def varPort(x, S):
        return x.T.dot(S).dot(x)[0,0]
```

Función de Cálculo de Portafolios Optimos

Dada la lista de precios historicos de M activos para N+1 periodos en términos de una matriz de MxN, la siguiente función calcula el portafolio óptimo (aquel de menor varianza).

In [4]:
```
# Dada la lista de precios historicos de M activos para N+1 periodos en términos de una matriz de MxN,
# calcula el portafolio óptimo (menor varianza) (matriz Mx1 con porcentaje de inversión en cada activo)

def calcularPortOpt(precios, tau):
    M, N = np.shape(precios)
    N -= 1

    # Cálculo de matriz de retornos (MxN)
    retornos = np.zeros((M, N))
    for k in range(M):
        for i in range(N):
            retornos[k, i] = np.log(precios[k, i+1]/precios[k, i])#(precios[k, i+1]-precios[k, i])/precios[k, i] #np.log(precios[k, i+1]/precios[k,
 i])

    # Cálculo de vector de promedio temporal de retornos de cada activo (Mx1)
    retProm = np.mean(retornos, 1, keepdims = True)

    # Cálculo de matriz de covarianzas
    S = np.zeros((M,M)) # Inicializacion en 0's

    for k in range(M): # Iterar con k sobre activos
        for l in range(M): # Iterar con l sobre activos
            for i in range(N): # Iterar sobre el tiempo con i
                # Para la combinacion de activos k y l se suma la contribución a la covarianza por el tiempo i
                S[k, l] += (retornos[k, i] - retProm[k])*(retornos[l, i] - retProm[l])

    S = S/N
    #print("Matriz de covarianzas:\n", S)

    # Cálculo de vector de varianzas y desviaciones estándar (matrices Mx1)
    varianzas = np.array([S[i, i] for i in range(M)]).reshape((M, 1))
    desvs = np.sqrt(varianzas)
    #print("\nVarianzas:\n", varianzas)

    # Cálculo de Parametros de la teoria
    Sinv = np.linalg.inv(S)
    u = np.ones((M, 1))

    A = u.T.dot(Sinv.dot(u))[0,0]
    B = u.T.dot(Sinv.dot(retProm))[0,0]
    C = retProm.T.dot(Sinv.dot(retProm))[0,0]
    D = A*C - B**2

    # Cálculo de rendimientro promedio mu consistente con la taza libre de riesgo tau
    def muTau(tau):
        return (C - tau*B)/(B - tau*A)

    muopt = muTau(tau)#B/A

    # Cálculo del portafolio optimo dado parametro mu. Devuelve vector Mx1
    def xOptMu(mu):
        return ((C - B*mu)/D) * Sinv.dot(u) + ((A*mu - B)/D) * Sinv.dot(retProm)

    xopt = xOptMu(muopt)

