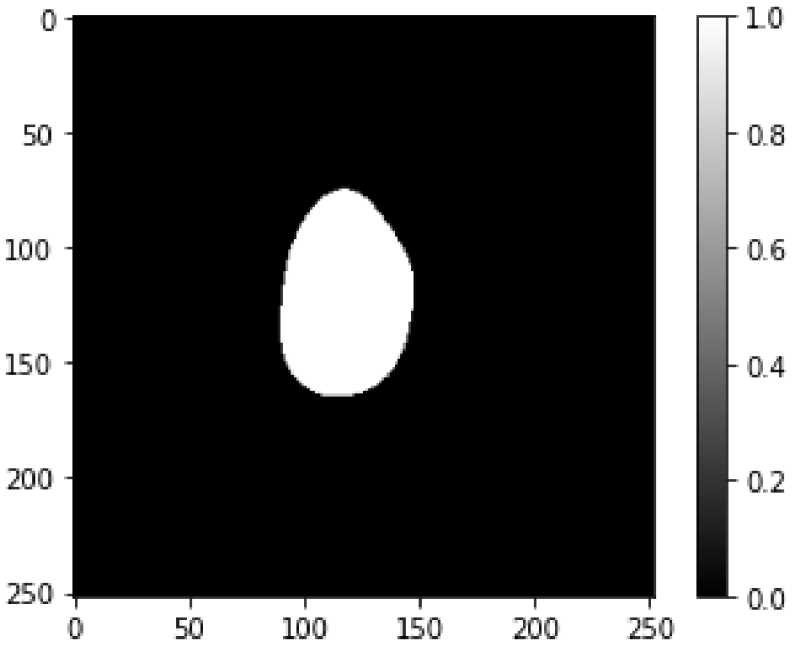
geodesic_equilibrium

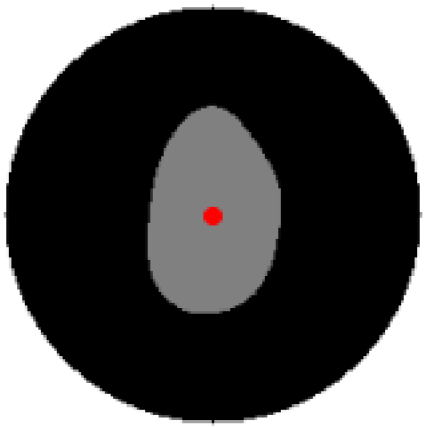
November 24, 2021

1 Geodesics in heat: time-dependent and steady state solutions

In [276]:
```
# Author: Karim Makki

import numpy as np
from matplotlib import pyplot as plt
from scipy import ndimage
from scipy.ndimage.filters import gaussian_filter
import cv2
from PIL import Image, ImageOps


mask = Image.open("./mask.png")
mask = ImageOps.grayscale(mask)/np.max(mask) #convert to grayscale

plt.imshow(mask, cmap='gray') #input mask
plt.colorbar()
plt.show()
```



In [277]:
```
##determine shape centroid and surrounding sphere

y,x = np.where(mask!=0)
cx = int(np.median(x))
cy = int(np.median(y))

radius = 90

nx,ny = mask.shape
mask2 = np.zeros(shape=[nx, ny], dtype=np.uint8)
cv2.circle(mask2, center=(cx,cy), radius=radius , color = 1, thickness=-1)

plt.imshow(mask+2*(1-mask2), cmap=plt.cm.gray)
plt.scatter(cx,cy,color='red')
plt.axis('off')
plt.show()
```



2 Heat flow integration

In the following, we propose to solve the heat equation $\frac{\partial h}{\partial t} - \alpha \Delta h = 0$ using two different numerical schemes.

- We can estimate heat diffusion (where dissipation time approaches zero, i.e. do the same as the heat method do), following the finite difference method de-

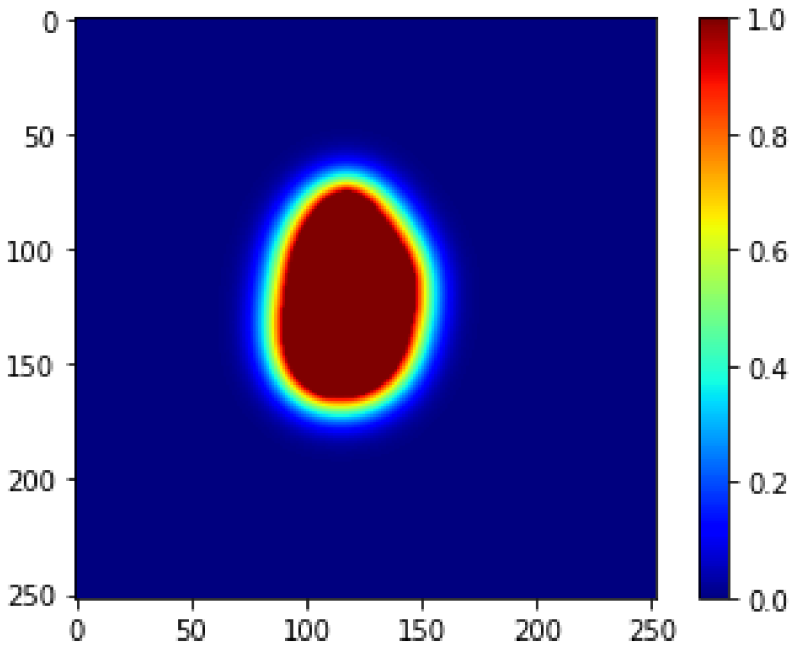
scribed in: https://levelup.gitconnected.com/solving-2d-heat-equation-numerically-using-python-3334004aa01a .

- We estimate heat diffusion at thermal equilibrium as presented in our manuscript.

In [278]:
```
## Heat flow integration (using our method)

in_boundary = mask
out_boundary = 1 - mask2
S_prime = np.where(out_boundary!=0)

R = np.where(out_boundary + in_boundary == 0) # This corresponds to the region Omega

## Dirichlet boundary conditions

h = np.zeros(mask.shape)
h[mask!=0] = 1


def heat_flow_integration(h, R, n_iters, method="heat_equilibrium"):

    h_n = np.zeros(h.shape)

    ht = []



    if method == 'heat':

        print("heat method")

        alpha = 1
        delta_x = 1

        delta_t = (delta_x ** 2)/(4 * alpha)
        gamma = (alpha * delta_t) / (delta_x ** 2)



        for it in range (n_iters):

            h_n = h

            h[R[0], R[1]] = gamma * (h_n[R[0]+1, R[1]] + h_n[R[0]-1,\
            R[1]] + h_n[R[0],R[1]+1] + h_n[R[0], R[1]-1]- 4*h_n[R[0], R[1]])\
            + h_n[R[0], R[1]]

            ht.append(h.copy())
```

```
            else:

                print("thermal equilibrium method")


                for it in range (n_iters):

                    h_n = h


                    h[R[0],R[1]]= (h_n[R[0]+1,R[1]] + h_n[R[0]-1,R[1]] +  \
                    h_n[R[0],R[1]+1] + h_n[R[0],R[1]-1]) / 4

                    ht.append(h.copy())



                return h, ht


        h, ht = heat_flow_integration(h, R, 201, method="heat_equilibrium")

        #fig = plt.figure(figsize=(10, 10))
        plt.imshow(h, cmap='jet')
        plt.clim(0, 1)
        plt.colorbar()

        plt.show()
```

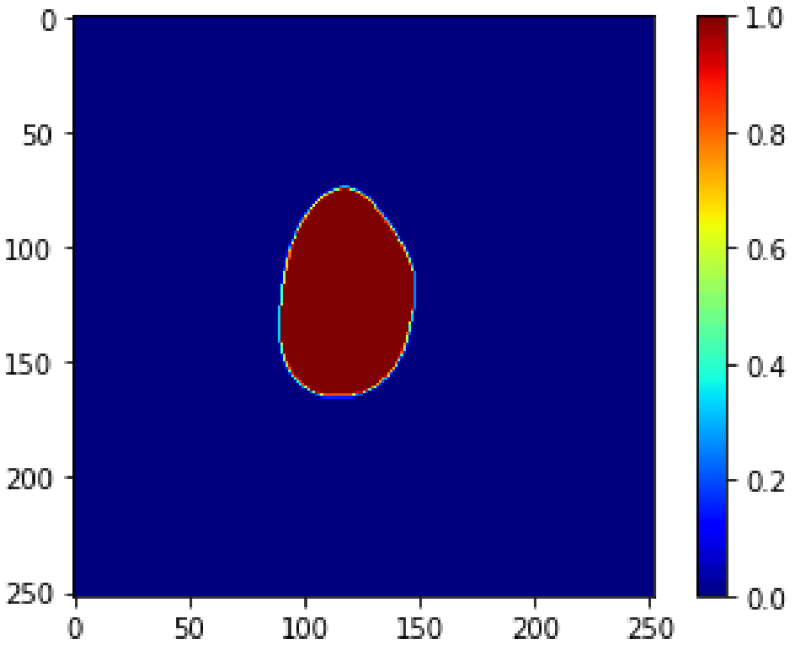
```
thermal equilibrium method
```



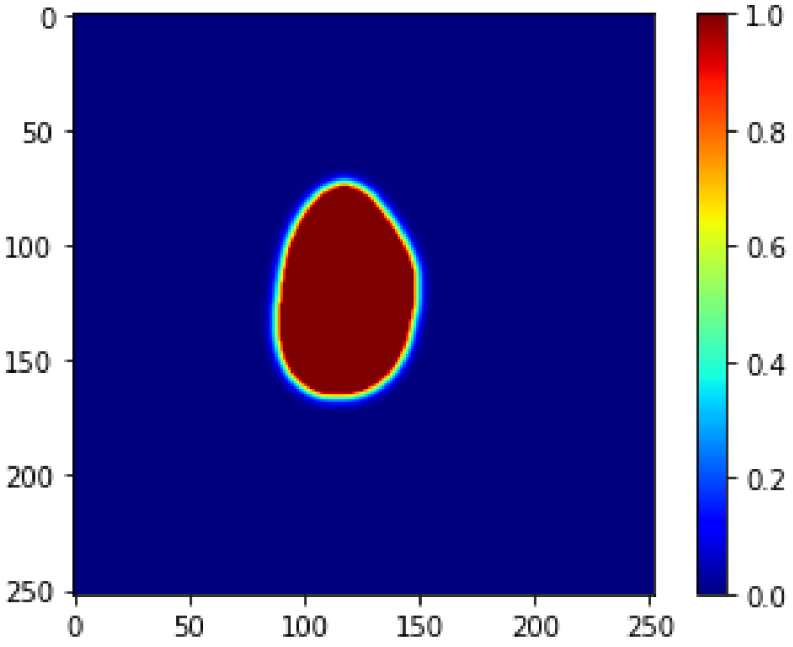
3 Display intermediate heat states

In [279]:
```
n_iters = 200

step = 20

for t in np.arange(0, n_iters+step, step):
    print("Time", t)
    plt.imshow(ht[t], cmap='jet')
    plt.colorbar()
    plt.show()
```

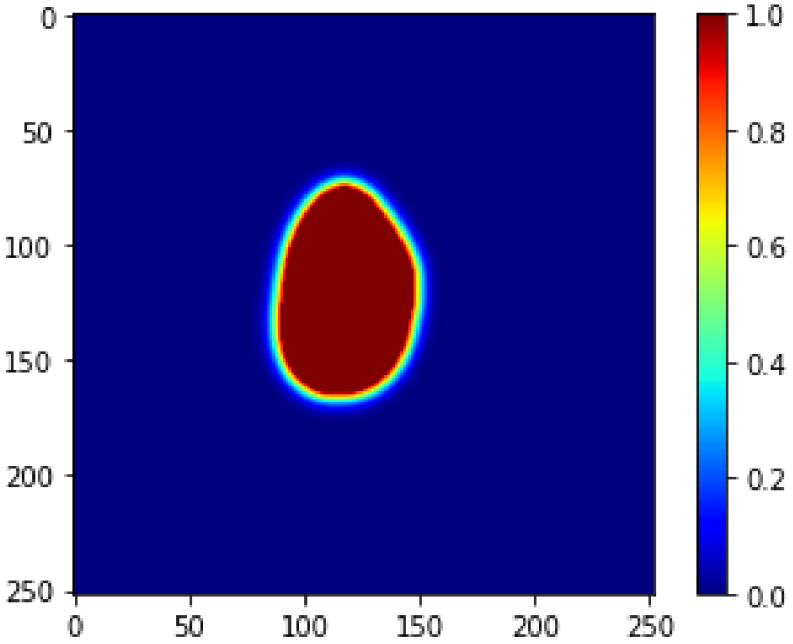
```
Time 0
```


0
50
100
150
200
250
0
50
100
150
200
250
1.0
0.8
0.6
0.4
0.2
0.0

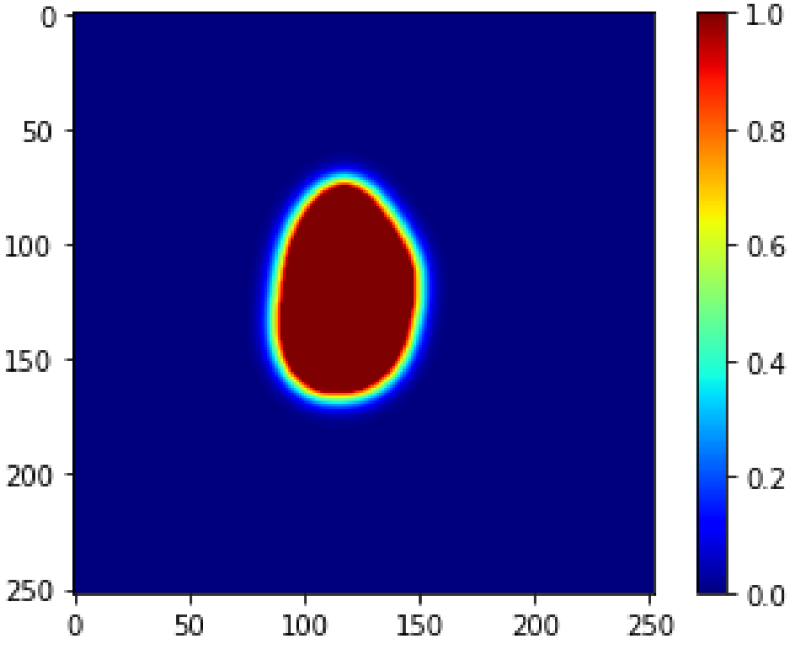
```
Time 20
```


0
50
100
150
200
250
0
50
100
150
200
250
1.0
0.8
0.6
0.4
0.2
0.0

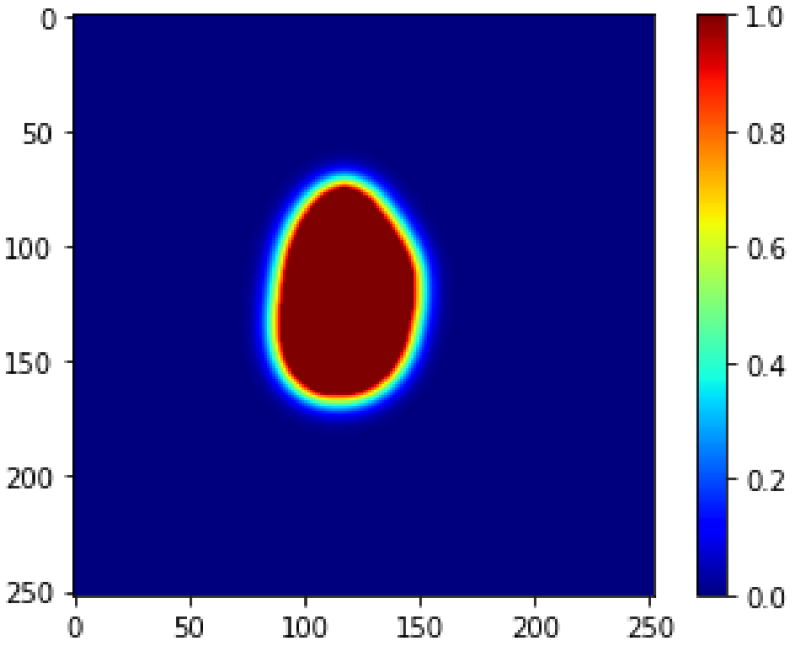
Time 40



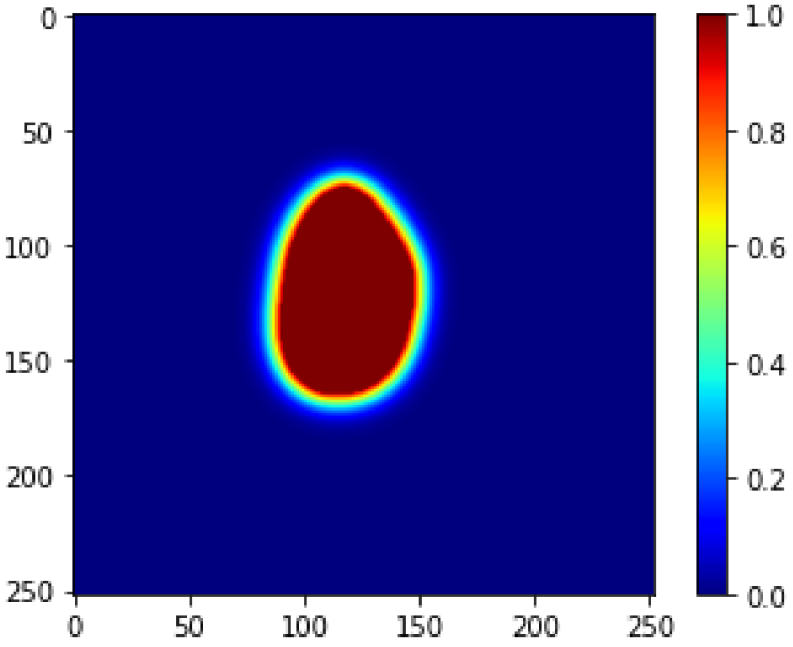
Time 60



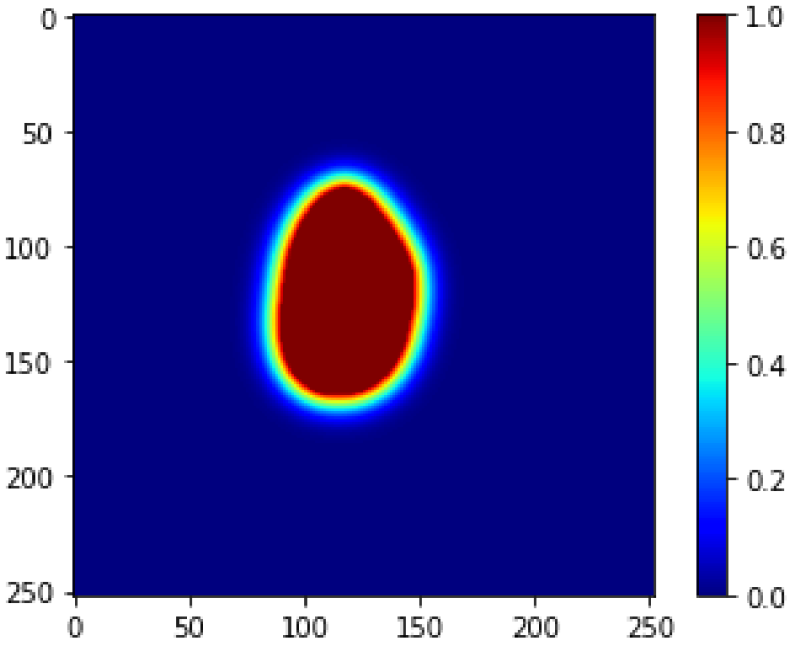
Time 80



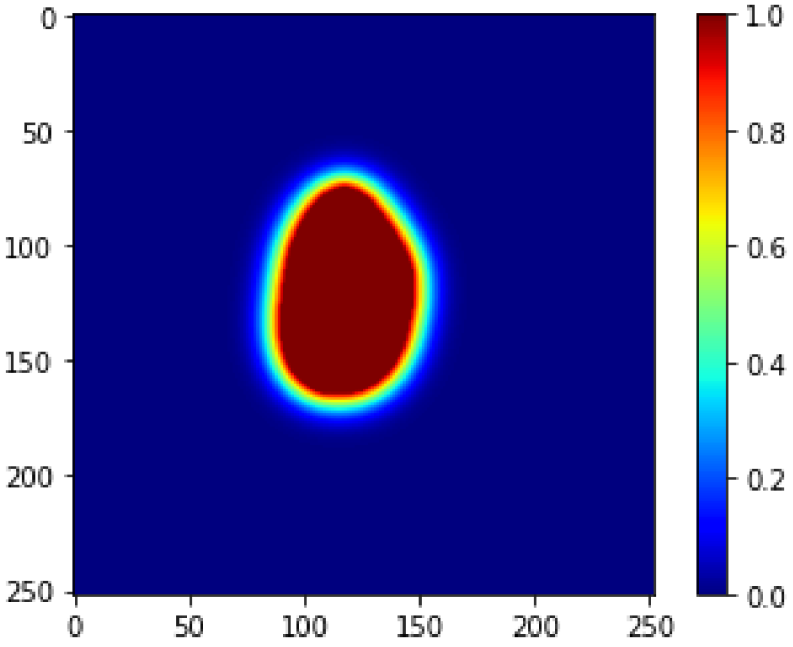
Time 100



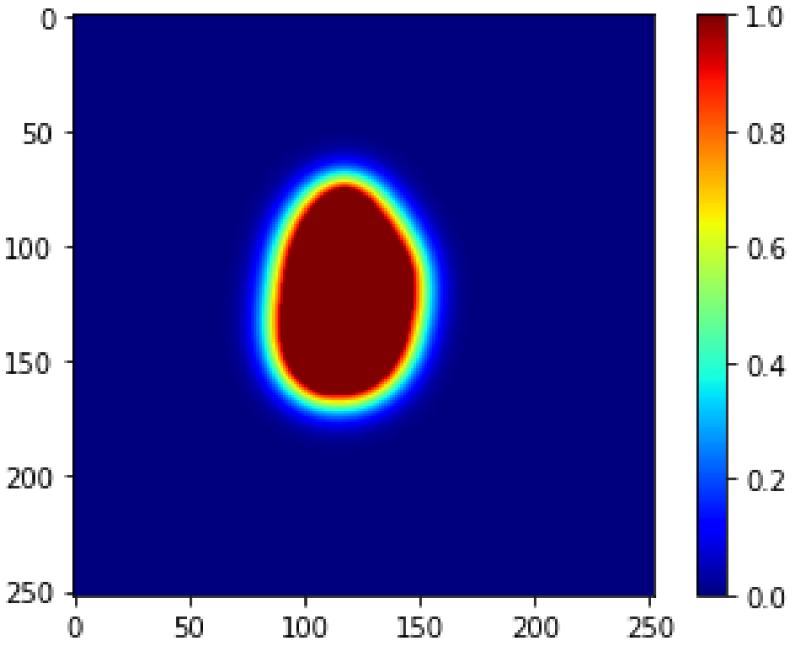
Time 120



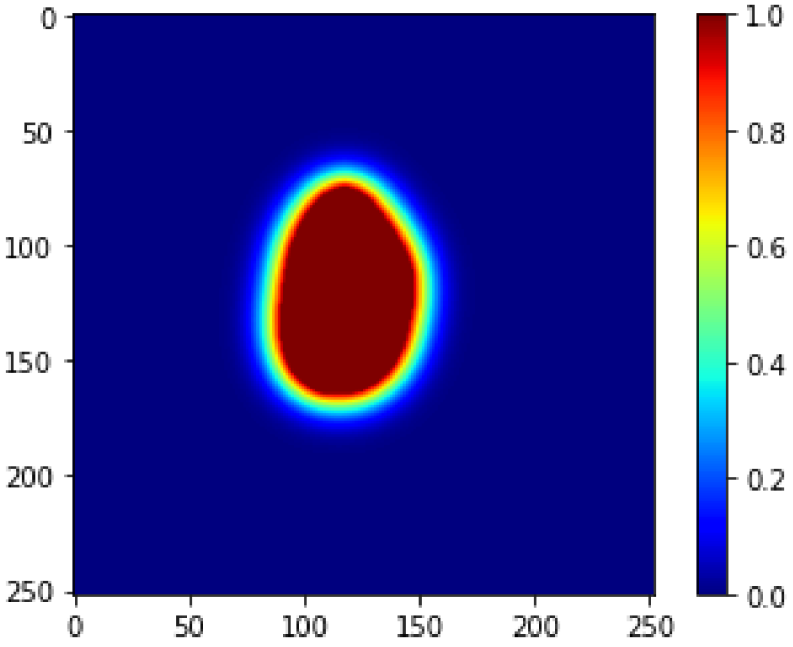
Time 140



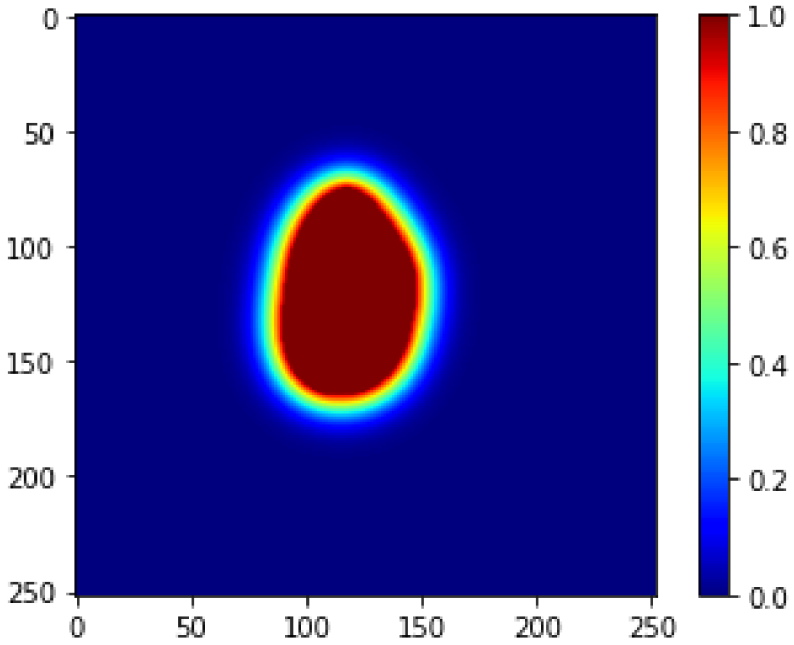
Time 160



Time 180



Time 200



4 N is a conservative vector field (proof)

To prove that the computed correspondence trajectories are geodesics, it is mandatory to prove that N is conservative. For this, it is sufficient to prove that N is curl free (irrotational) in the vicinity of each heat isosurface. But, this is non-trivial, since $||\nabla h|| \neq cste$ generally. In fact we have: $curl(N) = \nabla \times N = \nabla \times (\frac{\nabla h}{||\nabla h||}) = \nabla(\frac{1}{||\nabla h||}) \times \nabla h + \frac{1}{||\nabla h||} \underbrace{\nabla \times \nabla h}_{=curl(\nabla h)=0.