

## Kesit Seviyeleri, Kenar Bazlı İmaj Gruplamak

Bir dijital imajı renklere, objelere göre belli parçalara bölmek (segmentation) için, matematiksel bir formül kullanmak iyi çözümlerden biridir. Bunu yapmanın bazı yolları var. Basitleştirerek bir örnek verelim: diyelim ki gruplama için elimizdeki formül bir yuvarlak formülü  $x^2 + y^2 - c = 0$ , ki  $c$  bir sabit. Bu formülü  $x$  ve  $y$  koordinatları üzerinde bastığımız zaman radius'u  $\sqrt{c}$  olan bir çember elde ederiz. Gruplama için bu çembere buyutup küçülttüğümüzü farzedelim, çember imaj üzerindeki istediğimiz bölüme en iyi uydugu anda gruplamayı başarılı olarak kabul ediyoruz.

Fakat problem burada: eğer imajda birden fazla grup var ise, o zaman birden fazla çember gerekecektir, bu sefer algoritmik olarak üstteki formülü ikinci, üçüncü kere yaratmamız, ve o formüllerin o gruplara uyumunu ayrı ayrı takip etmemiz gerekirdi. Ya da diyelim ki kademeli (iterative) bir uydurma işlemi takip ediyoruz, bu işlem sırasında belki iki çemberin birleşmesi gerekse, o zaman iki formülü silip, yerine yenisini oluşturmakla uğrasmak gerekli olacaktı. Bunlar hem matematiksel, hem kodlama açısından kulfet oluşturmaktadır.

Kesit Seviyeleri kavramını kullanarak bu işi daha basitleştirebiliriz. Diyelim ki bölme görevini yapan  $\phi$  adlı fonksiyonumuzu 2 boyutlu olmak yerine 3 boyutlu eksenle tanımladık, ve, 2 boyutta bölme yapma görevini onun bir kesitine verdik. Kesit derken, alttaki üç boyutlu fonksiyonu yatay olarak bir noktadan “kestiğimizi” farz ediyoruz, ve o kesit üzerinde geçen  $\phi$  değerlerine bakıyoruz.

Bakıcılarımızı, tanımlamamızı değiştirerek, bazı avantajlar elde etmeyi umuyoruz aslında. Altta iki tane  $\phi$  fonksiyonu ve onların altında kesitlerini görebiliriz.

Kesit Seviyeleri tekniğini kullanarak elde ettiğimiz avantaj nedir? Artık sadece **tek** bir  $\phi$  fonksiyonu kullanarak 2 boyutlu imajımız üzerinde birbirinden ayrı gruplamalar yaratabiliyoruz. Bu gruplar birbiri ile birleşebilir, ayrılabilir, bu artık bizi ilgilendirmiyor. Biz sadece 3. boyuttaki  $\phi$  fonksiyonunu değiştirmekle uğrascakız, imaj üzerindeki gruplamalar ise o fonksiyonun 2. boyuta yansımaları (projection) üzerinden kendiliğinden gerçekleşecekler.

Matematiksel olarak  $\phi$  fonksiyonunu nasıl temsil ederiz?  $\phi$  fonksiyonu  $x$ ,  $y$ , boyutlarını alıp bize bir üçüncü  $z$  boyutu dondurmeli, ayrıca bu fonksiy-

onu imaji parcalarina ayirma islemini gerceklestirmek icin kademeli olarak degistirmeyi planladigimiza gore, o zaman bir  $t$  degiskeni de gerekiyor. Yani  $\phi(x, y, t)$  fonksiyonu. Gruplama icin kullanılacak kesiti ise sifir kesiti olarak alalim, yani  $\phi(x, y, t) = 0$ . Dogal olarak

$$\frac{d}{dt}(\phi(x, y, t) = 0) = 0$$

Simdi  $x$ , ve  $y$  degiskenlerinin zaman gore degisimini formule bir sekilde dahil etmek lazim. Bunun icin sifir kesit seviyesi uzerinde bir parcacik hayal edilir, ve bu parcacigin gittigi yol  $x(t)$ , ve  $y(t)$  olarak tanimlanir. O zaman

$$\frac{d}{dt}(\phi(x(t), y(t), t) = 0) = 0$$

Tam diferansiyel formulunden hareketle:

$$d(\phi(x(t), y(t), t)) = \frac{\partial \phi}{\partial x} dx + \frac{\partial \phi}{\partial y} dy + \frac{\partial \phi}{\partial t} dt = 0$$

$$\frac{d(\phi(x(t), y(t), t))}{dt} = \frac{\partial \phi}{\partial x} \frac{dx}{dt} + \frac{\partial \phi}{\partial y} \frac{dy}{dt} + \frac{\partial \phi}{\partial t} = 0$$

$$\frac{d(\phi(x(t), y(t), t))}{dt} = \frac{\partial \phi}{\partial x} \frac{dx}{dt} + \frac{\partial \phi}{\partial y} \frac{dy}{dt} + \phi_t = 0 \quad (1) \quad (\text{eq1})$$

Temsilen daha kısa bir isaret kullanmak gerekirse,  $\nabla$  ile  $\phi$ 'nin gradyanini (gradient) alarak, elde edilecek vektorun nokta carpimini kullanabiliriz. O zaman formül I daha kısa olarak:

$$\phi_t + \nabla \phi \cdot \vec{V} = 0$$

olarak temsil edilebilir, ki

$$\nabla \phi = \left( \frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y} \right)$$

$$\vec{V} = \left( \frac{dx}{dt}, \frac{dy}{dt} \right)$$

iki vektorun nokta carpimi bilindigi gibi sirayla her iki vektorun sirasiyla uyan elemanlarinin birbirleri ile carpilmasi ve o carpimlarin toplanmasidir.

$\vec{V}$  vektörü neyi temsil eder? Formule göre bu vektor  $\phi$ 'nin uzerindeki degisimi etkiliyor, ve bu degisimler  $t$ 'nin degisimine göre tanimlandigina göre bu degerler "hiz" olarak tanimlanabilir. Imaj baglaminda dusunursek mesela  $\phi$  renklerin

ayni oldugu yerlerde yuksek hizda, renklerin degistigi yerler dusuk hizda degisebilir seklinde bir kurgu yapılabilir, iste bu bolgelerde degisiminin hizini  $\vec{V}$  ile gosterebiliriz.

$\vec{V}$  yerine kesit seviyelerine dik olan (normal) vektorler ile calismak isteseydik,  $\vec{V}$ 'yi dik ve teget bilesenlerine ayirarak tekrar temsil edebilirdik:  $\vec{V} = V_N \vec{N} + V_T \vec{T}$ . Bu formulde  $\vec{T}$  teget,  $\vec{N}$  dik vektorler,  $N$  ve  $T$  skalar. Yerine koyalım:

$$\phi_t + \nabla \phi \cdot (V_N \vec{N} + V_T \vec{T}) = 0$$

$\phi$ 'ye gore dik vektorun diger bir formulu  $\vec{N} = \frac{\nabla \phi}{|\nabla \phi|}$  olduguna gore

$$\phi_t + (\nabla \phi \cdot V_N \frac{\nabla \phi}{|\nabla \phi|} + \nabla \phi \cdot V_T \vec{T}) = 0$$

Devam edelim:  $\nabla \phi$  yuzeye dik olduguna gore, bu dik vektorun teget olan  $\vec{T}$  ile noktasal carpimi sifir degerini verecektir, o carpim formulden atilabilir. Kalanlar:

$$\phi_t + (\nabla \phi \cdot V_N \frac{\nabla \phi}{|\nabla \phi|}) = 0$$

Daha da kisaltabiliriz:  $\nabla \phi \cdot \nabla \phi = |\nabla \phi|^2$  oldugunu biliyoruz, gradyanin kendisi ile noktasal carpimi, o gradyan vektorunun uzunlugunun karesidir. Daha genel olarak, bir vektorun uzunlugu, o vektorun kendisi ile noktasal carpiminin karekokudur. Ayni sey. O zaman en son formulde bu carpimi gerceklestirip, uzunluk olarak yazalım:

$$\phi_t + V_N \frac{|\nabla \phi|^2}{|\nabla \phi|} = 0$$

$$\phi_t + V_N |\nabla \phi| = 0$$

Simdi bu formul hakkında biraz anlayis gelistirelim. Eger elimizdeki bir  $\phi$  seviye kesitinin seklen oldugu gibi kalmasini ama sadece kuculmesini isteseydik,  $\phi$ 'nin normalinin tersi yonunde bir buyume tanimlamamız gerekirdi. Normal vektor disa dogru isaret ettigine gore ustteki formulde mesela  $V_N = -1$  tanimlayabilirdik. O zaman

$$\phi_t + -1 |\nabla \phi| = 0$$

$$\phi_t = |\nabla \phi|$$

Hesapsal olarak bunu nasil gerceklestiririz? 80 x 80 boyutunda bir matris

icinde  $\phi$  fonksiyonu ayriksal olarak tutalım. Yani 80 tane x, 80 tane ayrı y değeri var, her x ve y değerlerin kombinasyonlarına tekabül eden  $\phi$  değerleri bu matris icinde. Gradyanın ne olduğunu hatırlayalım. Gradyan

$$\nabla\phi = \left(\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y}\right)$$

olarak tanımlıdır, ve her  $(x_i, y_i)$  noktasındaki  $\phi(x_i, y_i)$  değerine göre değişik bir vektor sonucunu getirecektir. Bilgisayar dunyasında parcalı turevler hesap-sal “farklılıklara” donusurlar, **phi** matrisindeki farklılıkları Python ile

`gradPhiY, gradPhiX = np.gradient(phi)`

olarak hesaplayabiliriz. Ustte elimize gecen gradyan dizinlerindeki değerler ile  $|\nabla\phi|$  buyuklugunu hesaplayabiliriz, ve bu sonucu  $\phi$  uzerindeki degisim oranı  $\phi_t$  olarak kabul ederiz. O zaman  $\phi_t$  ile zaman  $t$  degimi **dt** carptigimiz zaman ele gecek olan  $\phi$ ’nin degisimidir. Dongunun her basamagında eski **phi** değerlerine bu farkları ekledigimiz zaman  $\phi$  fonksiyonu istedigimiz gibi evrilecektir.

Altteki kodda bizim baslangic  $\phi$ ’miz kenarlardan  $w$  uzakligında ici bos bir kutu olacak. Sifir seviyesindeki kesit seviyesinin nasıl iki boyutlu goruntudeki kirmizi cizgilere tekabül ettigini gorebiliriz.

Listing 1: active1.py

---

```
import matplotlib.pyplot as plt
import numpy as np
import plot_phi
import time

# initial function phi - level set is a square 4 pixels
# away from borders on each side, in 3D it looks like an empty
# box
c0=4; w=4
nrow, ncol= (80,80)
phi=c0*np.ones((nrow,ncol))
phi[w+1:-w-1, w+1:-w-1]=-c0
plot_phi.plot_phi(phi)

dt=1.
```

```

iter=0

plt.ion()

while iter < 20:
    # gradient of phi
    gradPhiY, gradPhiX = np.gradient(phi)
    # magnitude of gradient of phi
    absGradPhi=np.sqrt(gradPhiX**2+gradPhiY**2)
    dPhiBydT = 1 * absGradPhi
    # level set evolution equation
    phi = phi + ( dt * dPhiBydT )
    iter=iter+1
    time.sleep(0.6)
    plt.hold(False)
    CS = plt.contour(phi,0, colors='r')
    plt.draw()
    plt.hold(False)
    iter += 1

```

---

Ustteki kod isleyince sifir kesit seviyesinin (kirmizi cizgiler) olduklari gibi kuculduklerini gorecegiz.

### Ortalama Egim (Mean Curvature) Kullanmak

Eger sabit hiz yerine sifir kesit seviyesinin herhangi bir noktada ne kadar “egri” olduguna gore ilerlemesini isletseydik ne olurdu? Diyelim ki cok egri bolgelerde cok hizli, az egik (duz, duze yakin) bolgelerde ilerleme az hiz istiyoruz. O zaman hangi sekille baslarsa baslasindalar  $\phi$  kesiti sonucta bir cember sekline dogru evrilecektir. Ortalama egim (mean curvature) hesabi icin su denklem kullanilir:

$$\kappa = -div\left(\frac{\nabla\phi}{|\nabla\phi|}\right)$$

Bu formulun turetilmesini burada yapmayacagiz. Python kodu soyle:

---

#### Listing 2: active2.py

---

```

import matplotlib.pyplot as plt
import numpy as np

```

```

import time

# initial function phi - level set is a square 4 pixels
# away from borders on each side, in 3D it looks like an empty
# box
c0=2; w=2
nrow, ncol= (30,30)
phi=c0*np.ones((nrow,ncol))
phi[w+1:-w-1, w+1:-w-1]=-c0

dt=1.

phiOld=np.zeros((nrow,ncol))

iter=0

plt.ion()

while iter < 50:
    # gradient of phi
    gradPhiY, gradPhiX = np.gradient(phi)
    # magnitude of gradient of phi
    absGradPhi=np.sqrt(gradPhiX**2+gradPhiY**2)

    # normalized gradient of phi - eliminating singularities
    normGradPhiX=gradPhiX/(absGradPhi+(absGradPhi==0))
    normGradPhiY=gradPhiY/(absGradPhi+(absGradPhi==0))

    divYnormGradPhiX, divXnormGradPhiX=np.gradient(normGradPhiX)
    divYnormGradPhiY, divXnormGradPhiY=np.gradient(normGradPhiY)

    # curvature is the divergence of normalized gradient of phi
    K = divXnormGradPhiX + divYnormGradPhiY
    dPhiBydT = K * absGradPhi # makes everything circle

    # level set evolution equation
    phi = phi + ( dt * dPhiBydT )
    iter=iter+1

```

```
time.sleep(0.6)
plt.hold(False)
CS = plt.contour(phi,0, colors='r')
plt.draw()
plt.hold(False)
iter += 1
```

---

## Imaj Gruplamak

Imaji bolumlere ayirmak icin (segmentation) birkac faktorun bilesimi kullaniliyor. Koseleri kullanan aktif kontr (edge based active contour) yonde-minde ortalama egim ve imajin piksel degerlerinin farkliliklari (image gradient) ayni anda kullanilir. Yani kesit seviyesini ilerletirken hizi hem egime oranliyoruz, hem de imaj piksel renk degerleri arasindaki farka ters oranda hizlandiriyor, ya da yavaslatiyoruz. Boylece kesit seviyemiz renk farkliliği çok olmayan yani büyük bir ihtimalle tek bir objeye ait bir bolgede hizla ilerliyor, büyük renk farkinin olduğu büyük bir ihtimalle bir kenar noktasina gelince ise yavasliyor. O sirada kesit seviyesinin geri kalan taraflari tabii ki baska hizlarda hareket ediyor olabilirler, zaten isin puf noktası burada, sonunda resim bolgelere ayrilmis oluyor. Bu kodu da **active3.py** icinde bulabilir, **active4.py** icinde ise sürekli degisim sonrasi sayisal bazi yan etkilerden dolayı  $\phi$ 'nin dejenere olması sonucu onu “tekrar bastan olusturan (reinitialization)” iceren bir kisim var. Fakat teknigin ozu her iki kod icinde de gorulebilir.

Bitirirken önemli gözlemi vurgulayalım. Problemi matematiksel olarak temsil ederken, hedefe doğru turetirken sürekli (continous) alemde, sürekli, kesintisiz fonksiyonlarla iş yapıyoruz. Hesaplama anı gelince sürekli fonksiyonları ayriksal (discrete) hale çeviriyoruz, iste uygulamalı matematigin hesapsal kısmı burada devreye giriyor. Fakat diferansiyel denklemler, fonksiyonlar, turevler gibi sürekli matematigin kavramları çok önemli, bunlar olmasa problemi soyut bir şekilde temsil edemez, ve basitlestiremezdik. Temel matematigin kavramlarını kullanırken yuzyılların matematiksel bilgisi devreye girebiliyor, matematigin en yoğun şekilde kullanıldığı fizikten bol bol teknik alınabilir. Yani söylemek istedigimiz problemi cozmek icin hemen kodlamaya baslamıyoruz, dusunsel eylemin önemli bir kısmı matematiksel formüllerle (belki kalem kagitla) yapiliyor.

---

Listing 3: active3.py

---

```

import matplotlib.pyplot as plt
import numpy as np
import scipy.signal as signal
import scipy.ndimage as image
import time

def gauss_kern():
    """ Returns a normalized 2D gauss kernel array for convolutions """
    h1 = 15
    h2 = 15
    x, y = np.mgrid[0:h2, 0:h1]
    x = x-h2/2
    y = y-h1/2
    sigma = 1.5
    g = np.exp( -( x**2 + y**2 ) / (2*sigma**2) );
    return g / g.sum()

Img = plt.imread("twoObj.bmp")
Img = Img[:, :-1]
g = gauss_kern()
Img_smooth = signal.convolve(Img, g, mode='same')
Iy, Ix = np.gradient(Img_smooth)
absGradI = np.sqrt(Ix**2 + Iy**2);
rows, cols = Img.shape

# initial function phi - level set is a square 4 pixels
# away from borders on each side, in 3D it looks like an empty
# box
c0 = 4
w = 4
nrow, ncol = Img.shape
phi = c0 * np.ones((nrow, ncol))
phi[w+1:-w-1, w+1:-w-1] = -c0

# edge-stopping function
g = 1 / (1 + absGradI**2)

# gradient of edge-stopping function

```



```

gy,gx = np.gradient(g)

# gradient descent step size
#dt=.4
dt=1.

# number of iterations after which we reinitialize the surface
num_reinit=10

phiOld=np.zeros((rows,cols))

# number of iterations after which we reinitialize the surface
iter=0

plt.ion()

while True:
    # gradient of phi
    gradPhiY, gradPhiX = np.gradient(phi)
    # magnitude of gradient of phi
    absGradPhi=np.sqrt(gradPhiX**2+gradPhiY**2)
    # normalized gradient of phi - eliminating singularities
    normGradPhiX=gradPhiX/(absGradPhi+(absGradPhi==0))
    normGradPhiY=gradPhiY/(absGradPhi+(absGradPhi==0))

    divYnormGradPhiX, divXnormGradPhiX=np.gradient(normGradPhiX)
    divYnormGradPhiY, divXnormGradPhiY=np.gradient(normGradPhiY)

    # curvature is the divergence of normalized gradient of phi
    K = divXnormGradPhiX + divYnormGradPhiY
    tmp1 = g * K * absGradPhi
    tmp2 = g * absGradPhi
    tmp3 = gx * gradPhiX + gy*gradPhiY
    dPhiBydT =tmp1 + tmp2 + tmp3

    phiOld=phi
    # level set evolution equation
    phi = phi + ( dt * dPhiBydT )

```

```

iter=iter+1
if np.mod(iter,10)==0:
    time.sleep(0.6)
    plt.imshow(Img, cmap='gray')
    plt.hold(True)
    CS = plt.contour(phi,0, colors='r')
    plt.draw()
    plt.hold(False)

```

---

Listing 4: active4.py

---

```

import matplotlib.pyplot as plt
import numpy as np
import scipy.signal as signal
import scipy.ndimage as image
import time
from scipy import ndimage

def bwdist(a):
    """
    this is an intermediary function, 'a' has only True, False vals,
    so we convert them into 0, 1 values — in reverse. True is 0,
    False is 1, distance_transform_edt wants it that way.
    """
    b = np.ones(a.shape)
    b[a==True] = 0.
    return ndimage.distance_transform_edt(b)

def gauss_kern():
    """ Returns a normalized 2D gauss kernel array for convolutions """
    h1 = 15
    h2 = 15
    x, y = np.mgrid[0:h2, 0:h1]
    x = x-h2/2
    y = y-h1/2
    sigma = 1.5
    g = np.exp( -( x**2 + y**2 ) / (2*sigma**2) );
    return g / g.sum()

```

```

Img = plt.imread("twoObj.bmp")
Img = Img[:, :-1]
g = gauss_kern()
Img_smooth = signal.convolve(Img, g, mode='same')
Iy, Ix = np.gradient(Img_smooth)
absGradI = np.sqrt(Ix**2 + Iy**2);
rows, cols = Img.shape

# initial function phi - level set is a square 4 pixels
# away from borders on each side, in 3D it looks like an empty
# box
c0 = 4
w = 4
nrow, ncol = Img.shape
phi = c0 * np.ones((nrow, ncol))
phi[w+1:-w-1, w+1:-w-1] = -c0

# edge-stopping function
g = 1 / (1 + absGradI**2)

# gradient of edge-stopping function
gy, gx = np.gradient(g)

# gradient descent step size
#dt = .4
dt = 1.

# number of iterations after which we reinitialize the surface
num_reinit = 10

phiOld = np.zeros((rows, cols))

# number of iterations after which we reinitialize the surface
iter = 0

plt.ion()

```

```

#while np.sum(np.sum(np.abs(phi-phiOld))) != 0:
while True:
    # gradient of phi
    gradPhiY, gradPhiX = np.gradient(phi)
    # magnitude of gradient of phi
    absGradPhi=np.sqrt(gradPhiX**2+gradPhiY**2)
    # normalized gradient of phi - eliminating singularities
    normGradPhiX=gradPhiX/(absGradPhi+(absGradPhi==0))
    normGradPhiY=gradPhiY/(absGradPhi+(absGradPhi==0))

    divYnormGradPhiX, divXnormGradPhiX=np.gradient(normGradPhiX)
    divYnormGradPhiY, divXnormGradPhiY=np.gradient(normGradPhiY)

    # curvature is the divergence of normalized gradient of phi
    K = divXnormGradPhiX + divYnormGradPhiY
    tmp1 = g * K * absGradPhi
    tmp2 = g * absGradPhi
    tmp3 = gx * gradPhiX + gy*gradPhiY
    dPhiBydT =tmp1 + tmp2 + tmp3
    #dPhiBydT = K * absGradPhi

    phiOld=phi
    # level set evolution equation
    phi = phi + ( dt * dPhiBydT )
    iter=iter+1
    if np.mod(iter,num_reinit)==0:
        # reinitialize the embedding function
        # after num_reinit iterations
        phi=np.sign(phi)
        phi = (phi > 0) * (bwdist(phi < 0)) - \
            (phi < 0) * (bwdist(phi > 0))

    if np.mod(iter,10)==0:
        time.sleep(0.6)
        plt.imshow(Img, cmap='gray')
        plt.hold(True)
        CS = plt.contour(phi,0, colors='r')
        plt.draw()

```

```
plt. hold( False)
```

---

Listing 5: plot\_phi.py

---

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

def plot_phi(phi):
    fig = plt.figure()
    ax = Axes3D( fig )
    x = []
    y = []
    for (i,j),val in np.ndenumerate(phi):
        x.append(i)
        y.append(j)
    ax.plot( xs=x, ys=y, zs=phi.flatten(),
            zdir='z', label='ys=0, zdir=z' )
    plt.show()
```

---

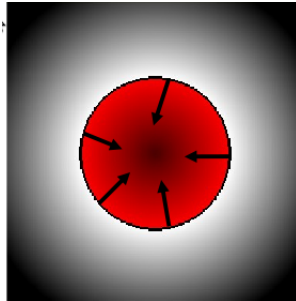


Figure 1:

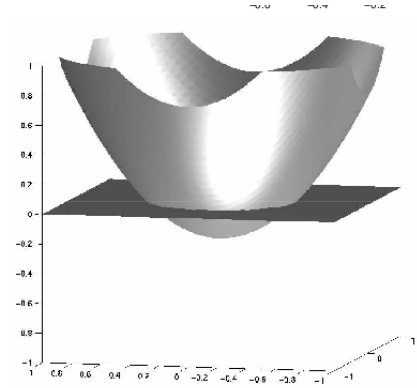


Figure 2:  $\phi$  Fonksiyonu

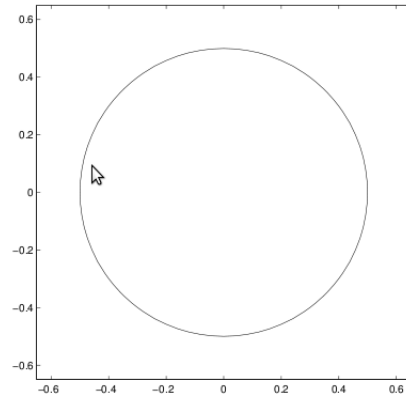


Figure 3: Kesit Seviyesi

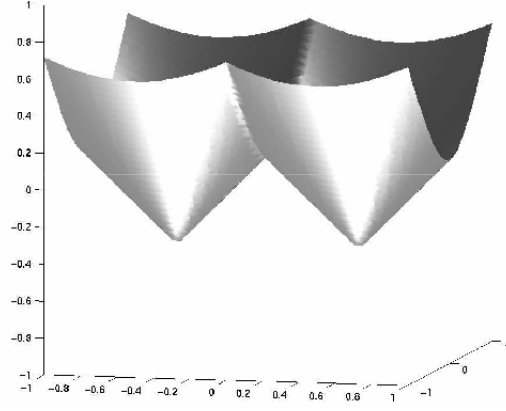


Figure 4:  $\phi$  Fonksiyonu

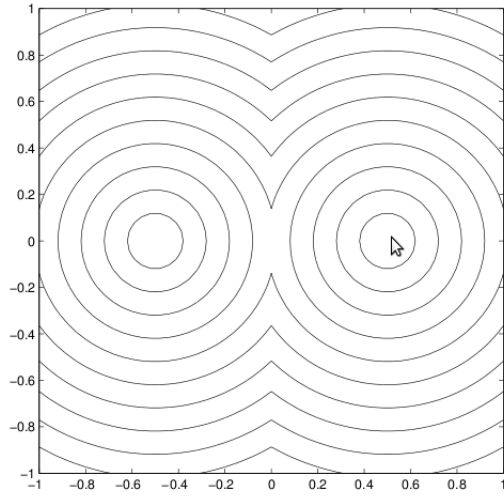


Figure 5: Birkac z Seviyesinden Kesitler

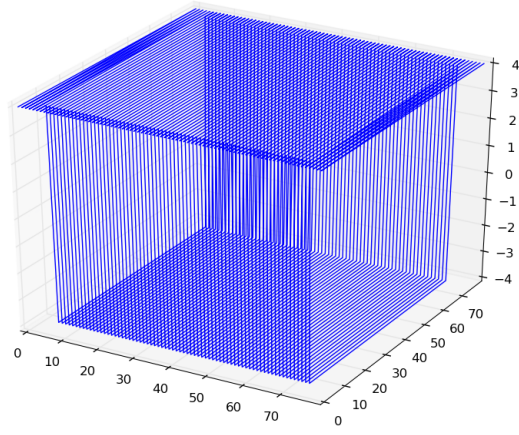


Figure 6:  $\phi$  Baslangici

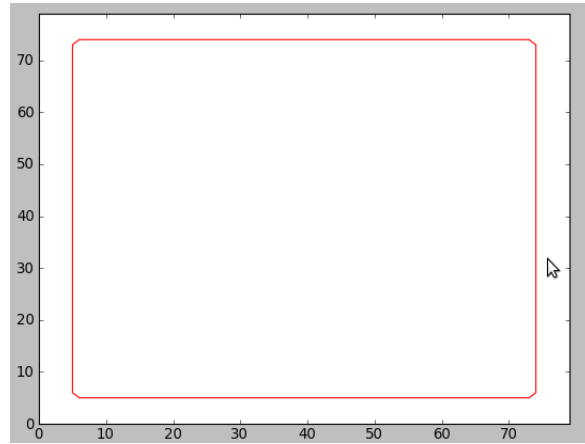


Figure 7:  $\phi$  Baslangici 2 Boyutta