    # Cálculo de varianza de un portafolio x: x^T S x

    varopt = varPort(xopt, S) #1/A

    return xopt, varopt, S
```

Ejemplo de Portafolios y Análisis de sus riesgos

A continuación utilizamos constantes los parámetros $\mu = 0.02, \sigma = 0.095$ y $r = 0.025$, donde μ es el rendimiento logarítmico promedio del activo subyacente, σ su volatilidad y r o τ es la tasa libre de riesgo; notamos que estos valores pueden corresponder a valores esperados para escalas temporales de meses.

Asumimos constante un tiempo de madura de 24 unidades de tiempo, un precio strike mínimo de $E_1 = 70$ y un precio de venta máximo de $E_2 = 200$, para una opción con valor inicial de $S_0 = 100$. Estos valores no son de gran relevancia, son escogidos de manera arbitraria para poder hacer cálculos explícitos.

Para analizar la utilidad de una opción call, vemos distintos portafolios que pueden ser creados con S, C y L, donde estamos llamando L una inversión libre de riesgo, cuya fórmula estará dada por $L = L_0 e^{rt}$.

Los portafolios escogidos son:

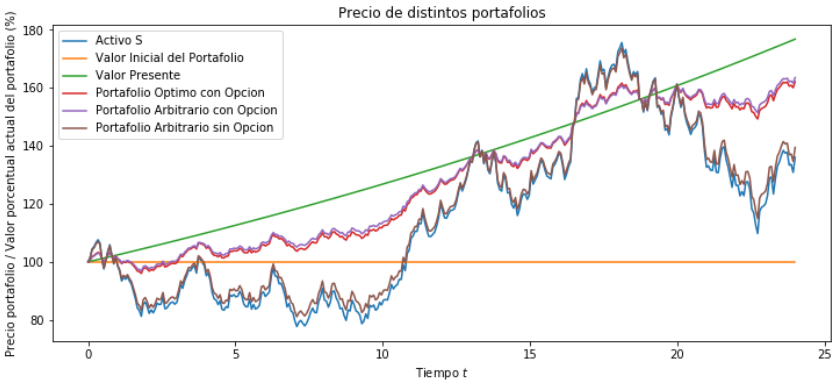
- Solo el activo S
- Solo la opción C
- Solo la inverisón libre de riesgo L
- El portafolio óptimo dado por la teoría de la cartera de combinaciones de S y C
- Un portafolio arbitrario con posición en corto para la opción call que combine S, C y L, con 165% del capital en S, 15% del capital en L, y -85% del capital en la opción C.
- El mismo portafolio arbitrario con posición en corto para la opción call, suprimiendo la porción correspondiente a la opción (e incrementando proporcinalmente las demás inversiones): S, y L
- El mismo portafolio arbitrario con posición en corto para la opción call, suprimiendo la porción correspondiente a la inversión libre de riesgo (e incrementando proporcinalmente las demás inversiones): S, C

Para estos portafolios hallamos el riesgo involucrado, medido en términos de la desviación estándar σ de sus rentabilidades logarítmicas.

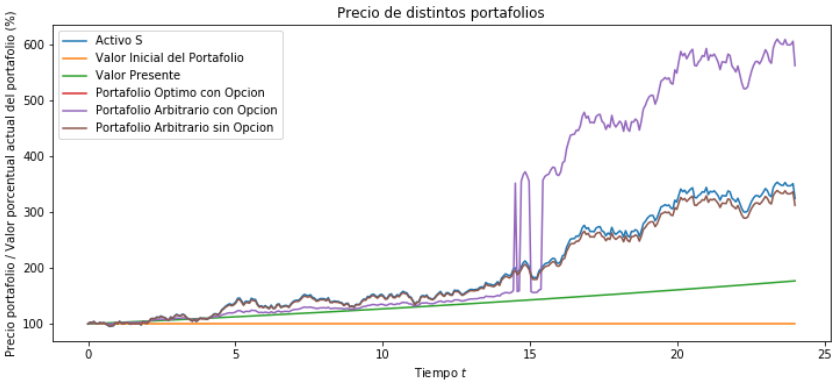
In [134]:

```
mmu = .02
ssig = 0.095
tau = ssig/4.

for i in range(3):
    ## Simulacion i
    print("\n\n\t\tSimulacion numero", i+1, "con los mismos parametros mu, sigma, tau =", mmu, ssig, tau,"\n")
    ts, Ss = grafico_valor_activo(mu = mmu, sig = ssig, S0 = 100, N = 24, txtad = "Sim. 1", graficar = False, pts = 360)
    #ts, S2s = grafico_valor_activo(mu = mmu, sig = ssig, S0 = 100, N = 60, txtad = "Sim. 2", graficar = False)

    Cs = callV(S = Ss, t = ts, E1 = 70, E2 = 200, r = tau, sig = ssig, T = 24) # r = mu / 1.2
    Ls = 100.*np.exp(tau*ts)
    print("Derivada de C respecto a S promedio:", np.mean(deri_C_S(Ss, Cs)))
    print("Desviacion estandar de derivada de C respecto a S promedio:", np.std(deri_C_S(Ss, Cs)))
    print()
    print("Sigma calculada para el portafolio S:", np.sqrt(S[0,0]))
    print("Sigma calculada para el portafolio C:", np.sqrt(S[1,1]))
    print("Sigma calculada para el portafolio L:", np.std(np.log(Ls[1:]/Ls[0:-1])))

    # Portafolio 1:
    precios = np.array([Ss, Cs])
    portMej3, sig3, S = calcularPortOpt(precios, tau)
    print("Portafolio optimo (S, C) compatible con tasa:\n", portMej3)
    musMej3 = (portMej3[0,0]*100/Ss[0])*Ss + (portMej3[1,0]*100/Cs[0])*Cs #+ portMej3[2,0]*Ls
    print("Sigma calculada para el portafolio optimo de S, opcion:", sig3)

    # Otros portafolios

    lamOp = -0.8
    lamS = 1.65
    lamL = 0.15

    coefOp = lamOp*100/Cs[0]
    coefS  = lamS *100/Ss[0]
    coefL  = lamL *100/Ls[0]