} = \frac{1}{||\nabla h||^2} \nabla(||\nabla h||) \times \nabla h$. And since $||\nabla h|| \neq cste$ generally, then the problem arises. Overall Ω, for each isosurface $h = h_{iso}$, the fact that $curl(N) = 0$ depends essentially on how the vector field N is defined away from that isosurface. However, on each isosurface, N is well defined and conservative~[**?**].

- Formulation 1: Suppose N to be the normalized gradient of h over Ω. In this case, $\nabla \times N = 0$, when evaluated on the isosurface h_{iso} if and only if $||\nabla h|| = cste$.
- Formulation 2: Define the function $f = \frac{1}{||\nabla h||}(h - h_{iso})$ inside Ω. Suppose now that the surface we are interested in is $\Gamma = \{f = 0\}$. Developing the expression of the gradient of f, we obtain:

$$\nabla f = \frac{\nabla h}{||\nabla h||} - \frac{(h - h_{iso})\nabla h \cdot \nabla^2 h}{||\nabla h||^3}, \tag{1}$$

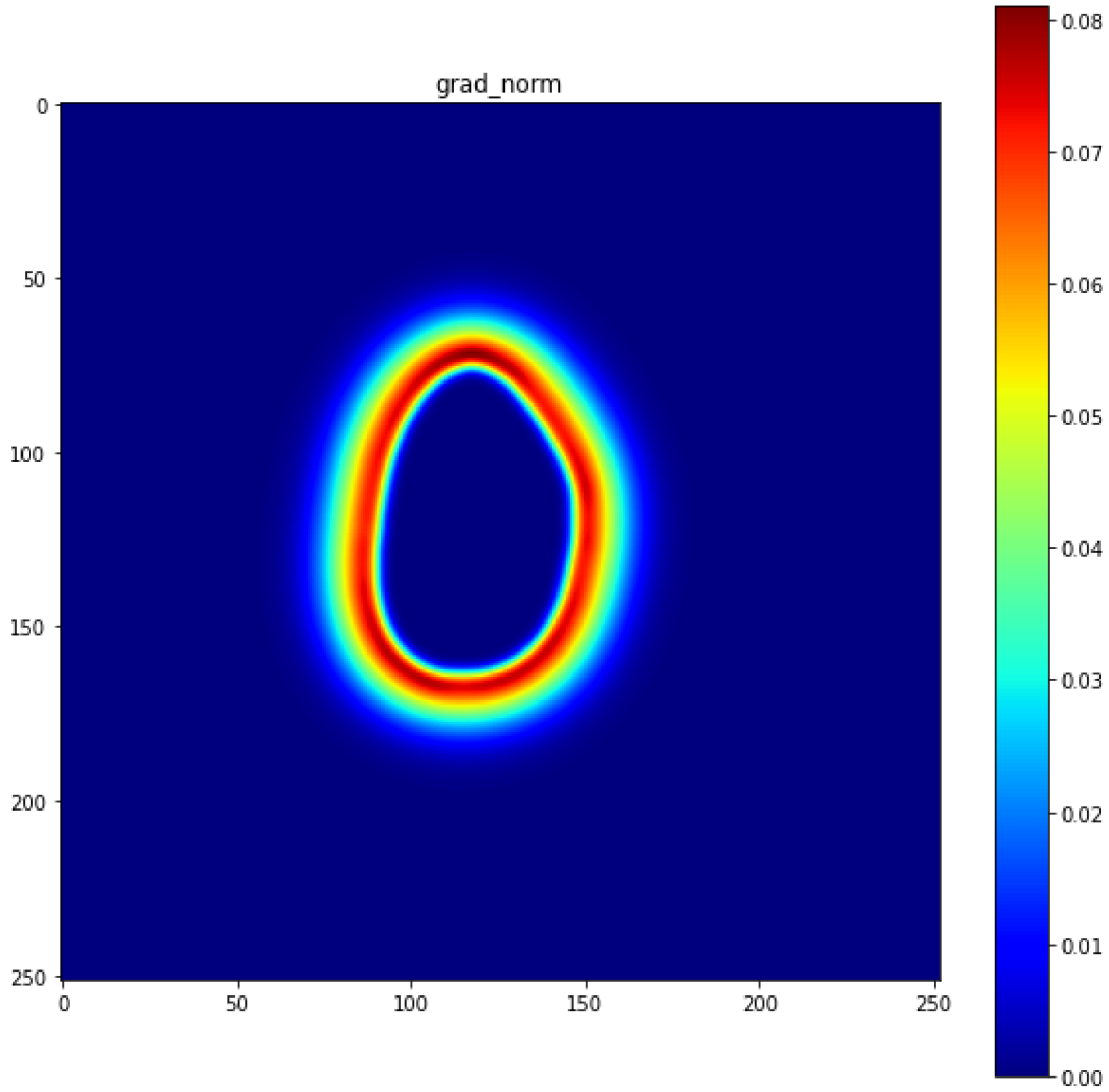
where $\nabla^2(.)$ is the Hessian operator. It is clear that the second term of the right-hand side of the above equation vanishes on Γ (since $h = h_{iso}$). Consequently, ∇f restricted to the surface is still the unit normal vector field N. Moreover, $\nabla \times \nabla f$ is clearly zero as argued before (by applying the linearity property to the curl operator).

