Question 2. Create a 2D version of the random-walk diffusion, by extending sto-diff-1d-comp.py. Compare the result against a 2D analytic theory.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import random
```

In [2]:

```
# Number of particles to use for simulation
N = 10000

# Size of the unit walk step.
dx = 1e-1

# Max steps of simulation
smax = 1000

# Number of bins for density distribution plot
nBin = 10
```

diffusion coefficient is approximated to $0.5(\delta x/dt)^2$ Since step size is constant for all directions, it is equal to 1 dimension case.

In [3]:

```
# Calculation of theoretical parameters
D = 0.5 * (dx) ** 2  # suppose dt=1
tau = 1.0 / (D * np.pi * np.pi)

# Zero-initialized particle positions
x=np.array(([0]*2)*N).reshape((N,2))
```

Set the x,y coordinate arra

In [4]:

```
t = np.arange(N) / N
x = t*0
y = t*0
```

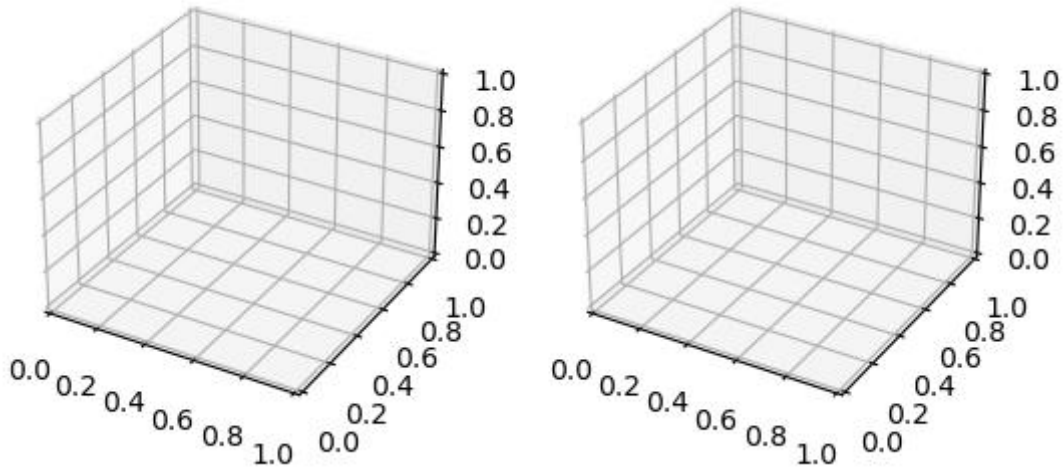
Create bin meshes for density distribution

In [5]:

```
f = np.zeros((nBin + 1, nBin + 1))
bnX = np.arange(0, nBin + 1)
bnX = bnX / (1.0 * nBin)  # x-value for the distr-plot
```

In [6]:

```
X, Y = np.meshgrid(bnX, bnX)
fig = plt.figure()
ax1=fig.add_subplot(121,projection='3d') # analytical value
ax2=fig.add_subplot(122,projection='3d') # by random walk
```




Set the location of N particles so that each x,y coordinate distribution follow probability of sine distribution.

In [7]:

```
i=0
while i< N:
    V=random.random()
    T=random.random()
    val=np.sin(np.pi*V)
    if T<= val:
        x[i] = V
        i+=1
```

In [8]:

```
i=0
while i< N:
    V=random.random()
    T=random.random()
    val=np.sin(np.pi*V)
    if T<= val:
        y[i] = V
        i+=1
```

In [9]:

```
for i in range(1): print(i)
```

0

main loop

- 2dimension density distributuion

```
f[m, n] += (1.0 - nx) * (1.0 - ny)
f[m + 1, n] += nx * (1.0 - ny)
f[m, n + 1] += (1.0 - nx) * ny
f[m + 1, n + 1] += nx * ny
```

- 2dimension random walk