    #portafolioCon = portafolio_de(lamS, lamOp, lamL)
    #portafolioSin = portafolio_de(lamS/(lamS + lamL), 0, lamL/(lamS + lamL))
    #portafolioNoL = portafolio_de(lamS/(lamS + lamOp), lamOp/(lamS + lamOp), 0)

    preciosPortCon = (100/Ss[0])*lamS * Ss                + (100/Cs[0])*lamOp * Cs                + (100/Ls[0])*lamL * Ls
    preciosPortSin = (100/Ss[0])*lamS/(lamS + lamL) * Ss  + 0 * Cs                                + (100/Ls[0])*lamL/(lamS + lamL) * Ls
    preciosPortNoL = (100/Ss[0])*lamS/(lamS + lamOp) * Ss + (100/Cs[0])*lamOp/(lamS + lamOp) * Cs + 0 * Ls

    retornosCon = np.log(preciosPortCon[1:]/preciosPortCon[:-1])
    retornosSin = np.log(preciosPortSin[1:]/preciosPortSin[:-1])
    retornosNoL = np.log(preciosPortNoL[1:]/preciosPortNoL[:-1])

    riesgoSTDCon = np.std(retornosCon)
    riesgoSTDSin = np.std(retornosSin)
    riesgoSTDNoL = np.std(retornosNoL)

    print("Sigma calculada para el portafolio sin opcion:", riesgoSTDSin)
    print("Sigma calculada para el portafolio con opcion:", riesgoSTDCon)
    print("Sigma calculada para el portafolio con opcion, sin tasa:", riesgoSTDNoL)

    ####### Graficas
    fig = plt.figure(figsize=(12, 5))
    plt.title("Precio de distintos portafolios")
    plt.xlabel("Tiempo $t$")
    plt.ylabel("Precio portafolio / Valor porcentual actual del portafolio (%)")

    plt.plot(ts, Ss)
    plt.plot(ts, 100*ts/ts)
    plt.plot(ts, Ls)
    plt.plot(ts, musMej3)
    plt.plot(ts, preciosPortCon)
    plt.plot(ts, preciosPortSin)
    #plt.plot(ts, preciosPortNoL)

    plt.legend(("Activo S", "Valor Inicial del Portafolio", "Valor Presente",
                "Portafolio Optimo con Opcion", "Portafolio Arbitrario con Opcion",
                "Portafolio Arbitrario sin Opcion", "Portafolio Arbitrario con Opcion sin Tasa"),
               loc = "upper left")
    plt.show()
```

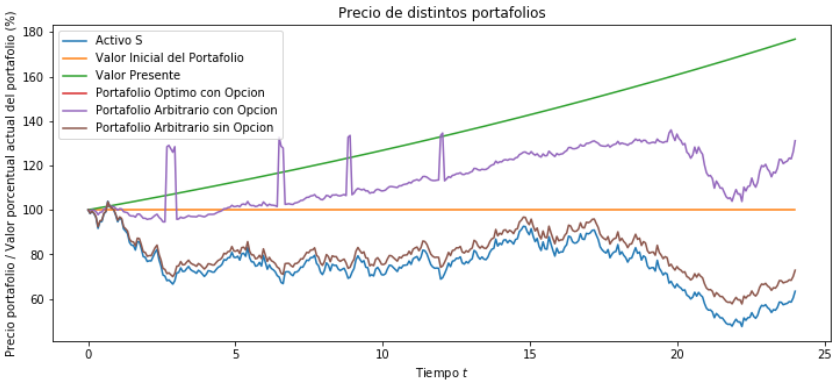
```
                Simulacion numero 1 con los mismos parametros mu, sigma, tau = 0.02 0.095 0.02375


/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: RuntimeWarning: divide by zero encountered in true_divide
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:9: RuntimeWarning: divide by zero encountered in true_divide
  if __name__ == '__main__':
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:65: RuntimeWarning: invalid value encountered in true_divide

Derivada de C respecto a S promedio: 0.9942357763816891
Desviacion estandar de derivada de C respecto a S promedio: 0.3228188388907163

Sigma calculada para el portafolio S: 0.02380497934109749
Sigma calculada para el portafolio C: 0.037595286555019015
Sigma calculada para el portafolio L: 8.424959336878418e-06
Portafolio optimo (S, C) compatible con tasa:
 [[ 1.97564665]
 [-0.97564665]]
Sigma calculada para el portafolio optimo de S, opcion: 4.676764072722233e-05
Sigma calculada para el portafolio sin opcion: 0.022583841105468884
Sigma calculada para el portafolio con opcion: 0.007285440454625087
Sigma calculada para el portafolio con opcion, sin tasa: 0.008640714739606797
```




