Efficient globally optimal segmentation of cells in fluorescence microscopy images using level sets and convex energy functionals

Tania Aguirre
ENS-Paris Saclay
Saclay, France
tania.aguirre_lordoguin@ens-paris-saclay.fr

Laura Fuentes
Université Paris Saclay
Saclay, France
laura.fuentes-vicente@universite-paris-saclay.fr

1 INTRODUCTION (L)

Image segmentation is the process of partitioning an image into multiple regions [1]. In the article, the authors address this technique for the purpose of cell nuclei segmentation in fluorescence microscopy images. Several approaches have been introduced over the years, and difficulties have arisen in parallel. One of the main approaches is called thresholding. This technique assumes that segments in the image have similar intensity and creates a binary mask based on a threshold intensity value. Nevertheless, its performance tends to be affected by the shading of intensity inhomogeneities. These problems arise from technical limitations or artifacts introduced by the imaged object.

Deformable models are based on parametric or implicit models, allowing the incorporation of a priori knowledge and adopting a broader range of shapes. Parametric models rely on explicit representations of objects, such as active contour models and region-based energy functionals, but are dependent on parameters and susceptible to difficulties associated with topological changes. Implicit models could offer an alternative solution as they address the issue of topological changes.

This method employs level set functions coupled with gradient-based energy functions and defines active contour gradient and region-based terms. The objective is to minimize the energy function; however, the main challenge lies in non-convex optimization problems leading to local minima and issues related to initialization.

A summary of the challenges in segmentation includes intensity inhomogeneities, topological changes, parameter initialization, and non-convex minimization problems.

The paper introduces an implicit method utilizing level set functions and active contour energy functionals, leading to convex optimization problems. Two distinct methods are proposed for segmenting clustered cell nuclei. The first method involves a 3-step implementation based on a convex reformulation of the equations found in [3] and [10]. The second technique adopts a 2-step approach based on the equation from [9] and [10]. The authors employ the Split Bregman algorithm paired with Gauss-Seidel to iteratively solve the optimization problem.

2 METHODS (L)

The implementation of the algorithm for any of the three energy functions necessitates the initialization of a level set function. Level set functions are utilized to define a contour along the border, being positively (respectively, negatively) defined in the foreground (respectively, background) and associated with zero values on the object boundary. Energy functions guide the evolution of the level set function over steps and consist of two terms: a data fidelity term that gauges the performance of the level set function in aligning with the image, and a regularization term that promotes smoothness. The level set function is iteratively updated by minimizing the energy function:

$$\min_{0 \le \phi_n \le 1} E_j^c(\Theta, \phi_n) = \lambda < \phi_n, r_j > + |\nabla \phi_n|_1 \quad (1)$$

where r_j refers to the the energy force related to $E_j^c(\Theta, \partial\Omega)$.

The article addresses cell nuclei segmentation by reiterating the problem described in (1) with various energy forces r_j at different iteration levels. [Figures 1, 2] depict two diagrams illustrating the two proposed modes in the article. The first approach resolves the convex problem described in (1) three times, whereas the second approach solves it twice, both with different energy functions at each step.

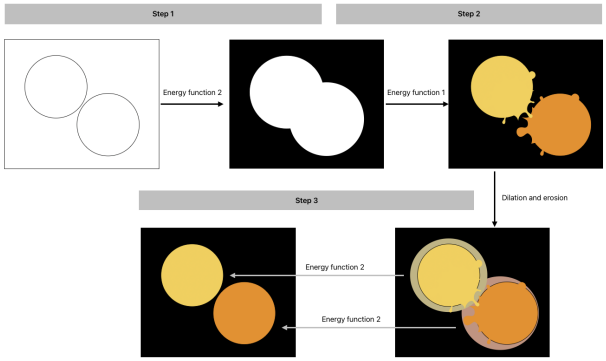


Figure 1: 3-step algorithm

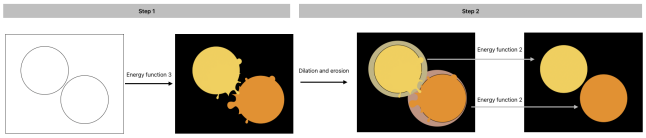


Figure 2: 2-step algorithm

2.1 The Split Bregman Method Applied to Globally Convex Segmentation (T)

The algorithm strongly relies on optimizing the problem described in (1), which formulates a globally convex segmentation problem (GSC). To address the non-convexity issue observed in previous studies and achieve easily manageable solutions, the authors employ two strategies. First, they utilize derivative versions of the

energy functions, and secondly, they omit the Heaviside function as solutions in gradient descent remain identical. To ensure a well-defined global minimal solution, the level set function needs to be confined to the interval $[0, 1]$[5].

The problem is equivalent to introducing an auxiliary variable $\vec{d}$ by enforcing the constraint $\vec{d} = \nabla\phi_n$ and transforming the problem into an unconstrained form:

$$\underset{0\le\phi_n\le 1,\vec{d}}{\arg\min}\left(\lambda < \phi_n, r_j > +|\vec{d}|_1 + \frac{\nu}{2}||\vec{d} - \nabla\phi_n||_2^2\right) \tag{2}$$

where we introduce a quadratic penalty function on $\vec{d}$. The optimal solution can be found by initially minimizing with respect to ϕ_n, solving the differentiable optimization problem. Subsequently, the final solution is determined by minimizing the auxiliary variable $\vec{d}$. Once the optimization problem is resolved, the segmented region is obtained by thresholding the level set function:

$$\Omega = \{x : \phi_n(x) > \alpha\} \tag{3}$$

for some $\alpha \in (0, 1)$

The article employs the Split Bregman algorithm iteratively to solve the problem due to its efficiency in L1-regularized problems and its ability to reach global maxima. This method involves variable splitting and Bregman iteration using a fixed λ, with a new auxiliary vector $\vec{b}$ incorporated in the penalty term. The problem is reformulated as follows: $(\phi_n^{k+1}, \vec{d}^k)$

$$= \underset{0\le\phi_n^k\le 1,\vec{d}^k}{\arg\min}\left(\lambda < \phi_n^k, r_j > +|\vec{d}^{k-1}|_1 + \frac{\nu}{2}||\vec{d}^{k-1} - \nabla\phi_n^k - \vec{b}^{k-1}||_2^2\right) \tag{4}$$

$$\vec{b}^k = \vec{b}^{k-1} + \nabla\phi_n^k - \vec{d}^k \tag{5}$$

where $\frac{\nu}{2}||d - \nabla\phi_n - b^{k-2}||_2^2$ refers to the constraint.

As mentioned previously, each step of the iteration involves two alternating minimizations. Initially, we minimize with respect to ϕ_n with $\vec{b}$ and $\vec{d}$ fixed. The minimizer is characterized as the solution of the following differential equation:

$$\Delta\phi_n(x) = \frac{\lambda}{\nu} r_j + \nabla\cdot(\vec{d} - \vec{b}) \tag{6}$$

The article suggests solving the differential equation using Gauss-Seidel, which will be described at the end of this section.

Secondly, the shrink operator is employed for updating $\vec{d}$:

$$\vec{d}^k = max\{|\vec{b}^{k-1} + \nabla\phi_n^k|_2 - \nu, 0\}\frac{\vec{b}^{k-1} + \nabla\phi_n^k}{|\vec{b}^{k-1} + \nabla\phi_n^k|_2}$$

and we finally update $\vec{b}^k = \vec{b}^{k-1} + \nabla\phi_n^k - \vec{d}^k$.

For our implementation, we particularly utilized the pseudo-code presented in [5]. This paper suggests a discretization of the level set function as well as $\vec{d}^k$ and $\vec{b}^k$, enabling straightforward computation of the iterative updates.

The next level set function is computed by iteratively solving the differential equation described in (6) using the Gauss-Seidel method. Considering the auxiliary vectors as $\vec{d} = (d_x, d_y)$ and $\vec{b} = (b_x, b_y)$, the method calculates each component of the level set using the following equations:

Algorithm 1 Split Bregman for GCS

while $||\phi^k - \phi^{k-1}|| > \epsilon$ **do**
  Define r^{k-1} the energy function for the level set ϕ_n^{k-1}
  $\phi_n^k = GS_{GSC}(r^{k-1}, \vec{d}^{k-1}, \vec{b}^{k-1})$
  $\vec{d}^k = shrink(\nabla\phi_n^k + \vec{b}^{k-1}, \nu)$
  $\vec{b}^k = \vec{b}^{k-1} + \nabla\phi_n^k - \vec{d}^k$
  Find $\Omega^k = \{x : \phi_n^k(x) > 0\}$
  Update μ_0 and μ_1
  (and σ_0, σ_1 depending on the energy functional)
end while

$$\alpha_{i,j} = d_{i-1,j}^x - d_{i,j}^x - b_{i-1,j}^x + b_{i,j}^x + d_{i,j-1}^y - d_{i,j}^y - b_{i,j-1}^y + b_{i,j}^y \tag{7}$$

$$\beta_{i,j} = \frac{1}{4}(\phi_{i-1,j} + \phi_{i+1,j} + \phi_{i,j-1} + \phi_{i,j+1} - r_{i,j} + \alpha_{i,j}) \tag{8}$$

$$\phi_{i,j} = max(min(\beta_{i,j}, 1), 0) \tag{9}$$

In conclusion, the presented solution iteratively solves the convex problem described in equation (1) for any energy force r_j. To address the problem, it is essential to initially minimize the level set function using the iterative Gauss-Seidel method. This is followed by an update on the auxiliary vectors $\vec{d}$ and $\vec{b}$ until convergence is achieved, resulting in the same level set function.

2.2 Energy functions (L)

The fundamental concept behind energy functions is to guide the evolution of level set functions subject to certain constraints [3]. Edge detectors are linked to the gradient of the image, and the minimization of the energy function involves positioning the level set function at the maximum of the image's gradient, emphasizing smoothness in the process.

In the paper, three functionals are introduced to address the segmentation problem:

$$r_1 = \kappa_1(I(x) - \mu_1)^2 - \kappa_0(I(x) - \mu_0)^2 \tag{10}$$

The first functional r_1 (10) corresponds to the energy functional [Appendix A.2: $E_1(\Theta, \partial\Omega)$], introduced in [3]. This function seeks to minimize the Mumford-Shah functional by identifying the optimal approximation of the real image with smooth regularization. It operates under the assumption that the image is a piecewise constant function. However, this assumption does not account for local intensity information, rendering it susceptible to intensity inhomogeneities. For each region, the authors dynamically compute μ_i, minimizing the global fit energy corresponding to the average.

$$r_2 = log(P(I(x)|\Omega_1) - log(P(I(x)|\Omega_0)^{\text{i}} \tag{11}$$

Next, r_2 (11) corresponds to a Bayesian functional [Appendix A.2: $E_2(\Theta, \partial\Omega)$], introduced in [10]. This functional relies on the conditional probability that $I(x)$ belongs to region Ω_i ($P(I(x)|\Omega_i)$), derived from the Gaussian density family function. By introducing the standard deviation term σ, emphasis is placed on local

[i] $\text{P(I(x)}|\Omega_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{(I(x)-\mu_i)^2}{2\sigma_i^2}}$

areas. Consequently, our energy function provides a localized approximation of information, effectively addressing global intensity inhomogeneities while enhancing the discrimination of regions. As ϕ evolves, the Gaussian parameters are updated iteratively.

$$r_3 = \kappa_1 \int K_\sigma(y-x)|I(x)-f_1(y)|^2 dy - \kappa_0 \int K_\sigma(y-x)|I(x)-f_0(y)|^2 dy \text{[ii]} \tag{12}$$

Finally, r_3 (12) corresponds to the region-scalable fitting energy functional [Appendix A.2: $E_3(\Theta, \partial\Omega)$], proposed in [9]. This function is a part of the region-based active contour functions. Similar to last implementation, this energy function incorporates a scaling parameter, σ, to determine the size of the region. This parameter is introduced in $f_i(x)$, representing the approximation of the image intensity centered at x, as well as in a Gaussian kernel. The kernel function utilizes the intensity information of the region controlled by σ by reducing the contribution of points that move far away from x. The significance of this scaling parameter lies in its ability to focus from small neighborhoods to the entire image, providing flexibility to address both global and local intensity inhomogeneities in the image.

2.3 Step by step implementations (L)

In the 3-step method, they employ a combination of $E_1^c(\Theta, \partial\Omega)$ and $E_2^c(\Theta, \partial\Omega)$.

2.3.1 3-step method.

As a first step, they apply E_2^c to the entire image for segmenting cell nuclei. The scaling factor σ_i accommodates varying background and foreground intensities, addressing intensity inhomogeneities. In the second step, they minimize E_1^c on the previously segmented regions from step 1, eliminating the need for a coupling term. This step is aimed at separating cell nuclei that were falsely merged in the initial segmentation (multiple merged cells segmented). However, E_1^c still considers homogeneity among regions, leaving intensity inhomogeneities unresolved. The final step involves reapplying E_2^c over a dilated version of the segmented outputs from step 2. This operation results in a more accurate and smoother boundary while overcoming intensity inhomogeneities [Figure 1].

2.3.2 2-step method.

In the two step approach, they rather use a combination of $E_3^c(\Theta, \partial\Omega)$ and $E_2^c(\Theta, \partial\Omega)$.

In the first step, they apply E_3^c, which addresses both local and global intensity inhomogeneities. This energy function incorporates the term K_σ that provides high sensitivity to local varying mean intensities. This characteristic prevents the merging issues encountered in the 3-step approach [Figure 1, Step 1] and handles variations in background intensities. Similar to the 3-step method, the final step involves implementing E_2^c over the dilated segmented output to enhance boundary precision and smoothness.

After segmentation is completed, regions of interest are extracted by thresholding on the level set function.

[ii] $K_\sigma(u) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{|u|^2}{2\sigma^2}}$

3 RESULTS (T)

The approach described above is applied to 2-D fluorescence microscopy images of cell nuclei from four different experiments. The experiments compare different cell types.

The authors used four different sets, three of which have ground truth available [4, 7], while for the last set they manually outline the borders of the nuclei. A full description of the dataset can be found in [Appendix: A.1].

The automatic evaluation is done using traditional region-based measures, like the Dice coefficient, and contour-based measures, such as the normalised sum of the distance and the Hausdorff distance [Appendix: A.3]. The article also presents two detection measures, such as the number of false positives (FP) and the number of false negatives (FN).

The article makes a complete analysis of the method: presents an evaluation of the state-of-the-art techniques of the moment, describes a sensitivity analysis of the parameters, and evaluates the time consumption between convex and non-convex approaches. In addition, they manually relabelled 5 images from each dataset to compare their results.

The choice of default parameters was based on a subset of the dataset that gave better results. However, it is not mentioned how this subset was selected or how many. The figures [Table 2, Table 1] show the default parameters for the 3-step and 2-step approaches. The parameter d is associated to the erosion, but there is no clear relation mentionned. For the convergence criteria of the Split Bregman algorithm, they evaluate whether the contour solution has changed using the Euclidean distance between the currently computed contour and the previous one.