More generally, for a compact smooth surface $\Gamma \subset \mathbb{R}^n$, there exists a real $r > 0$ such that on the set $\omega = \{x \in \mathbb{R}^3 \quad | \quad dist(x, \Gamma) < r\}$ one can solve the Eikonal PDE $|\nabla f| = 1$ to get a continuous function $f : \omega \mapsto \mathbb{R}$ which satisfies $\Gamma = f^{-1}(0)$ and such that ∇f is the unit normal vector field for any level set $f^{-1}(c)$. Under this formulation, the unit normal vector field $N = \nabla f$ is curl-free in a narrowband ω of the surface Γ. The real number r is related to the radius of the surface mean curvature. The Figure below illustrates that for each set $\omega_c \subset \Omega$ and for a small radius r, we have $|\nabla h| = cste$.

5 Display $|\nabla h|$

Below, we display $|\nabla h|$. It is clear that when evaluated on each isocurve $h = h_{iso}$, $|\nabla h| \simeq cste$.

In [280]:
```
##Compute the normalized gradient vector field N

        def compute_normalized_gradient(h):

            Nx = np.zeros(h.shape)
            Ny = np.zeros(h.shape)

            nabla_hx, nabla_hy = np.gradient(h)
            gaussian_filter(nabla_hx, sigma = 2, output = nabla_hx)
            gaussian_filter(nabla_hy, sigma = 2, output = nabla_hy)

            grad_norm = np.sqrt(nabla_hx**2 + nabla_hy**2)

            grad_norm[grad_norm==0] = 0.000000000001 ## to avoid dividing by zero

            np.divide(-nabla_hx, grad_norm, Nx)
            np.divide(-nabla_hy, grad_norm, Ny)

            return Nx, Ny, grad_norm

        Nx, Ny, grad_norm =  compute_normalized_gradient(h)

        fig = plt.figure(figsize=(10, 10))
        plt.imshow(grad_norm, cmap='jet')
        plt.colorbar()
        plt.title("grad_norm")
        plt.show()
```



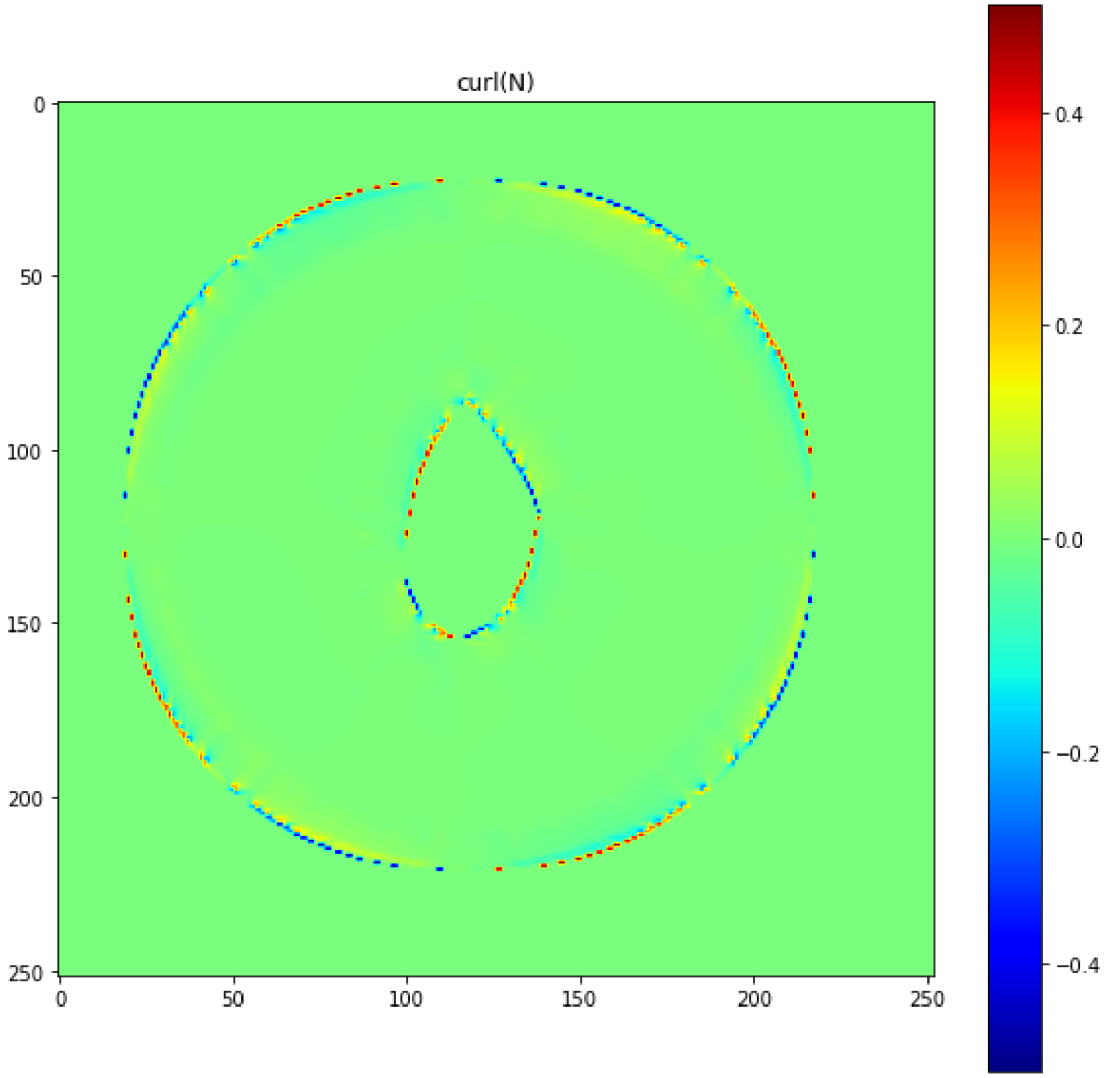
6 Compute $curl(N) = \nabla \times \nabla h$

Let $v = (p,q)^T$ be a 2D vector field, then the curl of v is given by: $curl(v) = \frac{\partial q}{\partial x} - \frac{\partial p}{\partial y}$. Source: https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/curl-grant-videos/v/2d-curl-formula

It is clear that N is curl free inside the region of interest Ω.

```
In [281]: def curl2D(p, q):
              dp_dx, dp_dy = np.gradient(p)
              dq_dx, dq_dy = np.gradient(q)

              return dq_dx - dp_dy
```

```
curl_N = curl2D(Nx, Ny)
fig = plt.figure(figsize=(10, 10))
plt.imshow(curl_N, cmap='jet')
plt.colorbar()
plt.title("curl(N)")
plt.show()
```

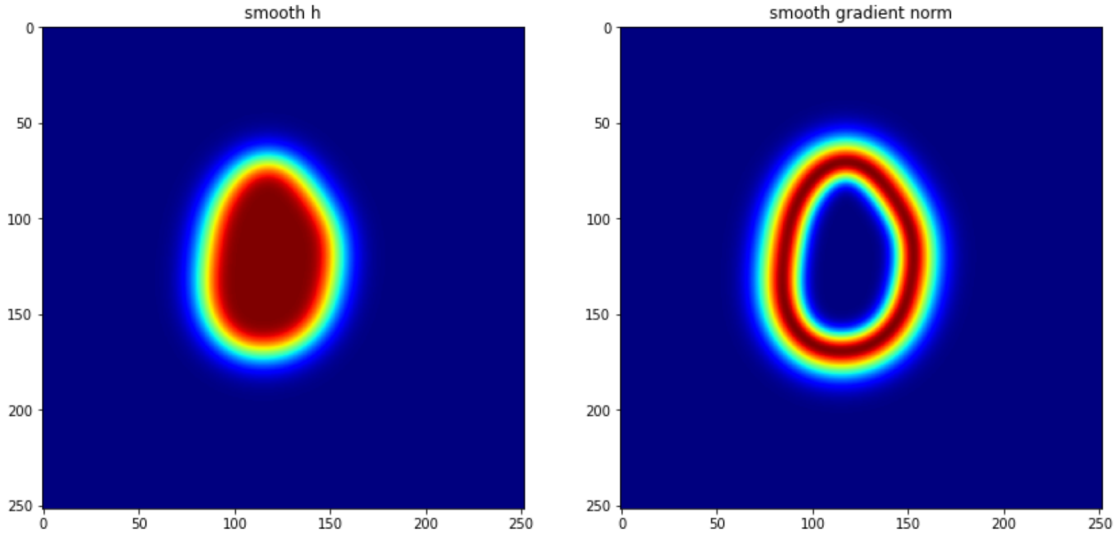


7 Effect of smoothing the heat distribution

In the next cell, we will show the effect of smoothing the solution h on the energy conservation property of the corresponding normalized gradient vector field N

```
In [282]: def local_gaussian_filter(scalar_function, sigma=2):
```

```
    mask = np.zeros(scalar_function.shape)
    mask[scalar_function!=0] = 1
    smooth_scalar_function = gaussian_filter(scalar_function*mask, sigma=sigma)

    return smooth_scalar_function

h_smooth = h.copy()

h_smooth = local_gaussian_filter(h_smooth,5)

Nx_smooth, Ny_smooth, grad_norm_smooth =  compute_normalized_gradient(h_smooth)


fig = plt.figure(figsize=(14, 14))

ax1 = plt.subplot(121)

ax1.imshow(h_smooth, cmap="jet")

plt.title("smooth h")

ax2 = plt.subplot(122)
ax2.imshow(grad_norm_smooth, cmap="jet")
plt.title("smooth gradient norm")
plt.show()
```

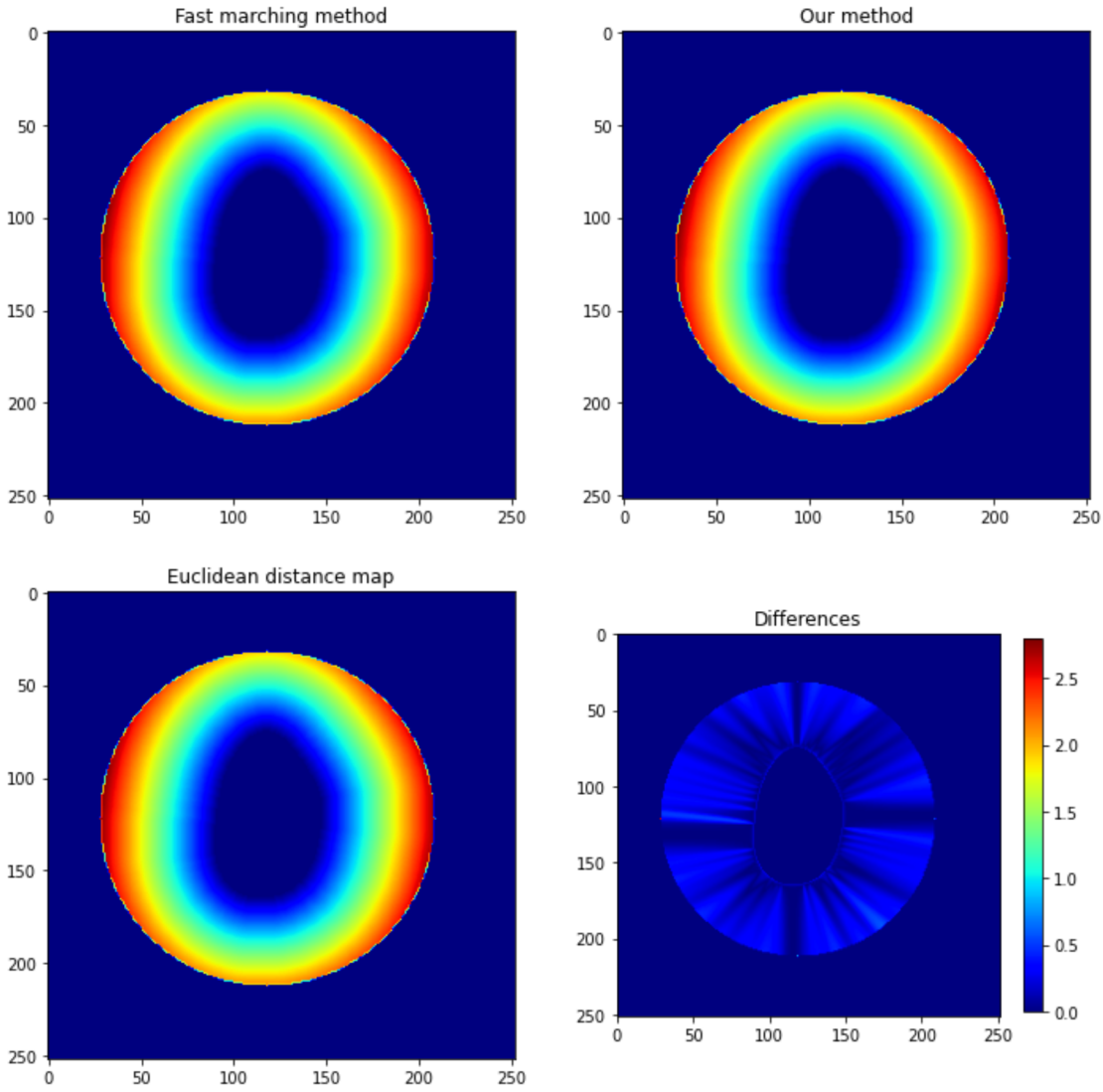


8 Recover geodesic distance from N

Below, we compare our results with those of the Fast marching method. We use the implementation of the Fast marching method (skfmm library) which solves directly the Eikonal PDE.

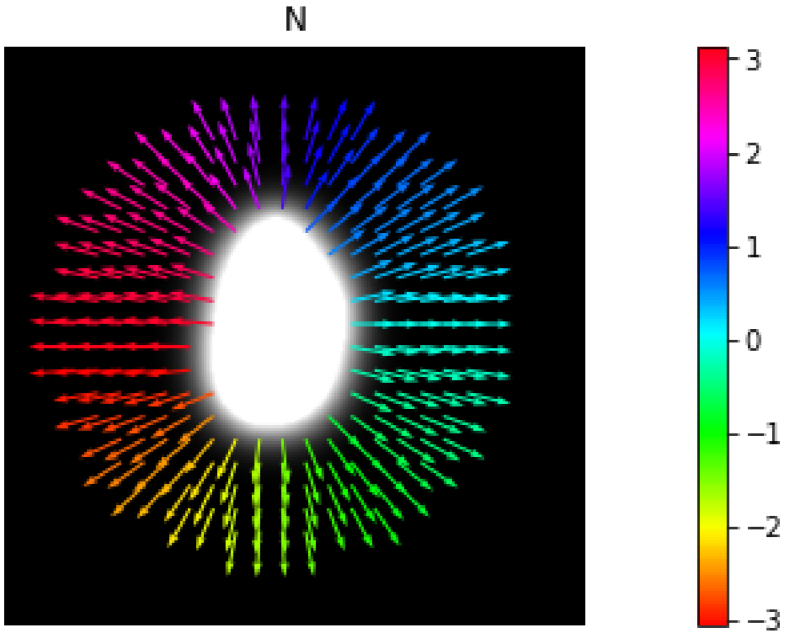
In [283]:
```
import  skfmm
from scipy import ndimage

# Boundary conditions for computing L
L = np.zeros(mask.shape)
L_n = np.zeros(mask.shape)

L[:,:]= -0.5 # For more details, see Figs 4.3 and 4.4
#in https://tel.archives-ouvertes.fr/tel-02414706/document

den = np.absolute(Nx)+ np.absolute(Ny)
den[den==0] = 0.000000000001 # to avoid dividing by zero


for it in range (200):
    #print(it)

    L_n = L

    L[R[0],R[1]] =  (1 + np.absolute(Nx[R[0],R[1]])* L_n[(R[0]\
        -np.sign(Nx[R[0],R[1]])).astype(int)\
        ,R[1]]+ np.absolute(Ny[R[0],R[1]]) * L_n[R[0],(R[1]-\
        np.sign(Ny[R[0],R[1]])).astype(int)])\
        /  den[R[0],R[1]]




### Fast marching method: https://pythonhosted.org/scikit-fmm/
L_fm = skfmm.distance(np.max(mask)-mask)

### Euclidean distance map

L_Euclid = ndimage.distance_transform_edt(np.max(mask)-mask)

Omega = np.zeros(mask.shape)
Omega[R]=1

fig = plt.figure(figsize=(12, 12))

ax1 = fig.add_subplot(2,2,1)
ax1.imshow(L_fm*Omega, cmap='jet')
ax1.set_title("Fast marching method")
```

```
ax2 = fig.add_subplot(2,2,2)
ax2.imshow(L*Omega, cmap='jet')
ax2.set_title("Our method")


ax3 = fig.add_subplot(2,2,3)
ax3.imshow(L_Euclid*Omega, cmap='jet')
ax3.set_title("Euclidean distance map")

diff = np.absolute(L_fm*Omega - L*Omega)

ax4 = fig.add_subplot(2,2,4)
ax4.imshow(diff, cmap='jet')
ax4.set_title("Differences")




pos = ax4.imshow(diff*Omega, cmap='jet')
fig.colorbar(pos,ax=ax4,shrink=0.8)
plt.show()
```



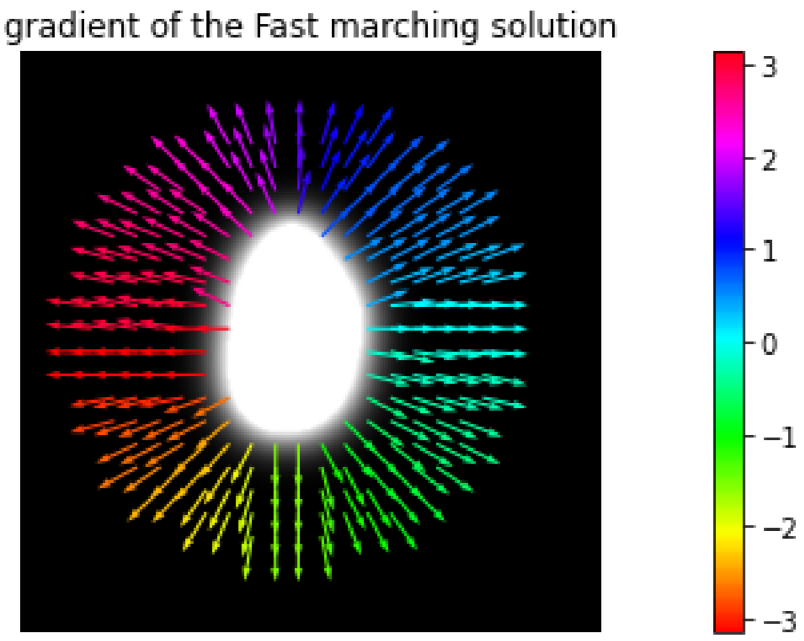

```
In [284]: def plot_gradient(Nx,Ny,R, title):

              x, y = R[0],R[1]
              step = 10
                  # grid downsampling
              xn = np.extract(np.logical_and(np.mod(x, step)==0,np.mod(y, step)==0), x)
              yn = np.extract(np.logical_and(np.mod(x, step)==0,np.mod(y, step)==0), y)
              vx = Nx[xn,yn]
              vy = Ny[xn,yn]

              plt.axis("equal")
              plt.imshow(h, cmap=plt.cm.gray)
              plt.quiver(yn, xn, vy,vx, - np.arctan2(vx, vy), angles='xy',\
```

```
        units = 'inches', cmap='hsv', scale_units='xy', scale=0.05)
    plt.colorbar()#orientation="horizontal")
    plt.axis('off')
    plt.title(title)
    #plt.savefig("./gradient_vector_field.png",dpi=500)
    plt.show()

plot_gradient(Nx,Ny,R, "N")
```



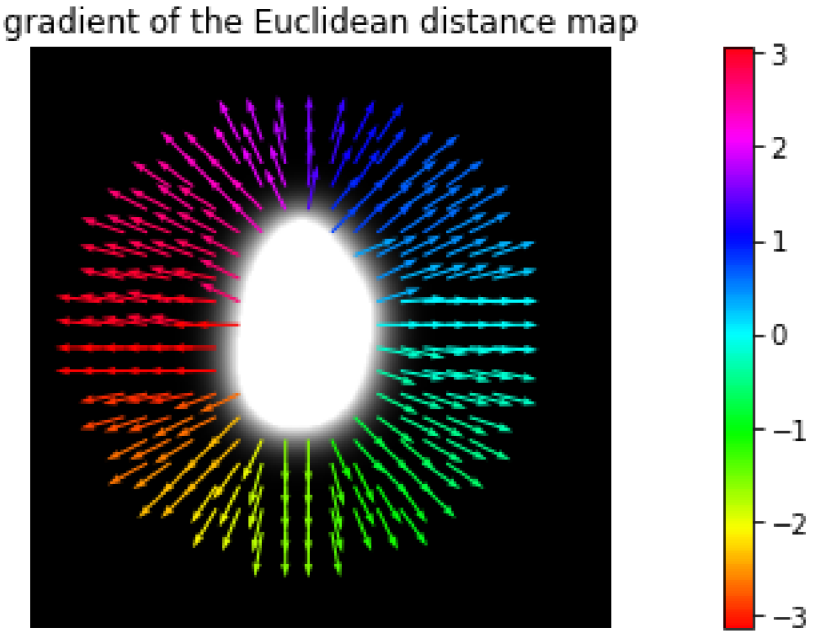
Above, we have plotted N, below, we show ∇L_{FM}, where L_{FM} is the solution obtained by directly solving the Eikonal equation.

```
In [285]: Nx_eikonal, Ny_eikonal = np.gradient(L_fm)
          plot_gradient(Nx_eikonal, Ny_eikonal,R, "gradient of the Fast marching solution")
```



Above, we have plotted N, below, we show ∇L_{Euc}, where L_{Euc} is the conventional Euclidean distance map.

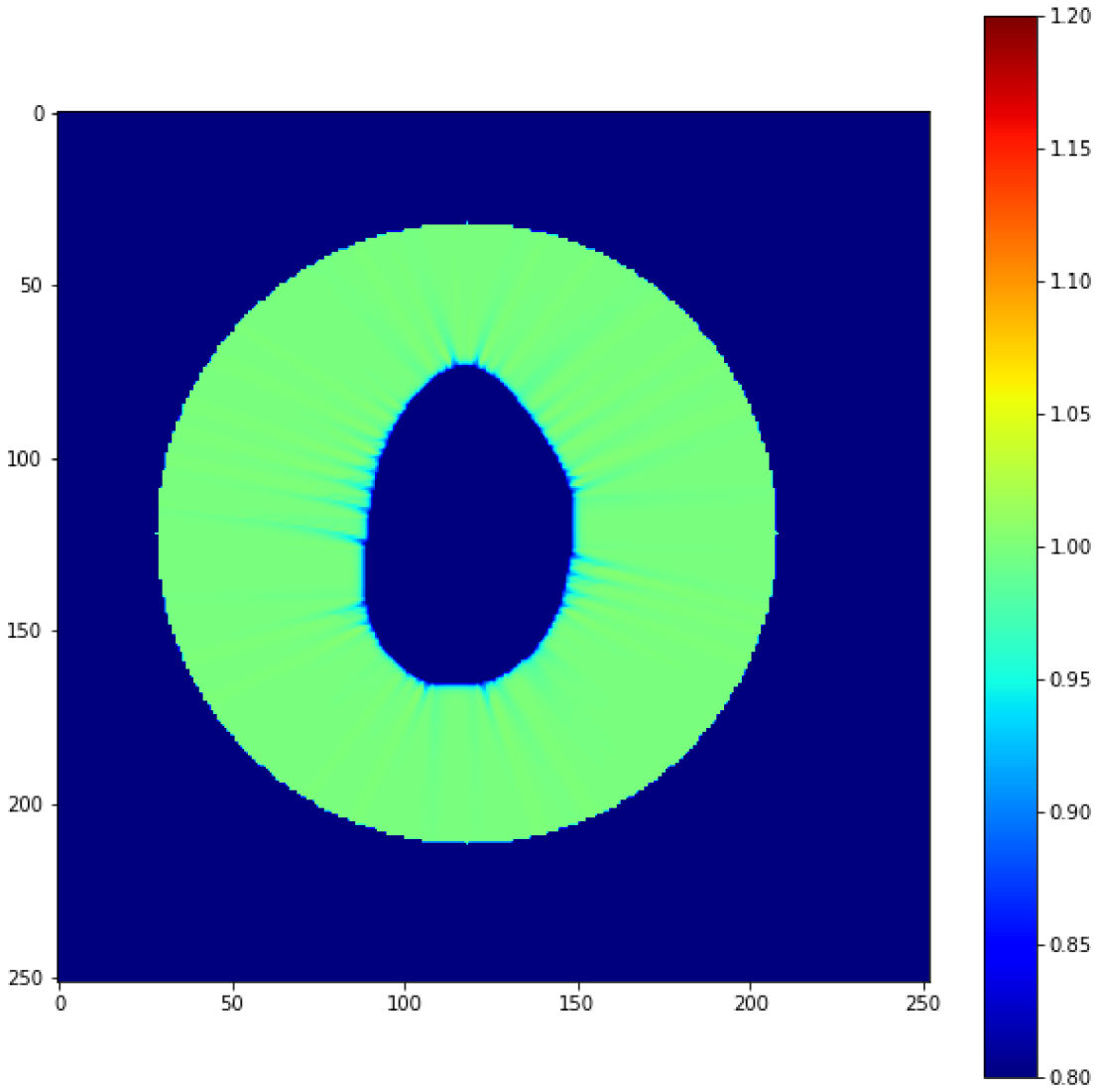
```
In [286]: Nx_Euclid, Ny_Euclid = np.gradient(L_Euclid)
          plot_gradient(Nx_Euclid, Ny_Euclid,R, "gradient of the Euclidean distance map")
```



9 Compute scalar product $N \cdot \nabla L_{FM}$

Finally, we show the result for the scalar product $N \cdot \nabla L_{FM}$ (Which should be equal to 1 inside Ω if our distances are equal or nearly equal to the distance map obtained when using the Fast marching algorithm)

In [287]:
```
dot_product = np.zeros(Nx.shape)
dot_product += Nx*Nx_eikonal + Ny*Ny_eikonal



fig = plt.figure(figsize=(10, 10))

plt.imshow(np.array(dot_product*Omega,float), cmap = 'jet')
plt.colorbar()
plt.clim(0.8,1.2)
plt.show()

print(np.max(dot_product[R]), np.mean(dot_product[R]))


'''
dot_product1 = np.zeros(Nx.shape)
dot_product1 += Nx_Euclid*Nx_eikonal + Ny_Euclid*Ny_eikonal

fig = plt.figure(figsize=(10, 10))

plt.imshow(np.array(dot_product1*Omega,float), cmap = 'jet')
plt.colorbar()
plt.clim(0.8,1.2)
plt.show()

print(np.max(dot_product1[R]), np.mean(dot_product1[R]))
'''
```




```
1.0083124774102554 0.9943276087309536
```

```
Out[287]: "\ndot_product1 = np.zeros(Nx.shape)\ndot_product1 += Nx_Euclid*Nx_eikonal + Ny_Euclid
```

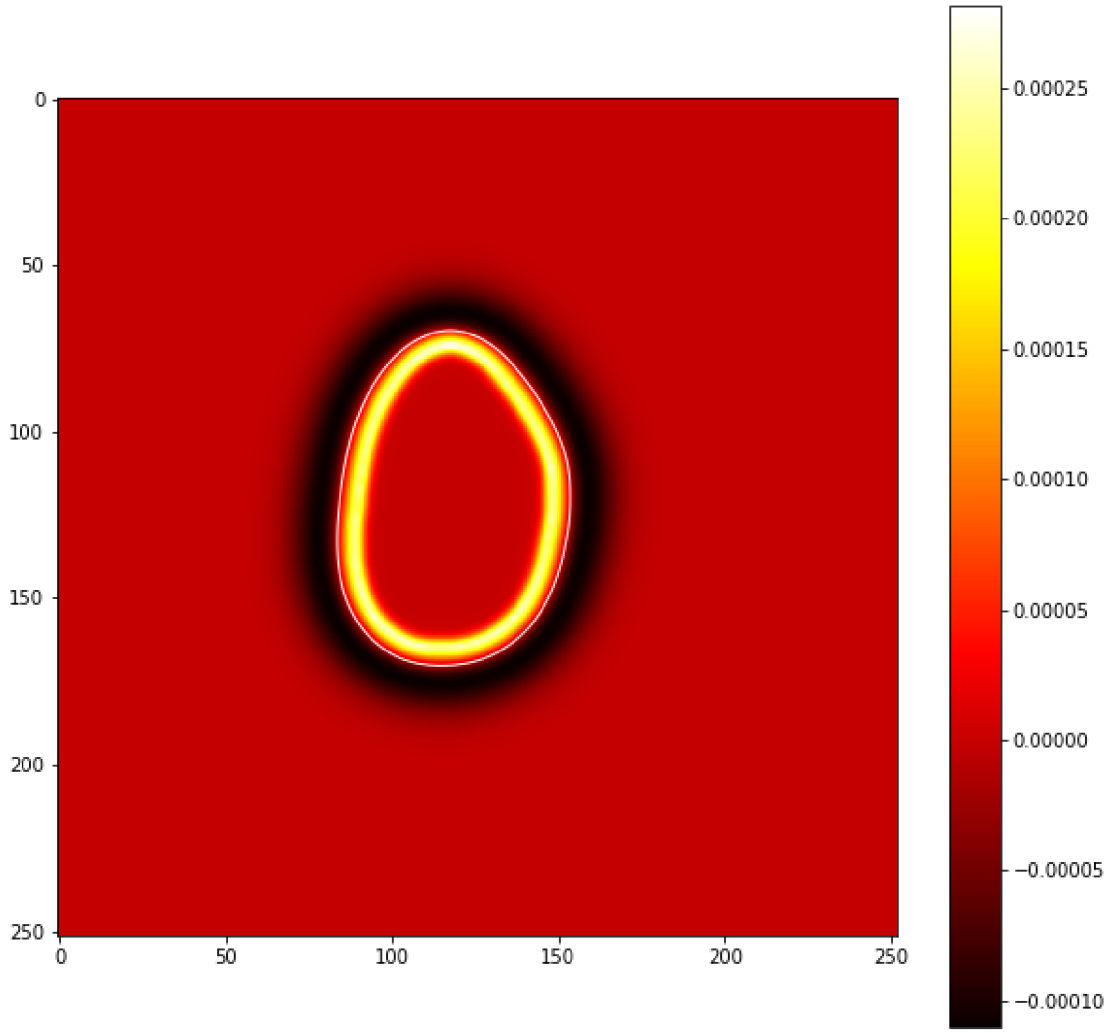
10 Compute $f = \frac{h-h_{iso}}{|\nabla h|}$ such that $h_{iso} = 0.5$

We display the isocurve $\{f = 0\}$ in black.

Remark: Since the curl is a linear operator, then we can use $f = \frac{h-h_{iso}}{\alpha|\nabla h|}$, for example, here we use $\alpha = 100$.

```
In [288]: f = np.zeros(h.shape)
          f = (h - 0.5)/100*grad_norm
```

```
fig = plt.figure(figsize=(10, 10))
#f = local_gaussian_filter(f,7)
plt.imshow(f, cmap='hot')
plt.colorbar()

#plt.clim(-100,1000)
plt.contour(f, [0.], colors=['w'], linewidths=1.0)
plt.show()
```



11 Display $|\nabla f|$

In [289]: `Nx, Ny, grad_norm =  compute_normalized_gradient(f)`

```
fig = plt.figure(figsize=(10, 10))
plt.imshow(grad_norm, cmap='jet')
plt.colorbar()
plt.title("grad_norm")
plt.show()
```