```
                Simulacion numero 2 con los mismos parametros mu, sigma, tau = 0.02 0.095 0.02375

Derivada de C respecto a S promedio: 0.22140358346515449
Desviacion estandar de derivada de C respecto a S promedio: 5.13563393055204

Sigma calculada para el portafolio S: 0.025049235769633544
Sigma calculada para el portafolio C: 0.04922119027311974
Sigma calculada para el portafolio L: 8.424959336878418e-06

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: RuntimeWarning: divide by zero encountered in log
  if sys.path[0] == '':
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: RuntimeWarning: divide by zero encountered in double_scalars
  if sys.path[0] == '':
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: RuntimeWarning: invalid value encountered in double_scalars
  if sys.path[0] == '':

Portafolio optimo (S, C) compatible con tasa:
 [[nan]
 [nan]]
Sigma calculada para el portafolio optimo de S, opcion: nan
Sigma calculada para el portafolio sin opcion: 0.02285264843659464
Sigma calculada para el portafolio con opcion: 0.09531884061855321
Sigma calculada para el portafolio con opcion, sin tasa: 0.10500799366025296
```



```
                Simulacion numero 3 con los mismos parametros mu, sigma, tau = 0.02 0.095 0.02375

Derivada de C respecto a S promedio: 1.1561415303337217
Desviacion estandar de derivada de C respecto a S promedio: 3.1266894566727874

Sigma calculada para el portafolio S: 0.024465712937202035
Sigma calculada para el portafolio C: nan
Sigma calculada para el portafolio L: 8.424959336878418e-06
Portafolio optimo (S, C) compatible con tasa:
 [[nan]
 [nan]]
Sigma calculada para el portafolio optimo de S, opcion: nan
Sigma calculada para el portafolio sin opcion: 0.021933477998846602
Sigma calculada para el portafolio con opcion: 0.03713234641355896
Sigma calculada para el portafolio con opcion, sin tasa: 0.04401194932977211
```



La primera gráfica muestra un comportamiento con pocas eventualidades, y se observa cómo los portafolios con la opción call en corto son mucho menos erráticas que lor portafolios sin ella, y además logrando un mayor precio que el activo subyacente.

El portafolio optimo siempre tiene a la opción en corto. El riesgo de este portafolio es casi nulo en caso de existir. No existe si en algun momento la opcion llega a tener un valor nulo, en cuyo caso ya no se posible calcular rentabilidades.

La opción siempre tiene una mayor desviación estándar que S, lo cual se debe a que su comportamiento es muy similar, pero básicamente disminuido en una cantidad constante E1, de modo que los cambios experimentados son, en proporción, mayores.

Para un portafolio arbitrario con la opcion call en corto y con parte del capital invertido libre de reisgo, se observa una desviación desviación estándar menor que en el mismo portafolio pero con la opción eliminada y las demás inversiones escaladas proporcionalmente. Sin embargo, en la segunda gráfica no se observa esto, pues la línea morada tiene una mayor desviación estándar. **En este caso no podemos interpretar la desviación estándar como riesgo, puesta esta variación se debe a que la opción tomó un valor nulo debido a un incremento superior en S al permitido para ejercer la opción call, de modo que de este portafolio súbitamente varía para generar más ingresos, no puede variar para generar pérdidas. Lo mismo puede suceder si el precio de la opción toma valores por debajo del percio Strike, pues de nuevo observaremos un incremento del precio del portafolio, el cual introduce una varianza en la gráfica pero que solo puede ser para introducir ganancias**.

La rentabilidad de un portafolio compuesto de S y t puede variar desde no tener riesgo y poca rentabilidad, siendo idéntico a L, hasta ser idéntico a S, quizá con una mayor rentabilidad. Sin embargo, no es posible lograr lo que se logra en las simulaciones anteriores al introducir la opción, donde para un riesgo menor al de S en hasta 1 orden de magnitud, se obtienen rentabilidades superiores a las de S.

Con estos portafolios, parece posible reducir por mucho el riesgo de pérdida incluso si el precio del activo baja su precio, como muestra la tercera gráfica. Es más, los momentos en que el activo subyacente toma valores por debajo del precio strike, el valor del portafolio incremente significativamente. Esto se debe a que, en caso de que se ejerciera la opción, la ganancia neta debida a ésta por nuestra parte sería la del valor inicial de la opción más la diferencia entre el precio del subyacente y el precio strike, que en todo caso en positiva y puede, como en este caso, imponerse sobre las pérdidas que hay en la inversión en el activo S.

Múltiples Simulaciones

A continuación realizamos múltiples simulaciones con distintos valores de μ y σ y analizamos si, en general, los portafolios donde se invierte en la opción involucran un menor riesgo, medido en desviaciones estándar. Recordamos que no siempre es posible entender la desviación estándar como riesgo, especialmente cuando el valor de la opción se va a 0, pues la variación introucida en el precio del portafolio solo implica ganancias.

Llamamos un éxito aquello donde se cumple cierta condición. Estamos calculando 6 medidas de éxito:

- "Exitos en riesgo de uso de la opcion": hace referencia al porcentaje de portafolios en el que, con las mismas proporciones en el portafolio, introducir la opción disminuyó la desviación estándar.
- "Exitos en rentabilidad de uso de la opcion": hace referencia al porcentaje de veces que introducir la opción a un portafolio, manteniendo las proporciones, implicó una mayor rentabilidad que el mismo portafolio sin la opción.
- "Exitos en rentabilidad y riesgo de uso de la opcion": referente al porcentaje de portafolios en el que la introducción de la opción a un mismo portafolio, manteniendo las proporciones en los otros 2 activos, implicó tanto una mejora en rentabilidad como una disminución de desviación estándar.
- "Exitos en rentabilidad de uso de la tasa libre de riesgo": referente al porcentaje en el que la introducción de una inversión libre de riesgo a un portafolio con el activo y una opción call mejoró la rentabilidad.
- "Exitos en riesgo de uso de la tasa libre de riesgo": referente al porcentaje en el que la introducción de una inversión libre de riesgo a un portafolio con el activo y una opción call mejoró el riesgo.
- "Exitos en rentabilidad y riesgo de uso de la tasa libre de riesgo": referente a cuando las 2 anteriores fueron éxitos al tiempo.

In [163]:
```
mus  = np.array([0.001, 0.005, 0.015, 0.03])
#mus = [0.02]
sigs = [0.005, 0.095, 0.5, 0.1]
#sigs = [0.1]