```
r = random.choice([0,1])
if r == 0 :
    x[i] = x[i] + random.choice([dx, -dx])
else :
    y[i] = y[i] + random.choice([dx, -dx])
```

- analytical solution for 2D diffusion equation

$$z = A\sin(\pi x)\sin(\pi y)e^{\frac{-t}{\tau}} (Aisconstant)$$

$$\int_0^1 \int_0^1 \sin(\pi x)\sin(\pi y)dxdy = (\frac{2}{\pi})^2$$

Therefore, normalization factor is $(\frac{\pi}{2})^2$ There are N particles. We should multiple N.

$$z = (\frac{\pi}{2})^2 N\sin(\pi x)\sin(\pi y)e^{\frac{-t}{\tau}}$$

Jupyter notebook cannot show figure in loop. So, I checked it for 20 steps. In q2.py, you can simulate more conveniently.

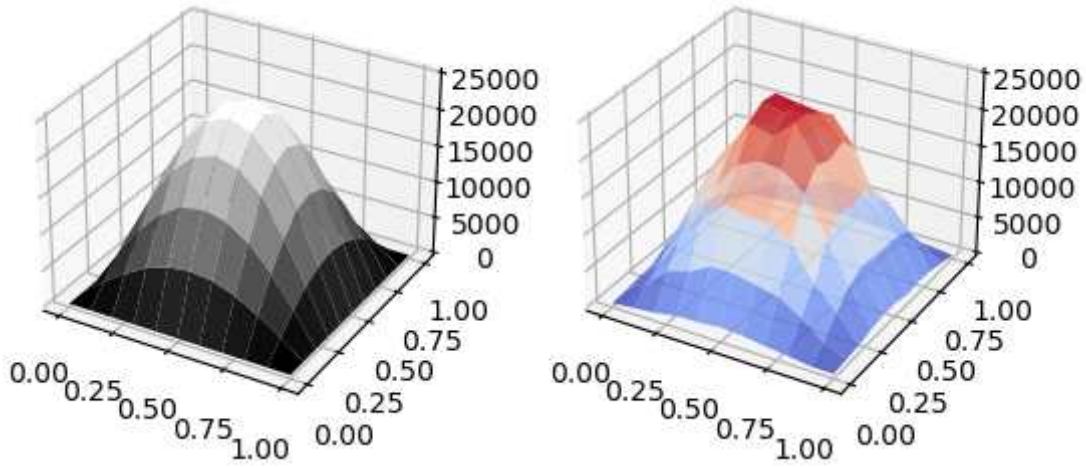
In [10]:

```
# Main loop : when initial condition
for s in range(1):
    f = 0.0 * f  # empty the bin for new constructon of the distr. every step
    for i in range(N):
        if 0 <= x[i] and x[i] < 1 and 0 <= y[i] and y[i] < 1:
            nx = x[i] * (1.0 * nBin)  # normalize the potision to bin
            m = int(nx)  # find the bin index
            nx -= m  # find the relative displacement in bin
            ny = y[i] * (1.0 * nBin)
            n = int(ny)
            ny -= n

            # 2 dimensional density distribution

            f[m, n] += (1.0 - nx) * (1.0 - ny)
            f[m + 1, n] += nx * (1.0 - ny)
            f[m, n + 1] += (1.0 - nx) * ny
            f[m + 1, n + 1] += nx * ny

            # 2 dimensional randomwalk
            r = random.choice([0,1])
            if r == 0 :
                x[i] = x[i] + random.choice([dx, -dx])
            else :
                y[i] = y[i] + random.choice([dx, -dx])

            # x[i] = x[i] + random.choice([dx, -dx])
            # y[i] = y[i] + random.choice([dx, -dx])  # random walk

    f = 1.0 * nBin * 1.0 * nBin * f  # normalization factor for distri.
    # plt.ylim(0,1.2*np.pi*N/2.0)
    z = (np.pi / 2) ** 2 * N * np.sin(np.pi * X) * np.sin(np.pi * Y) * np.exp(-(1.0 * s) / tau)
    if s%100 == 0 :
        ax1.clear()
        ax2.clear()
        ax1.set_zlim(0, N * 2.5)
        ax2.set_zlim(0, N * 2.5)
        surf1 = ax1.plot_surface(X, Y, z, cmap='gray')
        surf2 = ax2.plot_surface(X, Y, f, cmap='coolwarm', alpha = 0.8)
```

In [11]:

```
fig
```

Out[11]:



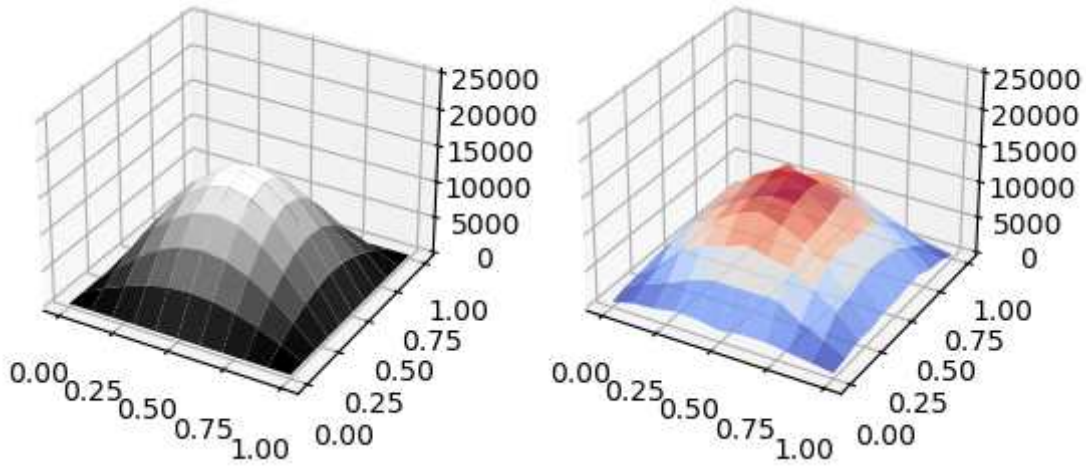
In [12]:

```
# Main loop
for s in range(20): #after 20
    f = 0.0 * f  # empty the bin for new constructon of the distr. every step
    for i in range(N):
        if 0 <= x[i] and x[i] < 1 and 0 <= y[i] and y[i] < 1:
            nx = x[i] * (1.0 * nBin)  # normalize the potision to bin
            m = int(nx)  # find the bin index
            nx -= m  # find the relative displacement in bin
            ny = y[i] * (1.0 * nBin)
            n = int(ny)
            ny -= n

            # 2 dimensional density distribution

            f[m, n] += (1.0 - nx) * (1.0 - ny)
            f[m + 1, n] += nx * (1.0 - ny)
            f[m, n + 1] += (1.0 - nx) * ny
            f[m + 1, n + 1] += nx * ny

            # 2 dimensional randomwalk
            r = random.choice([0,1])
            if r == 0 :
                x[i] = x[i] + random.choice([dx, -dx])
            else :
                y[i] = y[i] + random.choice([dx, -dx])

            # x[i] = x[i] + random.choice([dx, -dx])
            # y[i] = y[i] + random.choice([dx, -dx])  # random walk

    f = 1.0 * nBin * 1.0 * nBin * f  # normalization factor for distri.
    # plt.ylim(0,1.2*np.pi*N/2.0)
    z = (np.pi / 2) ** 2 * N * np.sin(np.pi * X) * np.sin(np.pi * Y) * np.exp(-(1.0 * s) / tau)
    if s%10 == 0 :
        ax1.clear()
        ax2.clear()
        ax1.set_zlim(0, N * 2.5)
        ax2.set_zlim(0, N * 2.5)
        surf1 = ax1.plot_surface(X, Y, z, cmap='gray')
        surf2 = ax2.plot_surface(X, Y, f, cmap='coolwarm', alpha = 0.8)
```

In [13]:

```
fig
```

Out[13]:



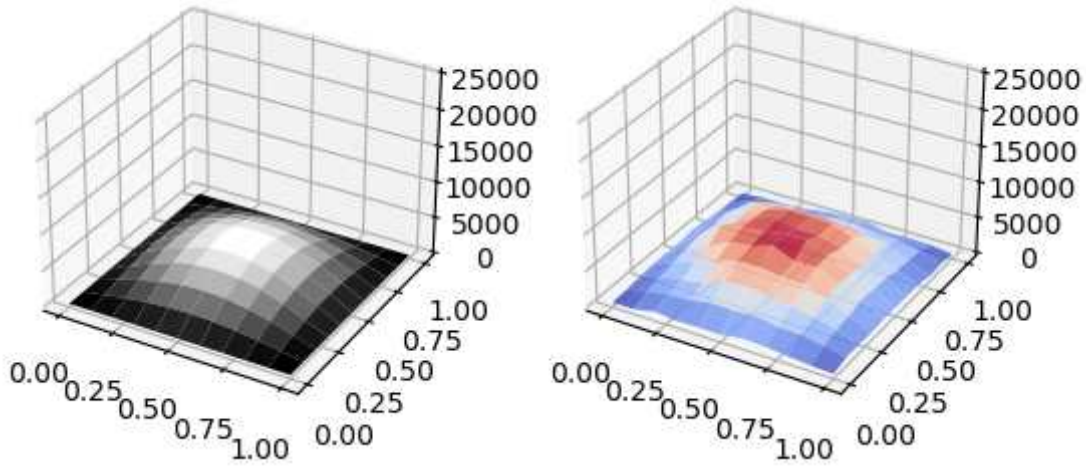
In [14]:

```
# Main loop
for s in range(20): # after 40 step
    f = 0.0 * f  # empty the bin for new constructon of the distr. every step
    for i in range(N):
        if 0 <= x[i] and x[i] < 1 and 0 <= y[i] and y[i] < 1:
            nx = x[i] * (1.0 * nBin)  # normalize the potision to bin
            m = int(nx)  # find the bin index
            nx -= m  # find the relative displacement in bin
            ny = y[i] * (1.0 * nBin)
            n = int(ny)
            ny -= n

            # 2 dimensional density distribution

            f[m, n] += (1.0 - nx) * (1.0 - ny)
            f[m + 1, n] += nx * (1.0 - ny)
            f[m, n + 1] += (1.0 - nx) * ny
            f[m + 1, n + 1] += nx * ny

            # 2 dimensional randomwalk
            r = random.choice([0,1])
            if r == 0 :
                x[i] = x[i] + random.choice([dx, -dx])
            else :
                y[i] = y[i] + random.choice([dx, -dx])

            # x[i] = x[i] + random.choice([dx, -dx])
            # y[i] = y[i] + random.choice([dx, -dx])  # random walk

    f = 1.0 * nBin * 1.0 * nBin * f  # normalization factor for distri.
    # plt.ylim(0,1.2*np.pi*N/2.0)
    z = (np.pi / 2) ** 2 * N * np.sin(np.pi * X) * np.sin(np.pi * Y) * np.exp(-(1.0 * (s+20)) /
    if s%10 == 0 :
        ax1.clear()
        ax2.clear()
        ax1.set_zlim(0, N * 2.5)
        ax2.set_zlim(0, N * 2.5)
        surf1 = ax1.plot_surface(X, Y, z, cmap='gray')
        surf2 = ax2.plot_surface(X, Y, f, cmap='coolwarm', alpha = 0.8)
```

In [15]:

```
fig
```

Out[15]:



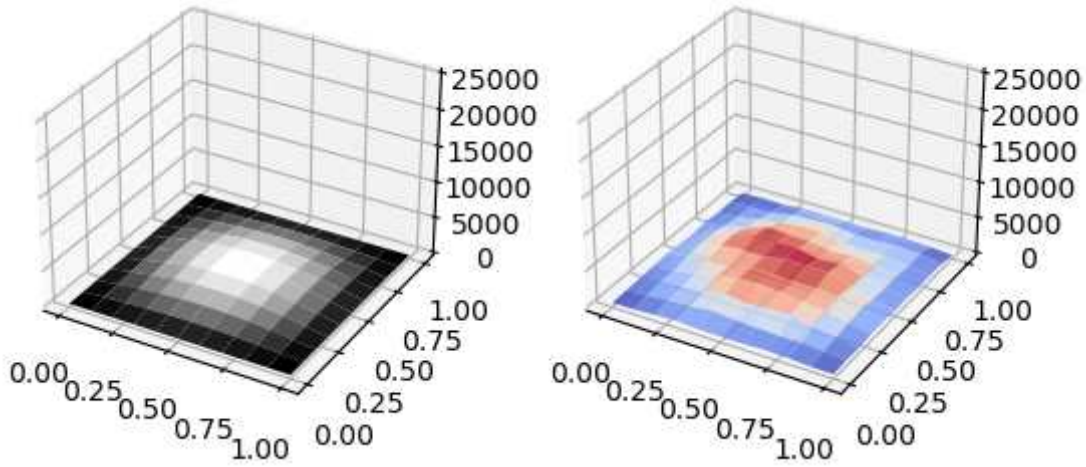
In [16]:

```
# Main loop
for s in range(20): # after 60 step
    f = 0.0 * f  # empty the bin for new constructon of the distr. every step
    for i in range(N):
        if 0 <= x[i] and x[i] < 1 and 0 <= y[i] and y[i] < 1:
            nx = x[i] * (1.0 * nBin)  # normalize the potision to bin
            m = int(nx)  # find the bin index
            nx -= m  # find the relative displacement in bin
            ny = y[i] * (1.0 * nBin)
            n = int(ny)
            ny -= n

            # 2 dimensional density distribution

            f[m, n] += (1.0 - nx) * (1.0 - ny)
            f[m + 1, n] += nx * (1.0 - ny)
            f[m, n + 1] += (1.0 - nx) * ny
            f[m + 1, n + 1] += nx * ny

            # 2 dimensional randomwalk
            r = random.choice([0,1])
            if r == 0 :
                x[i] = x[i] + random.choice([dx, -dx])
            else :
                y[i] = y[i] + random.choice([dx, -dx])

            # x[i] = x[i] + random.choice([dx, -dx])
            # y[i] = y[i] + random.choice([dx, -dx])  # random walk

    f = 1.0 * nBin * 1.0 * nBin * f  # normalization factor for distri.
    # plt.ylim(0,1.2*np.pi*N/2.0)
    z = (np.pi / 2) ** 2 * N * np.sin(np.pi * X) * np.sin(np.pi * Y) * np.exp(-(1.0 * (s+40)) /
    if s%10 == 0 :
        ax1.clear()
        ax2.clear()
        ax1.set_zlim(0, N * 2.5)
        ax2.set_zlim(0, N * 2.5)
        surf1 = ax1.plot_surface(X, Y, z, cmap='gray')
        surf2 = ax2.plot_surface(X, Y, f, cmap='coolwarm', alpha = 0.8)
```

In [17]:

```
fig
```

Out[17]:



In [18]:

```
# Main loop
for s in range(20): #after 80step
    f = 0.0 * f  # empty the bin for new constructon of the distr. every step
    for i in range(N):
        if 0 <= x[i] and x[i] < 1 and 0 <= y[i] and y[i] < 1:
            nx = x[i] * (1.0 * nBin)  # normalize the potision to bin
            m = int(nx)  # find the bin index
            nx -= m  # find the relative displacement in bin
            ny = y[i] * (1.0 * nBin)
            n = int(ny)
            ny -= n

            # 2 dimensional density distribution

            f[m, n] += (1.0 - nx) * (1.0 - ny)
            f[m + 1, n] += nx * (1.0 - ny)
            f[m, n + 1] += (1.0 - nx) * ny
            f[m + 1, n + 1] += nx * ny

            # 2 dimensional randomwalk
            r = random.choice([0,1])
            if r == 0 :
                x[i] = x[i] + random.choice([dx, -dx])
            else :
                y[i] = y[i] + random.choice([dx, -dx])

            # x[i] = x[i] + random.choice([dx, -dx])
            # y[i] = y[i] + random.choice([dx, -dx])  # random walk

    f = 1.0 * nBin * 1.0 * nBin * f  # normalization factor for distri.
    # plt.ylim(0,1.2*np.pi*N/2.0)
    z = (np.pi / 2) ** 2 * N * np.sin(np.pi * X) * np.sin(np.pi * Y) * np.exp(-(1.0 * (s+60)) /
    if s%10 == 0 :
        ax1.clear()
        ax2.clear()
        ax1.set_zlim(0, N * 2.5)
        ax2.set_zlim(0, N * 2.5)
        surf1 = ax1.plot_surface(X, Y, z, cmap='gray')
        surf2 = ax2.plot_surface(X, Y, f, cmap='coolwarm', alpha = 0.8)
```

In [19]:

```
fig
```

Out[19]: