""" Exercițiul 2 → Rezolvare """

$$f(x,y) = \frac{1}{2} * < A\begin{pmatrix} x \\ y \end{pmatrix}, \begin{pmatrix} x \\ y \end{pmatrix} > - < \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}, \begin{pmatrix} x \\ y \end{pmatrix} > =$$

$$= \frac{1}{2}a_{11}x^2 + \frac{1}{2}a_{22}y^2 + \frac{1}{2}a_{12}xy + \frac{1}{2}a_{21}xy - b_1x - b_2y \qquad (1)$$

Datele Problemei → V15

$$f(x,y) = \frac{7}{2}x^2 + xy + 3x + \frac{7}{2}y^2 - y \qquad (2)$$

Relațiile (1) și (2) sunt egale, deci putem afla valorile pentru matricea A și pentru vectorul b.

$$\begin{cases} \frac{1}{2}a_{11}x^2 = \frac{7}{2}x^2 \\ \frac{1}{2}a_{22}y^2 = \frac{7}{2}y^2 \\ \frac{1}{2}a_{12}xy + \frac{1}{2}a_{21}xy = xy \\ -b_1x = 3x \\ -b_2y = -y \end{cases}$$

Astfel, $a_{11} = 7, a_{22} = 7, a_{12} = a_{21} = 1, b_1 = -3, b_2 = 1$.

Răspuns: A = $\begin{pmatrix} 7 & 1 \\ 1 & 7 \end{pmatrix}$, b = $\begin{pmatrix} -3 \\ 1 \end{pmatrix}$. Matricea A este o matrice pătratică, simetrică, pozitiv definită (aplicând criteriul Sylvester). Funcția admite punct de minim local, iar acest punct coincide cu soluția sistemului Ax = b. (Informații și Demonstrații din Cursul #6, 12.11.2020)

In [1]:
```python
"""
Created on Sat Dec 12 17:28:46 2020

@author: [redacted]
@grupa:     341
@nr.crt:    159
@varianta:  15
"""
```

Out[1]:
```
'\nCreated on Sat Dec 12 17:28:46 2020\n\n@author: [redacted]\n@grupa:     341\n@nr.crt:    159\n@varianta:  15\n'
```

In [2]:
```python
# Librării
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:
```python
"""     Exercițiul 1 -> Rezolvare      """

# Datele Problemei -> Subpunctul 1
n = 5

b = np.zeros(n)
for i in range(1, n + 1):
    b[i - 1] = i ** 4
print(f"b: {b}\n")

# Definim Vectorul a
a = np.zeros(n)
for i in range(n, -1, -1):
    a[i % n] = 2 ** (n - (i % n))
print(f"Vectorul a: {a}\n")

# Definim Matricea Simetrică A
A = np.zeros((n, n))
for i in range(n):
    for j in range(i, n):
        A[i][j] = a[j - i]
        A[j][i] = a[j - i]
print(f"Matricea Simetrică A: \n{A}")
```

```
b: [  1.  16.  81. 256. 625.]

Vectorul a: [32. 16.  8.  4.  2.]

Matricea Simetrică A:
[[32. 16.  8.  4.  2.]
 [16. 32. 16.  8.  4.]
 [ 8. 16. 32. 16.  8.]
 [ 4.  8. 16. 32. 16.]
 [ 2.  4.  8. 16. 32.]]
```

In [4]:
```python
# Subpunctul 2 -> Criteriul Sylvester
def Sylvester(A):
    ok = 0
    for i in range(1, n + 1):
        aux = A[0:i, 0:i]
        if np.linalg.det(aux) <= 0:
            print("Matricea NU este Pozitiv Definită!")
            ok = 1

    return ok


if Sylvester(A) == 0:
    print("Matricea este Pozitiv Definită!")
```

```
Matricea este Pozitiv Definită!
```

In [5]:
```python
# Subpunctul 3 -> Gradient Conjugat (am mai adăugat parametrul x pentru a putea
# rezolva și ex3; x este ales arbitrat; pt acest exercițiu x = None)
def GradConjugat(A, b, eps, x = None):
    """

    Parameters
    ----------
    A : matrice simetrică, pozitiv definită.
    b : vector termeni liberi.
    eps : epsilon / toleranța.
    x : inițial.

    Returns
    -------
    x = soluția sistemului (A * x = b).
    x_array = vectorul cu toți x intermediari
    """

    # Verificăm dacă matricea este pătratică
    m, n = np.shape(A)
    if m != n:
        print("Matricea nu este pătratică. Introduceți altă matrice.")
        return None

    # Verificăm dacă matricea este simetrică
    A_transpus = np.transpose(A)
    if not np.array_equal(A, A_transpus):
        print("Matricea NU este Simetrică!")
        return None

    # Verificăm dacă matricea este pozitiv definită
    if Sylvester(A) == 1:
        return None

    # x(0) -> ales arbitrat, asa că îl aleg 0
    if x is None:
        x = np.zeros(b.shape)
        r = b                    # pt că am ales x = 0; altfel, r = b - A@x
    else:
        r = b - A@x

    x_array = np.empty(len(b) + 1, dtype = object)
    x_array[0] = x # vectorul unde voi reține toți x, pt viitoarele exerciții
    p = r
    rsold = np.transpose(r)@r

    # Algoritm
    for i in range(n):
        Ap = A@p
        alpha = rsold / (np.transpose(p)@Ap)
        x = x + alpha * p
        x_array[i + 1] = x
        r = r - alpha * Ap
        rsnew = np.transpose(r)@r

        if np.sqrt(rsnew) < eps:
            break

        p = r + (rsnew / rsold) * p
        rsold = rsnew


    return x, x_array
```

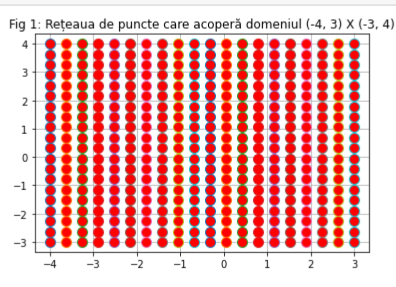
In [6]:
```python
# Subpunctul 4 -> Rezolvare Sistem
eps = 10 ** (-10)
x, x_array = GradConjugat(A, b, eps)
print(f'Soluția Sistemului x: \n{x}')
print(f'Verificare:\nb: {b}\nA * x: {A@x}')
```

```
Soluția Sistemului x:
[-0.29166667 -0.875      -1.44791667 -1.375      20.70833333]
Verificare:
b: [  1.  16.  81. 256. 625.]
A * x: [  1.  16.  81. 256. 625.]
```

In [7]:
```python
"""     Exercițiul 3 -> Rezolvare      """

# Datele Problemei
a = -4
b = 3
c = -3
d = 4

A = np.array([[7., 1.],
              [1., 7.]])
b_vector = np.array([[-3], [1]])

def f(x, y):
    return (7/2) * (x ** 2) + x * y + 3 * x + (7/2) * (y ** 2) - y
```

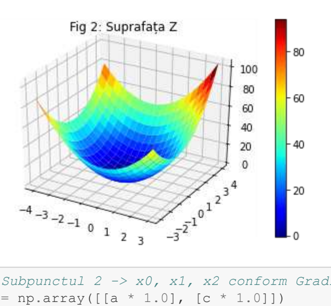
In [8]:
```python
# Subpunctul 1 -> Suprafața z = f(x, y) pe domeniul dat
(Nx, Ny) = (20, 20) #nr de puncte în care să fie împărțit intervalul

x_grafic = np.linspace(a, b, Nx)
y_grafic = np.linspace(c, d, Ny)
[X, Y] = np.meshgrid(x_grafic, y_grafic)

plt.figure(1)
plt.plot(X, Y, 'o', markerfacecolor = 'red', markersize = 10)
plt.grid(True)
plt.title(f'Fig 1: Rețeaua de puncte care acoperă domeniul ({a}, {b}) X ({c}, {d})')
plt.show()

Z = f(X, Y)

fig = plt.figure(2)
axes = plt.axes(projection = '3d')
surf = axes.plot_surface(X, Y, Z, cmap = plt.cm.jet)
plt.title('Fig 2: Suprafața Z')
fig.colorbar(surf)
```

Fig 1: Rețeaua de puncte care acoperă domeniul (-4, 3) X (-3, 4)



Out[8]:
```
<matplotlib.colorbar.Colorbar at 0x1758e1cb850>
```

Fig 2: Suprafața Z

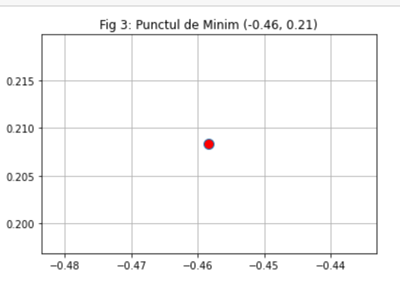


In [9]:
```python
# Subpunctul 2 -> x0, x1, x2 conform Gradientului Conjugat
x = np.array([[a * 1.0], [c * 1.0]])
x, x_array = GradConjugat(A, b_vector, eps, x)
for i in range(3):
    print(f'x^({i}): \n{x_array[i]}\n')
print(f"Punct Minim = x:\n{x}")
```

```
x^(0):
[[-4.]
 [-3.]]

x^(1):
[[-0.49880096]
 [ 0.25111339]]

x^(2):
[[-0.45833333]
 [ 0.20833333]]

Punct Minim = x:
[[-0.45833333]
 [ 0.20833333]]
```

In [10]:
```python
# Subpunctul 3 -> Punctul Minim
plt.figure(3)
plt.plot(x[0][0], x[1][0], 'o', linewidth = 3,
            markerfacecolor = 'red', markersize = 10)
plt.grid()
plt.title(f'Fig 3: Punctul de Minim ({"{:.2f}".format(x[0][0])}, {"{:.2f}".format(x[1][0])})')
plt.show()
```

Fig 3: Punctul de Minim (-0.46, 0.21)



In [11]:
```python
# Subpunctul 4 -> Curbele de Nivel si Traseul
a = np.zeros(3)
b = np.zeros(3)
for i in range(3):
    a[i] = x_array[i][0][0]
    b[i] = x_array[i][1][0]

for i in range(3):
    aux = f(x_array[i][0][0], x_array[i][1][0])
    plt.contour(X, Y, Z, levels = [aux])
plt.show()

plt.title('Fig 4.1: Curbele de Nivel (Apropiere)')
plt.plot(a, b, 'o-', markersize = 10, color = 'r', mfc = 'blue')
plt.show()



plt.title('Fig 4.2: Curbele de Nivel (Distanță)')
x_grafic = np.linspace(-6, 5, Nx)         # ca să se vadă desenul mai clar
y_grafic = np.linspace(-5.5, 5.7, Ny)
[X, Y] = np.meshgrid(x_grafic, y_grafic)
Z = f(X, Y)

for i in range(3):
    aux = f(x_array[i][0][0], x_array[i][1][0])
    plt.contour(X, Y, Z, levels = [aux])
plt.show()

plt.plot(a, b, 'o-', markersize = 10, color = 'r', mfc = 'blue')
plt.show()
```

Fig 4.1: Curbele de Nivel (Apropiere)

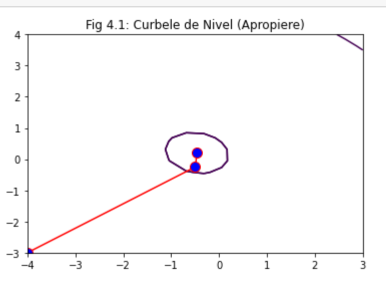
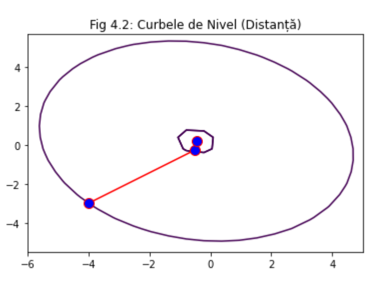


Fig 4.2: Curbele de Nivel (Distanță)

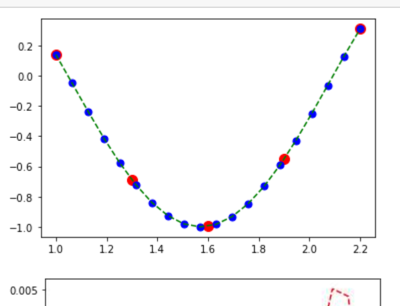


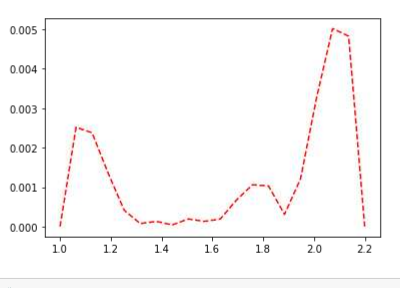
In [12]:
```python
"""     Exercițiul 4 -> Rezolvare      """

# Datele Problemei
n = 4
(a, b) = (1, 2.2)
def f(x):
    return np.sin(3 * x)
```

In [13]:
```python
# Subpunctul a)
# Funcție Auxiliară: Metodă Substituție Ascendentă
def metSubAsc(A, b, tol):
    """

    Parameters
    ----------
    A : matrice inferior triunghiulară.
    b : vectorul termenilor liberi.
    tol : toleranța.

    Returns
    -------
    soluția.

    """

    # Verificăm dacă matricea este pătratică
    m, n = np.shape(A)
    if m!= n:
        print("Matricea nu este pătratică. Introduceți altă matrice.")
        x = None
        return x

    # Verificăm dacă matricea este superior triunghiulară
    for i in range(m):
       for j in range(i):
           if abs(A[j][i]) > tol:
               print("Matricea nu este inferior triunghiulară.")
               x = None
               return x

    # Verificam dacă toate elementele de pe diagonala principală sunt nenule =>
    # Si. este compatibil ddeterminat (adică am soluție unică)
    for i in range(n):
        if abs(A[i][i]) <= tol:
            print("Sistemul nu este compatibil determinat.")
            x = None
            return x

    x = np.zeros((m, 1))
    x[0] = b[0] / A[0][0]

    for k in range(1, n):

        sum = 0
        for j in range(k):
            sum += A[k][j] * x[j]

        x[k] = (1 / A[k][k]) * (b[k] - sum)

    return x


# Metoda Newton Propriu-zisă
def MetNewton(X, Y, x):
    n = len(X) - 1
    A = np.zeros((n + 1, n + 1))

    # Determinăm A
    for i in range(n + 1):
        A[i][0] = 1

    for i in range(1, n + 1):
        for j in range(1, i + 1):
            p = 1
            for k in range(j):
                p *= (X[i] - X[k])
            A[i][j] = p

    # Aplic metSubAsc pt aflarea valorilor c1, c2, ..., cn + 1
    c = metSubAsc(A, Y, 10 ** (-10))

    # Determin Polinomul
    Pn = c[0][0]
    for i in range(1, n + 1):
        p = c[i][0]
        for j in range(i):
            p *= (x - X[j])
        Pn += p

    return Pn
```

In [14]:
```python
# Subpunctul b) -> Metoda Newton cu Diferențe Divizate
def MetNDD(X, Y, x):
    nx = len(x)
    Pn = [0.0] * (nx)

    def a(j0, j1 = None):
        if j1 is None: j1 = j0; j0 = 0
        if j0 == j1: return Y[j0]
        elif j1 - j0 == 1:
            return (Y[j1] - Y[j0]) / (X[j1] - X[j0])
        else:
            return (a(j0 + 1, j1) - a(j0, j1 - 1)) / (X[j1] - X[j0])

    def z(j, x_):
        v = 1.0
        for i in range(j):
            v *= x_ - X[i]
        return v

    for i in range(nx):
        for j in range(n + 1):
            Pn[i] += a(j) * z(j, x[i])
    return Pn
```

In [15]:
```python
# Subpunctul c)
X = np.linspace(a, b, n + 1)
Y = f(X)
x_graf = np.linspace(a, b, 20)
y_graf = f(x_graf)


y = MetNewton(X, Y, x_graf)

plt.plot(X, Y, 'o', color='red', markersize=10)
plt.plot(x_graf, y_graf, 'o', color='blue', markersize=7)
plt.plot(x_graf, y, '--', color='green', markersize=4)
plt.show()

E = np.abs(y_graf - y)
plt.plot(x_graf, E, '--', color='red')
plt.show()
```





In [16]:
```python
# Subpunctul d)
y = MetNDD(X, Y, x_graf)

plt.plot(X, Y, 'o', color='red', markersize=10)
plt.plot(x_graf, y_graf, 'o', color='blue', markersize=7)
plt.plot(x_graf, y, '--', color='green', markersize=4)
plt.show()

E = np.abs(y_graf - y)
plt.plot(x_graf, E, '--', color='red')
plt.show()
```