#ultimosS = np.array([])
#ultimosMus = np.array([])
#ratios = np.array([])
#rendsMej2 = []
#coeficiente = []
exitosSigma = 0
exitosMu = 0
exitosDobles = 0

exitosTasaMu = 0
exitosTasaSig = 0
exitosTasaDob = 0

totales = 0


for j in range(1):
    for mmu in mus:
        for ssig in sigs:
            tau = mmu/2.

            for i in range(50):
                ## Simulacion i bajo las mismas condiciones
                ts, Ss = grafico_valor_activo(mu = mmu, sig = ssig, S0 = 100, N = 24, txtad = "Sim. 1", graficar = False, pts = 48)

                Cs = callV(S = Ss, t = ts, E1 = 70, E2 = 250, r = tau, sig = ssig, T = 30) # r = mu / 1.2
                Ls = 100.*np.exp(tau*ts)

                # Portafolio Optimo usando S y C compatible con la tasa libre de riesgo (Si existe):
                precios = np.array([Ss, Cs])
                portMej3, sig3, S = calcularPortOpt(precios, tau)

                # Si existe el portafolio, mirar en que aspectos fue mejor respecto a S
                if (portMej3[0,0] == portMej3[0,0]) or (portMej3[1,0] == portMej3[1,0]):
                    muMej3 = (portMej3[0,0]*100/Ss[0])*Ss[-1] + (portMej3[1,0]*100/Cs[0])*Cs[-1] #+ portMej3[2,0]*Ls

                    if muMej3 >= Ss[-1]:
                            exitosMu += 1

                    if sig3 <= np.sqrt(S[0, 0]):
                        exitosSigma += 1
                        if muMej3 >= Ss[-1]:
                            exitosDobles += 1

                    totales += 1


                # Otros portafolios

                lamOp = -0.8
                lamS = 1.65
                lamL = 0.15

                preciosPortCon = (100/Ss[0])*lamS * Ss                + (100/Cs[0])*lamOp * Cs                + (100/Ls[0])*lamL * Ls
                preciosPortSin = (100/Ss[0])*lamS/(lamS + lamL) * Ss  + 0 * Cs                                + (100/Ls[0])*lamL/(lamS + lamL) * Ls
                preciosPortNoL = (100/Ss[0])*lamS/(lamS + lamOp) * Ss + (100/Cs[0])*lamOp/(lamS + lamOp) * Cs + 0 * Ls

                retornosCon = np.log(preciosPortCon[1:]/preciosPortCon[:-1])
                retornosSin = np.log(preciosPortSin[1:]/preciosPortSin[:-1])
                retornosNoL = np.log(preciosPortNoL[1:]/preciosPortNoL[:-1])

                riesgoSTDCon = np.std(retornosCon)
                riesgoSTDSin = np.std(retornosSin)
                riesgoSTDNoL = np.std(retornosNoL)

                muUltCon = preciosPortCon[-1]
                muUltSin = preciosPortSin[-1]
                muUltNoL = preciosPortNoL[-1]

                # Mirar si fue exitoso usar la opcion respecto a no usarla
                if muUltCon >= muUltSin:
                    exitosMu += 1
                if riesgoSTDCon <= riesgoSTDSin:
                    exitosSigma += 1
                    if muUltCon >= muUltSin:
                        exitosDobles += 1

                # Mirar si fue exitoso combinar los portafolios con la tasa libre de riesgo
                if muUltCon >= muUltNoL:
                    exitosTasaMu += 1
                if riesgoSTDCon <= riesgoSTDNoL:
                    exitosTasaSig += 1
                    if muUltCon >= muUltNoL:
                        exitosTasaDob += 1

                totales += 1

print("Exitos en riesgo de uso de la opcion:", 100*exitosSigma/totales)
print("Exitos en rentabilidad de uso de la opcion:", 100*exitosMu/totales)
print("Exitos en rentabilidad y riesgo de uso de la opcion:", 100*exitosDobles/totales)

print("Exitos en rentabilidad de uso de la tasa libre de riesgo:", 100*exitosTasaMu/totales)
print("Exitos en riesgo de uso de la tasa libre de riesgo:", 100*exitosTasaSig/totales)
print("Exitos en rentabilidad y riesgo de uso de la tasa libre de riesgo:", 100*exitosTasaDob/totales)

print("Simulaciones totales: ", totales)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: RuntimeWarning: divide by zero encountered in true_divide
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:9: RuntimeWarning: divide by zero encountered in true_divide
  if __name__ == '__main__':
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: RuntimeWarning: divide by zero encountered in log
  if sys.path[0] == '':
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: RuntimeWarning: divide by zero encountered in double_scalars
  if sys.path[0] == '':
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: RuntimeWarning: invalid value encountered in double_scalars
  if sys.path[0] == '':
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:65: RuntimeWarning: invalid value encountered in log
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:63: RuntimeWarning: invalid value encountered in log
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: RuntimeWarning: invalid value encountered in add
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: RuntimeWarning: invalid value encountered in subtract
```

```
Exitos en riesgo de uso de la opcion: 78.31031681559708
Exitos en rentabilidad de uso de la opcion: 42.24207961007311
Exitos en rentabilidad y riesgo de uso de la opcion: 30.950446791226646
Exitos en rentabilidad de uso de la tasa libre de riesgo: 40.861088545897644
Exitos en riesgo de uso de la tasa libre de riesgo: 61.41348497156783
Exitos en rentabilidad y riesgo de uso de la tasa libre de riesgo: 38.18034118602762
Simulaciones totales:  1231
```

Rango mas reducido, con μ y σ consistentes

In [166]:

```
mus  = np.linspace(0.01, 0.05, 10)
sigs = np.linspace(0.05, 0.18, 10)

#ultimosS = np.array([])
#ultimosMus = np.array([])
#ratios = np.array([])
#rendsMej2 = []
#coeficiente = []
exitosSigma = 0
exitosMu = 0
exitosDobles = 0

exitosTasaMu = 0
exitosTasaSig = 0
exitosTasaDob = 0

totales = 0


for j in range(1):
    for mmu in mus:
        for ssig in sigs:
            tau = mmu/2.

            for i in range(10):
                ## Simulacion i bajo las mismas condiciones
                ts, Ss = grafico_valor_activo(mu = mmu, sig = ssig, S0 = 100, N = 24, txtad = "Sim. 1", graficar = False, pts = 48)

                Cs = callV(S = Ss, t = ts, E1 = 70, E2 = 250, r = tau, sig = ssig, T = 30) # r = mu / 1.2
                Ls = 100.*np.exp(tau*ts)

                # Portafolio Optimo usando S y C compatible con la tasa libre de riesgo (Si existe):
                precios = np.array([Ss, Cs])
                portMej3, sig3, S = calcularPortOpt(precios, tau)

                # Si existe el portafolio, mirar en que aspectos fue mejor respecto a S
                if (portMej3[0,0] == portMej3[0,0]) or (portMej3[1,0] == portMej3[1,0]):
                    muMej3 = (portMej3[0,0]*100/Ss[0])*Ss[-1] + (portMej3[1,0]*100/Cs[0])*Cs[-1] #+ portMej3[2,0]*Ls

                    if muMej3 >= Ss[-1]:
                            exitosMu += 1

                    if sig3 <= np.sqrt(S[0, 0]):
                        exitosSigma += 1
                        if muMej3 >= Ss[-1]:
                            exitosDobles += 1

                    totales += 1


                # Otros portafolios

                lamOp = -0.8
                lamS = 1.65
                lamL = 0.15

                preciosPortCon = (100/Ss[0])*lamS * Ss                + (100/Cs[0])*lamOp * Cs                + (100/Ls[0])*lamL * Ls
                preciosPortSin = (100/Ss[0])*lamS/(lamS + lamL) * Ss  + 0 * Cs                                + (100/Ls[0])*lamL/(lamS + lamL) * Ls
                preciosPortNoL = (100/Ss[0])*lamS/(lamS + lamOp) * Ss + (100/Cs[0])*lamOp/(lamS + lamOp) * Cs + 0 * Ls

                retornosCon = np.log(preciosPortCon[1:]/preciosPortCon[:-1])
                retornosSin = np.log(preciosPortSin[1:]/preciosPortSin[:-1])
                retornosNoL = np.log(preciosPortNoL[1:]/preciosPortNoL[:-1])

                riesgoSTDCon = np.std(retornosCon)
                riesgoSTDSin = np.std(retornosSin)
                riesgoSTDNoL = np.std(retornosNoL)

                muUltCon = preciosPortCon[-1]
                muUltSin = preciosPortSin[-1]
                muUltNoL = preciosPortNoL[-1]

                # Mirar si fue exitoso usar la opcion respecto a no usarla
                if muUltCon >= muUltSin:
                    exitosMu += 1
                if riesgoSTDCon <= riesgoSTDSin:
                    exitosSigma += 1
                    if muUltCon >= muUltSin:
                        exitosDobles += 1

                # Mirar si fue exitoso combinar los portafolios con la tasa libre de riesgo
                if muUltCon >= muUltNoL:
                    exitosTasaMu += 1
                if riesgoSTDCon <= riesgoSTDNoL:
                    exitosTasaSig += 1
                    if muUltCon >= muUltNoL:
                        exitosTasaDob += 1

                totales += 1

print("Exitos en riesgo de uso de la opcion:", 100*exitosSigma/totales)
print("Exitos en rentabilidad de uso de la opcion:", 100*exitosMu/totales)
print("Exitos en rentabilidad y riesgo de uso de la opcion:", 100*exitosDobles/totales)

print("Exitos en rentabilidad de uso de la tasa libre de riesgo:", 100*exitosTasaMu/totales)
print("Exitos en riesgo de uso de la tasa libre de riesgo:", 100*exitosTasaSig/totales)
print("Exitos en rentabilidad y riesgo de uso de la tasa libre de riesgo:", 100*exitosTasaDob/totales)

print("Simulaciones totales: ", totales)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: RuntimeWarning: divide by zero encountered in true_divide
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:9: RuntimeWarning: divide by zero encountered in true_divide
  if __name__ == '__main__':
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: RuntimeWarning: divide by zero encountered in log
  if sys.path[0] == '':
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: RuntimeWarning: divide by zero encountered in double_scalars
  if sys.path[0] == '':
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: RuntimeWarning: invalid value encountered in double_scalars
  if sys.path[0] == '':
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: RuntimeWarning: invalid value encountered in add
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: RuntimeWarning: invalid value encountered in subtract

Exitos en riesgo de uso de la opcion: 73.8857938718663
Exitos en rentabilidad de uso de la opcion: 55.571030640668525
Exitos en rentabilidad y riesgo de uso de la opcion: 33.42618384401114
Exitos en rentabilidad de uso de la tasa libre de riesgo: 26.601671309192202
Exitos en riesgo de uso de la tasa libre de riesgo: 69.63788300835654
Exitos en rentabilidad y riesgo de uso de la tasa libre de riesgo: 26.601671309192202
Simulaciones totales:  1436
```

Vemos que, en efecto, la introdución de la opción disminuye en más del 70% de los casos la desviación estándar, y en más de un 50% de los casos mejora la rentabilidad del portafolio, aunque solo en un 30% de los casos estas mejoras son simultáneas.

Como era de esperarse, aunque sorpresivo por su bajo valor, es que la introducción de una inversión libre de riesgo disminuye el riesgo del portafolio en más de un 60% de los casos.

Apéndice: Probando la Funcion Call

In [89]:
```
#callV(S = 100, t = 20, E1 = 80, E2 = 200, r = 0.015, sig = 0.09, T = 100)
#ts = np.linspace(0, 7, 100)
#Ss = np.linspace(100, 300, 100)
#cs = callV(S = Ss, t = ts, E1 = 70, E2 = 250, r = 0.01, sig = 0.02, T = 10) # Funciona chevere

mmu = 0.02
ssig = 0.05
tau = mmu/2.

ts, Ss = grafico_valor_activo(mu = mmu, sig = ssig, S0 = 100, N = 48, txtad = "Sim. 1", graficar = False, pts = 120)
cs = callV(S = Ss, t = ts, E1 = 70, E2 = 300, r = tau, sig = ssig, T = 24)

plt.figure()
plt.title("Valor de la opcion ventana si S0 = 100\nE1 = 70, E2 = 250, r = 0.01, sig = 0.02, T = 10")
plt.plot(ts, Ss)
plt.plot(ts, cs)

plt.legend(("S", "C"), loc = "upper left")
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: RuntimeWarning: divide by zero encountered in true_divide
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:9: RuntimeWarning: divide by zero encountered in true_divide
  if __name__ == '__main__':
```

Out[89]:
```
<matplotlib.legend.Legend at 0x7fab141afeb8>
```

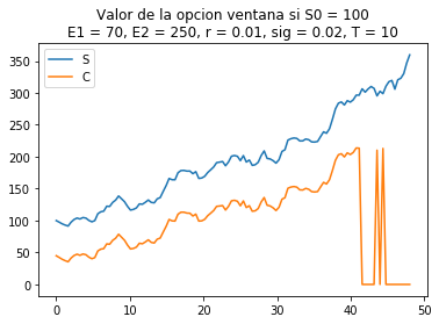
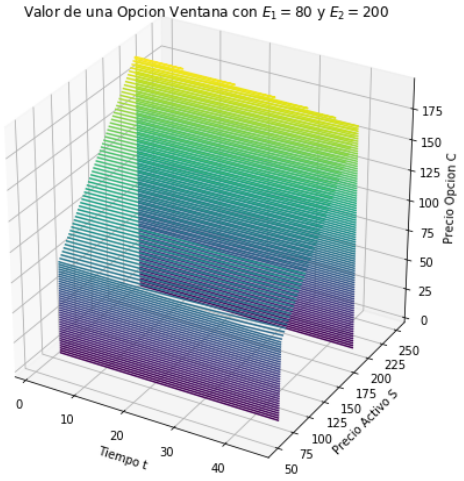


Grafico 3D del precio de la Opción

In [186]:
```
vCallV = np.vectorize(callV)

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(111, projection='3d')
ts = np.linspace(1, 45.1, 100)
ss = np.linspace(50, 250, 100)
Ts, Ss = np.meshgrid(ts,
                     ss)
#Ts = Ts.T
#Ss = Ss.T
#print("ts,\n", ts)
#print("ss,\n", ss)
Cs = vCallV(S = Ss, t = Ts, E1 = 80, E2 = 200, r = 0.03, sig = 0.09, T = 100)
#print("Cs,\n", Cs)

#ax.plot_surface(Ss, Ts, Cs)
ax.contour3D(Ts, Ss, Cs, 100)#np.asarray([[1, 2, 3]], [[4, 3, 2]], [[2, 3, 4]])
ax.set_title("Valor de una Opcion Ventana con $E_1 = 80$ y $E_2 = 200$")
ax.set_xlabel('Tiempo t')
ax.set_ylabel('Precio Activo S')
ax.set_zlabel('Precio Opcion C');
plt.savefig('Precio_Call_Ventana.png')
plt.savefig('Precio_Call_Ventana.jpg')
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: RuntimeWarning: divide by zero encountered in double_scalars
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:9: RuntimeWarning: divide by zero encountered in double_scalars
  if __name__ == '__main__':
```



/