3 - step approach						
Step	**Energy**	κ_0	κ_1	ν	λ	d
1	E_2	-	-	10	10000	-
2	E_1	1	1	10	1000	-
3	E_2	-	-	10	1000	28[iii]

Table 1: Default parameters used for the 3 - step approach.

2 - step approach							
Step	**Energy**	κ_0	κ_1	ν	λ	σ	d
1	E_2	-	-	10	10000	16[iv]	-
2	E_1	1	1	10	1000		28[v]

Table 2: Default parameters used for the 2 - step approach.

In light of the expanded analysis, our attention will be exclusively directed towards the U20S dataset, given its larger image volume and superior image quality.

The method is compared with other traditional algorithms such as Otsu Thresholding, Watershed algorithm and Merging algorithm. When considering the region and contour based metrics, the best of these three is the Merging algorithm. Nevertheless, the 3-step approach improves the results slightly, giving an improvement of +0.02, +0.05 in Dice, NSD and equal results in Hausdorff. While the

2-step approach improves the Dice metric by +0.02, the NSD metric by +0.06 and the Hausdorff metric by +0.5. Looking at the detection analysis, it is not possible to extract any useful information.

This analysis is also compared using the energy functions as a non-convex and convex approach for energy functions two ($E_2(\Theta, \partial\Omega)$) and three ($E_3(\Theta, \partial\Omega)$). In all cases the results are better with convex approaches. An improvement of +0.05, +0.04 and +7 for each metric for $E_2(\Theta, \partial\Omega)$. While for $E_3(\Theta, \partial\Omega)$ the improvements are +0.05, +0.05 and +3.4 for each metric. Furthermore, using the convex approach significantly reduces the number of iterations, for example for energy function two using the proposed convex approach takes 5 iterations (15 seconds) while using the non-convex approach takes 400 iterations (9 minutes). While for energy function three it goes from 10 iterations (100 seconds) for the convex approach to 1000 iterations (35 minutes).

The article compares both approaches to manually describing 5 images within each dataset. The results are presented in [Table 3], where both approaches yield similar results, although the 2-Step approach measures better in metrics for both cases. The figure [Table 4] demonstrates the same experiment conducted on the NIH3T3 dataset, which contains images with significant inhomogeneities. The results reveal that manual labelling is challenging to achieve accurate results. However, the 2-step approach is more robust and enhances the results compared to the manual method.

Approach	Dice	NSD	Hausdorff
Manual	0.93	0.04	9.8
3-Step approach	0.94	0.06	13.3
2-Step approach	**0.94**	**0.05**	**12.8**

Table 3: Comparison between the proposals and manual labeling on 5 images for the dataset U20S cells.

Approach	Dice	NSD	Hausdorff
Manual	0.87	0.07	12.1
3-Step approach	0.83	0.06	13.3
2-Step approach	**0.94**	**0.05**	**12.8**

Table 4: Comparison between the proposals and manual labeling on 5 images for the dataset NIH3T3 cells.

The sensitivity analysis demonstrated that variations in the parameters do not substantially alter the metrics, indicating that the algorithm is notably robust.

3.1 Our results

We established a GitHub repository to apply a 3-step approach using a simplified version of the U20S cell dataset featured in the referenced article. Access the repository via this link. Our implementation incorporates the Split Bregman algorithm, investigating the impact of each image force r_j for $j \in 1, 2, 3$ post-convergence.

Notably, most level set outputs required normalization to confine results between 0 and 1. However, the paper lacked a solution for this issue. To address this, two normalisation methods were introduced; one by clipping values between 0 and 1 [Appendix A.4: Figure 3], and the other by normalising all level sets based on the first calculated level set [Appendix A.4: Figure 6]. Using the initial approach resulted in a significant information loss, leading the algorithm to converge towards an incorrect level set. Consequently, we opted for the alternative approach.

After evaluating the energy functions using the same level set input, distinct behaviors were detected between the two observed functions. Notably, it was observed that energies two and three were inverted. This was prompted by the convergence of energy three to energy one, as stated in the article. To attain algorithm convergence, we had to change the sign of both energies. This raised concerns about potential errors in our implementation of the energy function or the Bregman algorithm. Nonetheless, we verified both functions individually. Another point to mention is that the article does not specify the size of the Gaussian kernel, which is dependent on the value of sigma. We arbitrarily choose a value of $kernel\_size = (2 * \sigma + 1, 2 * \sigma + 1)$. A full analysis of energy 3 can be seen in the notebook [Appendix A.5].

With these changes, we observed comparable results in the outputs of energy functions 2 and 3, except for the α selection influencing region definitions. By manipulating the sigma parameter in energy function 3, we shed light on its role in both local and global focus [Appendix A.4: Figure 7].

In scrutinizing the 3-step approach, various inaccuracies posed challenges during our implementation. In this case, we did not achieve good results when working separately [Appendix A.4: Figure 8]. The code for each step was generated using an energy solver (force and convergence algorithm as inputs). Thanks to this implementation, we have become aware of missing details in the article. Specifically, we noted that before utilizing energy function 1 (3-Step: Step 2), several critical aspects were left unexplained. For instance, the paper did not clarify the criteria for selecting α to define regions ($\Omega = x : \phi_n(x) > \alpha$), determining the number of segments, or the necessity of applying a distance function. Lastly, the third step involves dilation and erosion of the output from the preceding step, yet the specification of the binary mask required for dilation was omitted.

Additionally, we attempted a different implementation specifically for the first energy, which was based on skimage library implementation. This alternative used a different algorithm to calculate the differential equation, for this implementation we were able to completely segmentate one of the cells [Appendix A.4, Figure 9]. Unfortunately, we were unable to adapt this implementation to the other energy functions due to time constraints. More information about the work done can be found in Appendix A.6.

4 CONCLUSION (L)

In the realm of medical image segmentation, the primary objective has traditionally revolved around identifying pixels associated with organs or lesions to extract valuable information on their shapes and volumes [6]. The current state-of-the-art solutions in this domain largely leverage deep learning applications for their versatility, high performance, and generalization capabilities [2]. However, it's important to note that these applications are relatively recent, and a pivotal moment can be traced back to 2012 when neural networks demonstrated significant performance in computer

vision tasks. Prior to this turning point, segmentation primarily relied on edge detection filters and mathematical approaches. Since then, the advent of deep learning has witnessed the emergence of increasingly sophisticated networks, facilitated by the development of robust computational infrastructures, GPUs, and the exponential growth of artificial intelligence. Present-day medical image segmentation research prominently features Convolutional Neural Networks (CNNs) in 2-D, 2.5-D, and 3-D dimensions, Recurrent Neural Networks (RNNs), and attention models [2].

It's crucial to acknowledge a temporal nuance when assessing this article in comparison to contemporary research. Published in 2012, this article aligns with earlier studies, offering an insightful perspective compared to the latest techniques. While their approach effectively addresses many challenges encountered in medical image segmentation, it presents a handcrafted technique based on previous studies. The algorithm proposed in the article draws inspiration from various fields, such as region-based methods [Equations: Appendix (13), (15)], Expectation-Maximization (EM), [Equations: Appendix (14)], region-based methods, and Partial Differential Equations [Equations: (1), (6)], incorporating both the advantages and drawbacks of each [8]. Notably, their algorithm tackles intensity inhomogeneities and convexity problems but is computationally complex and resource-intensive.

The paper describes that managing the automatic analysis was a challenge for some datasets with strong non homogeneity, but does not describe how they tackled this problem.

Lastly, a critical observation regarding the article's structure concerns the length of the introduction compared to the specificity of the Gauss-Seidel algorithm implementation. While such an implementation is mentioned, crucial parameters are not specified, complicating the reproducibility of results. This problem repeats for other parts of the algorithm like the definition of the parameter d of erosion or the definition of the Gaussian kernel size. This lack of detail hinders the transparency and robustness of the study, impeding the replication and building upon of the proposed methodology.

In our case, we could not produce a precise implementation of the algorithm due to insufficient information in the article. Probably with more time on our hands, we could have attempted to solve the PDEs using alternative methods. However, we deemed it a deviation from the objectives of the report.

REFERENCES

[1] [n. d.]. Segmentation definition. ([n. d.]). https://en.wikipedia.org/wiki/Image_segmentation

[2] Andrés Anaya-Isaza, Leonel Mera-Jiménez, and Martha Zequera-Diaz. 2021. An overview of deep learning in medical imaging. *Informatics in medicine unlocked* 26 (2021), 100723.

[3] Tony F Chan and Luminita A Vese. 2001. Active contours without edges. *IEEE Transactions on image processing* 10, 2 (2001), 266–277.

[4] Luis Pedro Coelho, Aabid Shariff, and Robert Murphy. 2009. Nuclear Segmentation in Microscope Cell Images: A Hand-Segmented Dataset and Comparison of Algorithms. *Proceedings / IEEE International Symposium on Biomedical Imaging: from nano to macro. IEEE International Symposium on Biomedical Imaging* 5193098 (06 2009), 518–521. https://doi.org/10.1109/ISBI.2009.5193098

[5] Tom Goldstein, Xavier Bresson, and Stanley Osher. 2010. Geometric applications of the split Bregman method: segmentation and surface reconstruction. *Journal of scientific computing* 45 (2010), 272–293.

[6] Mohammad Hesam Hesamian, Wenjing Jia, Xiangjian He, and Paul Kennedy. 2019. Deep learning techniques for medical image segmentation: achievements and challenges. *Journal of digital imaging* 32 (2019), 582–596.

[7] Thouis Jones, Anne Carpenter, and Polina Golland. 2005. Voronoi-Based Segmentation of Cells on Image Manifolds. *Lect Notes Comput Sci* 3765, 535–543. https://doi.org/10.1007/11569541_54

[8] Dilpreet Kaur and Yadwinder Kaur. 2014. Various image segmentation techniques: a review. *International Journal of Computer Science and Mobile Computing* 3, 5 (2014), 809–814.

[9] Chunming Li, Chiu-Yen Kao, John C Gore, and Zhaohua Ding. 2008. Minimization of region-scalable fitting energy for image segmentation. *IEEE transactions on image processing* 17, 10 (2008), 1940–1949.

[10] Mikaël Rousson and Rachid Deriche. 2002. A variational framework for active and adaptative segmentation of vector valued images. In *Workshop on Motion and Video Computing, 2002. Proceedings.* IEEE, 56–61.

A APPENDIX

A.1 Datasets (T)

Two of the sets from [4] are Hoechst stained cells nuclei, for the first set there are 48 images with a size of 1349×1030 pixels, in total there are 1831 U20S Hoechst stained cell nuclei. Whereas for the second set, there are 49 images with size 1344 × 1024 pixels that contains 2178 NIH3T3[vi] Hoechst stained nuclei. One disadvantage of the second set is that the images contain intensity inhomogeneities and artifacts. The last set extracted from [4] is a set 7 images of of mouse neuroblastoma cells stained by N1E115 DAPI, each image has a size of 1392 × 1040 pixels and it can be found 389 cells nuclei, for this images the ground truth was generated by the authors.

The last set used in [7] is 16 images from a size between 400×400 and 512 × 512 pixels that includes 1280 Drosophila cells nuclei.

A.2 Definitions (L)

$$E_1(\Theta, \partial\Omega) = \lambda \left(\sum_{i=0}^{1} \kappa_i \int_{\Omega_i} (I(x) - \mu_i)^2 dx \right) + Per(\Omega_1) \tag{13}$$

with $\Theta = (\mu_0, \mu_1)$, and μ_0, (resp. μ_1) are the mean intensities of the background Ω_0 (resp. foreground Ω_1) regions and $\partial\Omega$ defines the boundaries between both regions. I(x) is the image intensity at position x and $Per(\Omega_1)$ is the perimeter, so, the regularization term. And finally, $\kappa_0, \kappa_1 and \lambda$ are weighting factors.

$$E_2(\Theta, \partial\Omega) = \lambda \left(\sum_{i=0}^{1} \int_{\Omega_i} -log(P(I(x)|\Omega_i)) dx \right) + Per(\Omega_1) \tag{14}$$

with $\Theta = (\mu_0, \mu_1, \sigma_0, \sigma_1)$, and μ_i, (resp. σ_i) are the mean intensities (resp. standard deviation) of the Ω_i regions. $P(I(x)|\Omega_i)$ is the conditional probability that pixel x, with image intensity I(x) belongs to region Ω_i.

$$E_3(\Theta, \partial\Omega) = \lambda \left(\sum_{i=0}^{1} \kappa_i \int_{\Omega_i} \int_{\Omega_1} K_\sigma(x-y) |I(y) - f_i(x)|^2 dy dx \right) + Per(\Omega_1) \tag{15}$$

with $\Theta = (f_0(x), f_1(x))$, and $f_i(x)$ represents a local approximation of the image intensity centered at x in Ω_i. And finally, K_σ is the gaussian kernel, with $\sigma > 0$ regulating the size of the region in which local intensities are approximated.

A.3 Metrics Formulas (T)

The Dice coefficient is defined as follows:

$$Dice(R, S) = \frac{2|R \cap S|}{|R| + |S|} \tag{16}$$

[vi] NIH: NIH Swiss mouse embryo & 3T3: **3**-day **t**ransfer, inoculum **3**×105 cells.

where R is the binary reference image, and S is the binary segmented image.

The normalized sum of distance(NSD) can be expressed as follows:

$$NSD(R,S) = \frac{\sum_{i \in R \cup S / R \cap S} D(i)}{\sum_{i \in R \cup S} D(i)} \tag{17}$$

where $D(i)$ is the minimal Euclidean distance of pixel i to the contour of the reference object. The Hausdorff distance is defined as:

$$h(R,S) = \max_{i \in S_c} \{D(i)\} \tag{18}$$

with S_c being the contour of the segmented object.

A.4 Figures

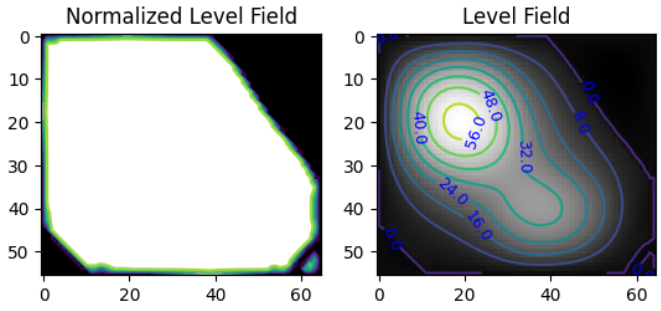


Figure 3: Level Set Obtained using Max Min normalization vs Without normalization

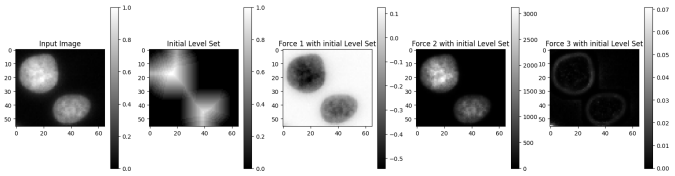


Figure 4: Forces images implementation

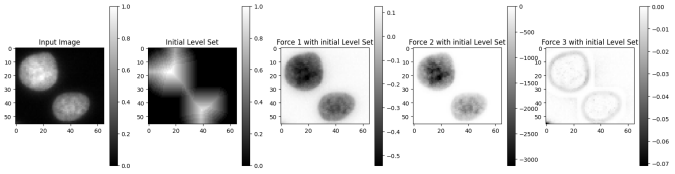


Figure 5: Forces images implementation Inverting Force 2 and Force 3

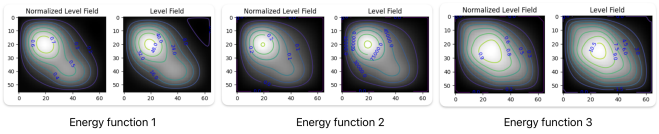


Figure 6: Normalization of each level field

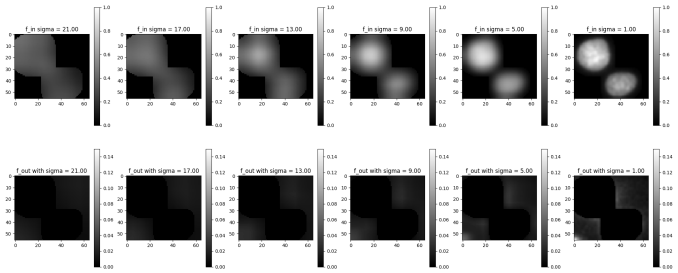


Figure 7: Effect of σ on $E_3(\Theta, \partial)$

Figure 8: Level Set obtained by using each energy function with Split Bregman.

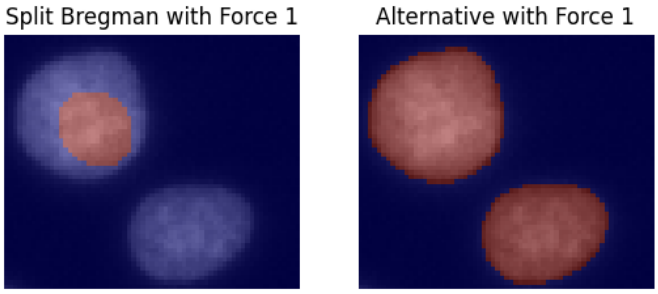


Figure 9: Level Set obtained using energy function 1 with Split Bregman and altenative.

A.5 Energy Evaluation Notebook

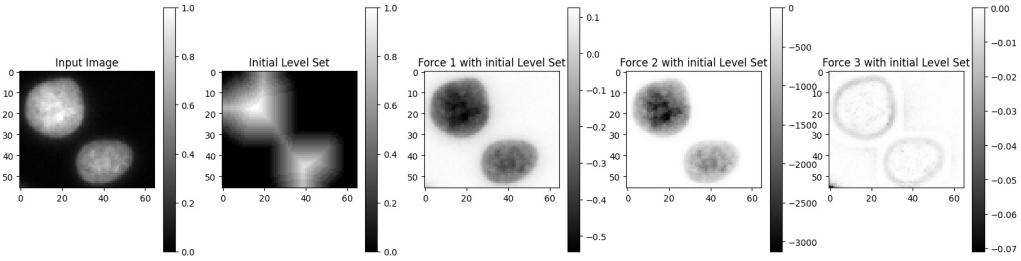
In [3]:
```
import matplotlib.pyplot as plt
import cv2 as cv
import numpy as np

from source.split_bregman_gcs import SplitBregmanGCS, NormalizationMode
from source.utils import normalization_automatic
from source.image_force import Force1, Force2, Force3
```

Evaluation Energy Functions

In [4]:
```
initial_level_set = cv.imread('test_images/simplify_cells_distance_multiply_10.tif', cv.CV_8U)
initial_level_set = normalization_automatic(initial_level_set)
image = cv.imread('test_images/simplify_cells.tif', cv.CV_16U)
image = normalization_automatic(image)
```

In [17]:
```
k0 = 1
k1 = 1
sigma = 1
force1 = Force1(image, k0=k0, k1=k1)
r1 = force1.compute_force(initial_level_set>0)
force2 = Force2(image)
r2 = force2.compute_force(initial_level_set>0)
force3 = Force3(image, k0=k0, k1=k1, sigma=sigma)
r3 = force3.compute_force(initial_level_set>0)
```

In [18]:
```
fig, ax = plt.subplots(1,5, figsize=(20,5))
ax[0].set_title('Input Image')
pos = ax[0].imshow(image, 'gray')
fig.colorbar(pos, ax=ax[0])

ax[1].set_title('Initial Level Set')
pos = ax[1].imshow(initial_level_set, 'gray')
fig.colorbar(pos, ax=ax[1])

ax[2].set_title('Force 1 with initial Level Set')
pos = ax[2].imshow(r1, 'gray')
fig.colorbar(pos, ax=ax[2])


ax[3].set_title('Force 2 with initial Level Set')
pos = ax[3].imshow(r2, 'gray')
fig.colorbar(pos, ax=ax[3])


ax[4].set_title('Force 3 with initial Level Set')
pos = ax[4].imshow(r3, 'gray')
fig.colorbar(pos, ax=ax[4])
plt.show()
```



Force 1

The force, denoted as r_1, is determined by the following mathematical expression:

$$r_1 = \kappa_1 (I(x) - \mu_1)^2 - \kappa_0 (I(x) - \mu_0)^2$$

Here, μ_1 represents the mean value of the region inside the contour, and μ_0 denotes the mean value of the region outside.

Force 2

Definition:

The force, denoted as r_2, is expressed as follows:

$$r_2 = log(P(I(x)\,|\,\Omega_1) - log(P(I(x)\,|\,\Omega_0)$$

where $P(I(x)\,|\,\Omega_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(I(x)-\mu_i)^2}{2\sigma_i^2}}$

For simplicity we rewrite the operation as:

$$r_2 = log\left(\frac{1}{\sqrt{2\pi\sigma_1^2}}\right) - \frac{(I(x)-\mu_1)^2}{2\sigma_1^2} - log\left(\frac{1}{\sqrt{2\pi\sigma_0^2}}\right) + \frac{(I(x)-\mu_0)^2}{2\sigma_0^2}$$

NOTE: Force 2 is inverted(line 54) to simulate the behavior of force 1.

Force 3

The formulation of this force is follows:

$$r_3 = \kappa_1 \int K_\sigma(y-x)\,|\,I(x) - f_1(y)\,|^2 dy - \kappa_0 \int K_\sigma(y-x)\,|\,I(x) - f_0(y)\,|^2 dy$$

where $K_\sigma(u) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{|u|^2}{2\sigma^2}}$ is a Gaussian Kernel, and the functions f_0 and f_1 are the inside and outside approximations of the image centered at each point, defined by:

$$f_0 = \frac{K_\sigma(x) * [(1 - H(\phi(x)))I(x)]}{K_\sigma(x) * [1 - H(\phi(x))]}$$
$$f_1 = \frac{K_\sigma(x) * [H(\phi(x))I(x)]}{K_\sigma(x) * H(\phi(x))}$$

For simplicity, in the code, we removed the use of Heaviside function by using masks.

Let's consider M_{in} the image mask that is one when we are inside the region, and M_{out} the image mask that is one when we are outside the region. Given these difiniations we can define: I_{in} and $Iout$ as their respective masked images.

We can rewrite the equations $f_{in}(f_1)$ and $f_{out}(f_0)$ as:

$$f_{in} = \frac{K_\sigma(x) * I_{in}(x)}{K_\sigma(x) * M_{in}(x)}$$
$$f_{out} = \frac{K_\sigma(x) * I_{out}(x)}{K_\sigma(x) * M_{out}(x)}$$

The convolution on the denominator will be always its corresponding mask, because M_{in} and M_{out} are arrays with only 0 and 1. Then, we can rewrite:

$$f_{in} = K_\sigma(x) * I_{in}(x)$$
$$f_{out} = K_\sigma(x) * I_{out}(x)$$

Then the force could be define as:

$$r_3(x) = \kappa_1 \sum_{y \in interior} K_\sigma(y-x)\,|\,I(x) - f_{in}(y)\,|^2 - \kappa_0 \sum_{y \in outside} K_\sigma(y-x)\,|\,I(x) - f_{out}(y)\,|^2$$
$$r_3(x) = \kappa_1 \sum_{y \in interior} K_\sigma(y-x)[I^2(x) - 2I(x)f_{in}(y) + f_{in}^2(y)] - \kappa_0 \sum_{y \in outside} K_\sigma(y-x)[I^2(x) - 2I(x)f_{out}(y) + f_{out}^2(y)]$$
$$r_3(x) = (\kappa_1 - \kappa_0)\sum_{y \in image} K_\sigma(y-x)I^2(x) - 2\kappa_1 I(x)\sum_{y \in inside} K_\sigma(y-x)f_{in}(y) + 2\kappa_0 I(x)\sum_{y \in outside} K_\sigma(y-x)f_{out}(y) + \kappa_1 \sum_{y \in inside} K_\sigma(y-x)f_{in}^2(y) - \kappa_0 \sum_{y \in outside} K_\sigma(y-x)f$$
$$r_3(x) = (\kappa_1 - \kappa_0)(K_\sigma * I^2)(x) - 2\kappa_1 I(x)[(K_\sigma * f_{in})(x)] + 2\kappa_0 I(x)[(K_\sigma * f_{out})(x)] + \kappa_1 (K_\sigma * f_{in}^2)(x) - \kappa_0 (K_\sigma * f_{out}^2)(x)$$

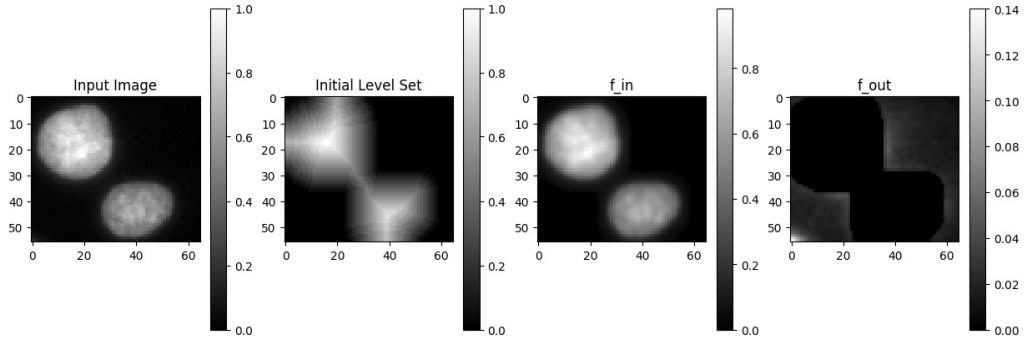
In the usual definition of Gaussian Kernel, we have two parameters `kernel_size` and `sigma`. However, in the article, the authors consider only sigma and they afirm that sigma control the size of the region in which the image intensities are approximated. Taking this into consideration, in our implementation, we decide that `kernel_size = (2*sigma+1, 2*sigma+1)`, to obtain a valid kernel size for every sigma (it has to be an integer).

Local Function Approximation

In [15]:
```
f_in = force3.f_in()
f_out = force3.f_out()
fig, ax = plt.subplots(1,4, figsize=(15,5))
```

```
ax[0].set_title('Input Image')
pos = ax[0].imshow(image, 'gray')
fig.colorbar(pos, ax=ax[0])

ax[1].set_title('Initial Level Set')
pos = ax[1].imshow(initial_level_set, 'gray')
fig.colorbar(pos, ax=ax[1])

ax[2].set_title('f_in')
pos = ax[2].imshow(f_in, 'gray')
fig.colorbar(pos, ax=ax[2])

ax[3].set_title('f_out')
pos = ax[3].imshow(f_out, 'gray')
fig.colorbar(pos, ax=ax[3])
```

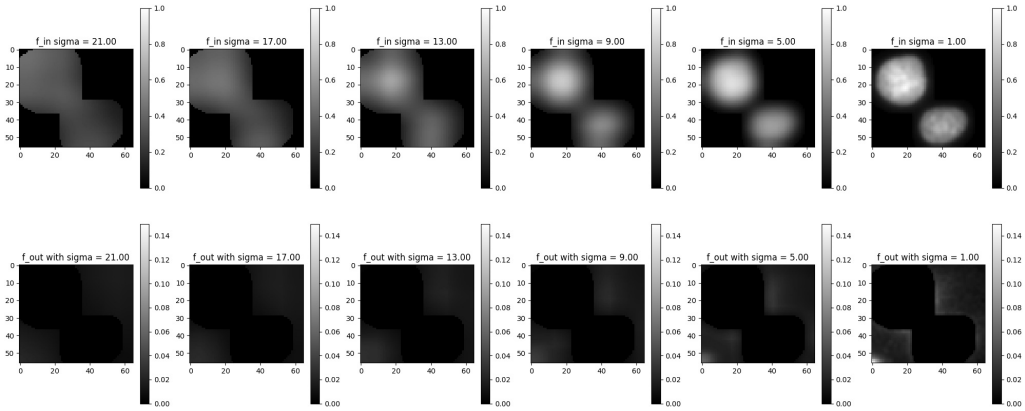
Out[15]: <matplotlib.colorbar.Colorbar at 0x1a27ecd1610>



Observation

Visually the aproximation for the interior seems to be coherent. However, the approximation for the outside of the level set has a little artifact at the left corner that is probably an border artifact.

In [13]:
```
sigma_list = np.arange(1, 25, 4)
vmin_in, vmax_r3 = 0, 1
vmin_out, vmax_out = 0, .15
N = len(sigma_list)
fig, ax = plt.subplots(2,N, figsize=(5*(N-1),10))
for i, sigma in enumerate(sigma_list[::-1]):
    force3 = Force3(image, k0=k0, k1=k1, sigma=sigma)
    _ = force3.compute_force(initial_level_set>0)
    f_in = force3.f_in()
    f_out = force3.f_out()

    ax[0, i].set_title(f'f_in sigma = {sigma:.2f}')
    pos = ax[0, i].imshow(f_in, 'gray', vmin=vmin_in, vmax=vmax_r3)
    fig.colorbar(pos, ax=ax[0, i])

    ax[1, i].set_title(f'f_out with sigma = {sigma:.2f}')
    pos = ax[1, i].imshow(f_out, 'gray', vmin=vmin_out, vmax=vmax_out)
    fig.colorbar(pos, ax=ax[1, i])

    if np.min(f_in) < vmin_in or np.max(f_in) > vmax_r3:
        print(f'Visualization error!! The selected vmin and vmax for the region inside are not valid')
    if np.min(f_out) < vmin_out or np.max(f_out) > vmax_r3:
        print(f'Visualization error!! The selected vmin and vmax for the region outside are not valid')
```



As we expected, when we lower the parameter σ, our functions f_{in} and f_{out} represent a more local approximation. Thus, when we decrease the σ parameter, the sum of $f_{in} + f_{out}$ approximates I.

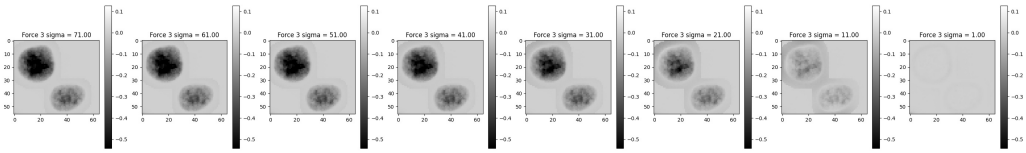
Evolution of the force

If we consider $k_1 = k_2$ as they do in the paper. Using convolution properties we can rewrite the force like:

$$r_3(x) = -2\kappa I(x)[(K_\sigma * f_{in})(x)] + 2\kappa I(x)[(K_\sigma * f_{out})(x)] + \kappa(K_\sigma * f_{in}^2)(x) - \kappa(K_\sigma * f_{out}^2)(x)$$
$$r_3(x) = -2\kappa I(x)[(K_\sigma * (f_{in} - f_{out})(x)] + \kappa(K_\sigma * (f_{in}^2 + f_{out}^2))(x)$$

NOTE: Force 3 is inverted(line 153) to simulate the behavior of force 1.

In [16]:
```
sigma_list = np.arange(1, 80, 10)
N = len(sigma_list)
vmin_in, vmax_r3 = np.min(r1), np.max(r1)
fig, ax = plt.subplots(1,N, figsize=(5*(N-1),5))
for i, sigma in enumerate(sigma_list[::-1]):
    force3 = Force3(image, k0=k0, k1=k1, sigma=sigma)
    r3 = force3.compute_force(initial_level_set>0)
    ax[i].set_title(f'Force 3 sigma = {sigma:.2f}')
    pos = ax[i].imshow(r3, 'gray', vmin=vmin_in, vmax=vmax_r3)
    fig.colorbar(pos, ax=ax[i])
```



Observation

In our current case, the region outside is practically zero, so ONLY in this example $f_{in} - f_{out} \approx I$ for lower values of σ, which means we are getting values similar to zero in our force.

The Split Bregman Method Applied to Globally Convex Segmentation

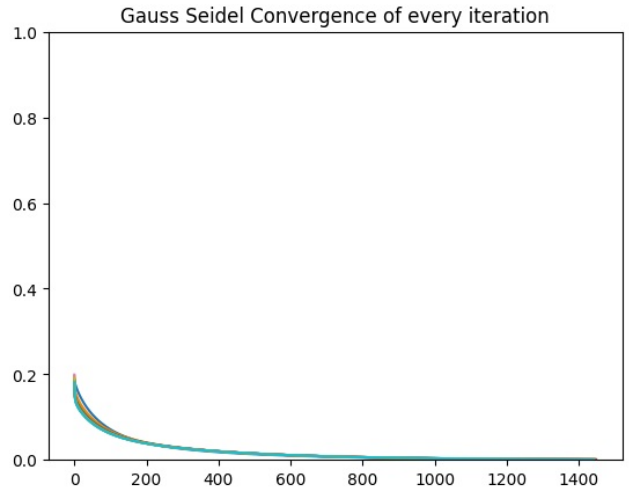
In [19]:
```
# Default parameters for method
lambda_value = 1
nu_value = 0.5
epsilon_value=0.1
gs_error=1e-3
```

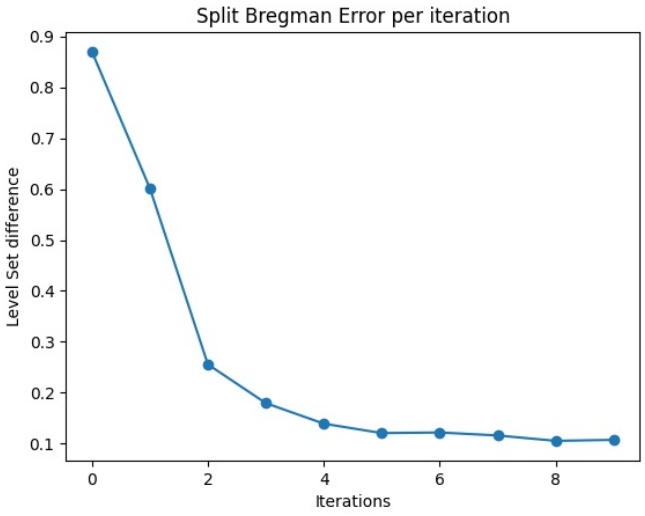
Using force 1

In [21]:
```
force1 = Force1(image, k1=k1, k0=k0)
segmentator = SplitBregmanGCS(
    force1,
    lambda_value=lambda_value,
    nu_value=nu_value,
```

```
    epsilon_value=epsilon_value,
    gs_error=gs_error,
    mode=NormalizationMode.Clip,
    debug=True)

last_level_set, last_level_set_no_normalized = segmentator.run(initial_level_set)
```

```
---------- Iteration error 0.8700236246608009 ----------
Gauss Seidel Iteration:   0%|          | 0/10000 [00:00<?, ?it/s]Gauss Seidel Iteration:  14%|        | 1402/10000 [00:10<01:04, 133.51it/s]
The solution converged after 1402 iterations
---------- Iteration error 0.6012317998865605 ----------
Gauss Seidel Iteration:  14%|        | 1446/10000 [00:11<01:07, 127.53it/s]
The solution converged after 1446 iterations
---------- Iteration error 0.25548236461147417 ----------
Gauss Seidel Iteration:  14%|        | 1449/10000 [00:10<01:04, 131.76it/s]
The solution converged after 1449 iterations
---------- Iteration error 0.17934898814053488 ----------
Gauss Seidel Iteration:  14%|        | 1447/10000 [00:10<01:04, 132.26it/s]
The solution converged after 1447 iterations
---------- Iteration error 0.13902397334859312 ----------
Gauss Seidel Iteration:  14%|        | 1446/10000 [00:10<01:00, 142.04it/s]
The solution converged after 1446 iterations
---------- Iteration error 0.1205376982865722 ----------
Gauss Seidel Iteration:  14%|        | 1444/10000 [00:10<01:04, 132.74it/s]
The solution converged after 1444 iterations
---------- Iteration error 0.12171601749139245 ----------
Gauss Seidel Iteration:  14%|        | 1443/10000 [00:11<01:07, 127.23it/s]
The solution converged after 1443 iterations
---------- Iteration error 0.11585421578858164 ----------
Gauss Seidel Iteration:  14%|        | 1443/10000 [00:10<00:59, 142.68it/s]
The solution converged after 1443 iterations
---------- Iteration error 0.10533201020569914 ----------
Gauss Seidel Iteration:  14%|        | 1442/10000 [00:11<01:07, 126.83it/s]
The solution converged after 1442 iterations
---------- Iteration error 0.10720064105625197 ----------
Gauss Seidel Iteration:  14%|        | 1441/10000 [00:11<01:05, 130.57it/s]
The solution converged after 1441 iterations
Converged with an error 0.09124782974391564
```



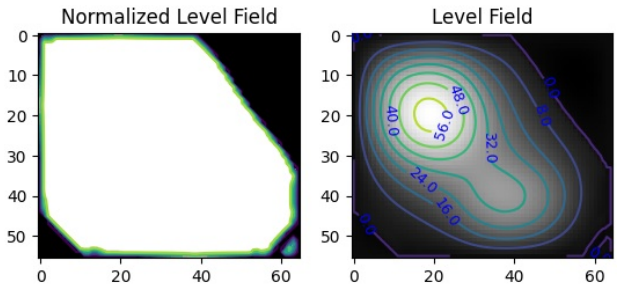


Observation

Looking at the error graphs we can say that the algorithm coverge, so we should be able to have a good segmentation

In [25]:
```
fig, ax = plt.subplots(1,2)
ax[0].set_title('Normalized Level Field')
pos = ax[0].imshow(last_level_set, 'gray')
ax[0].contour(last_level_set)
ax[1].set_title('Level Field')
ax[1].imshow(last_level_set_no_normalized, 'gray')
cs = ax[1].contour(last_level_set_no_normalized)
ax[1].clabel(cs, fmt='%2.1f', colors='blue', fontsize=9)
```

Out[25]:
```
<a list of 11 text.Text objects>
```



Observation

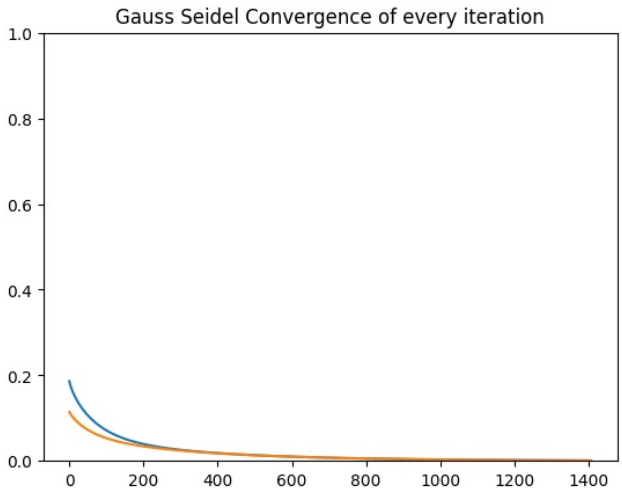
In the article the normalization of the level set is not specified. First we followed The suggestion in *T.Goldstein, X.Bresson, and S.Osher. Geometric applications of the split Bregman method*, where the level field is clipped between 0 and 1. Clearly this method is not good, because we lose important information, in the graph above we can see that without normalization we can see coherent diferent level sets but all their values are greater than 1. In the code this normalization mode is: `NormalizationMode.Clip`

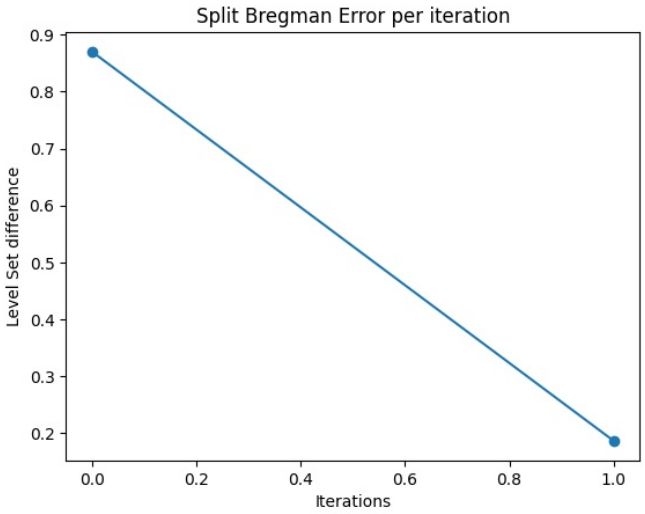
Alternative

Then, we modify the normalization by clipping all the level set using the min and max value of the first level set(`NormalizationMode.FirstImageParameters`).

In [26]:
```
force1 = Force1(image, k1=k0, k0=k1)
segmentator = SplitBregmanGCS(
    force1,
    lambda_value=lambda_value,
    nu_value=nu_value,
    epsilon_value=epsilon_value,
    gs_error=gs_error,
    mode=NormalizationMode.FirstImageParameters,
    debug=True)

last_level_set, last_level_set_no_normalized = segmentator.run(initial_level_set)
```

```
---------------- Iteration error 0.8700236246608009 ------------------------
Gauss Seidel Iteration:   0%|          | 0/10000 [00:00<?, ?it/s]Gauss Seidel Iteration:  14%|          | 1402/1
0000 [00:09<01:00, 143.22it/s]
The solution converged after 1402 iterations
---------------- Iteration error 0.18679710819589943 ------------------------
Gauss Seidel Iteration:  14%|          | 1408/10000 [00:10<01:06, 128.29it/s]
The solution converged after 1408 iterations
Converged with an error 0.0911956154497904
```

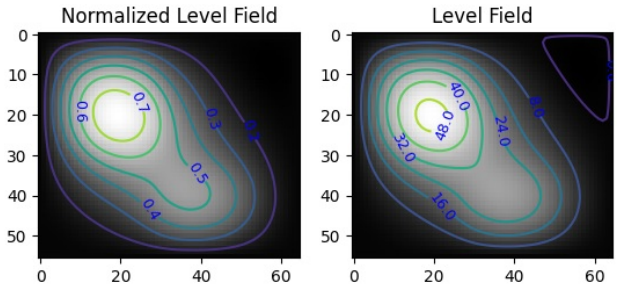




In [27]:
```
fig, ax = plt.subplots(1,2)
ax[0].set_title('Normalized Level Field')
ax[0].imshow(last_level_set, 'gray')
```

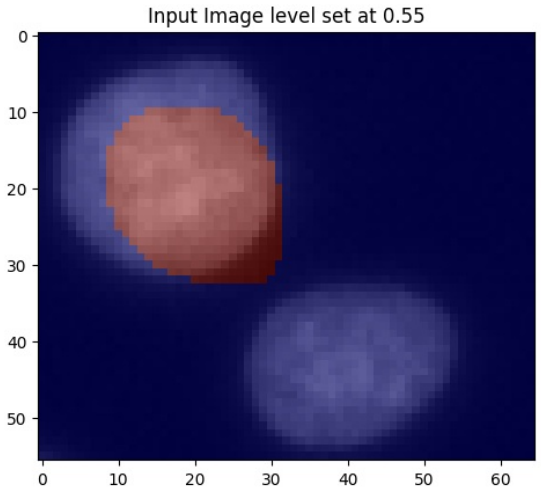
```
cs = ax[0].contour(last_level_set)
ax[0].clabel(cs, fmt='%2.1f', colors='blue', fontsize=9)
ax[1].set_title('Level Field')
ax[1].imshow(last_level_set_no_normalized, 'gray')
cs = ax[1].contour(last_level_set_no_normalized)
ax[1].clabel(cs, fmt='%2.1f', colors='blue', fontsize=9)
```

Out[27]: <a list of 7 text.Text objects>



In [33]:
```
alpha = 0.55
level_set_at_alpha = last_level_set > alpha
fig, ax = plt.subplots(1,1)
ax.set_title(f'Input Image level set at {alpha}')


ax.imshow(image, cmap='gray')
ax.imshow(level_set_at_alpha, 'jet', interpolation='none', alpha=0.5)
```

Out[33]: <matplotlib.image.AxesImage at 0x1a2815ca210>

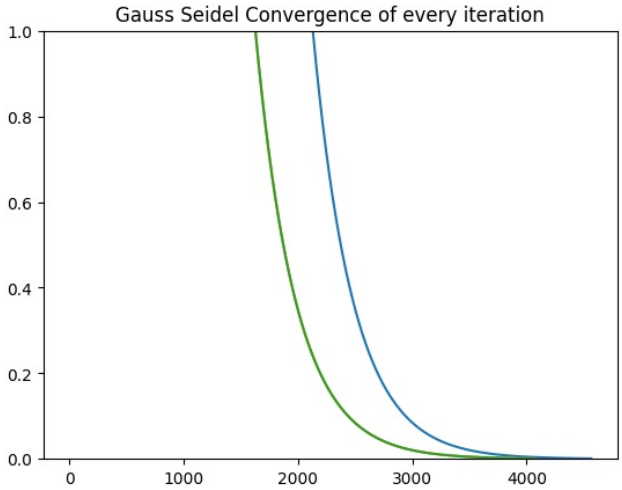


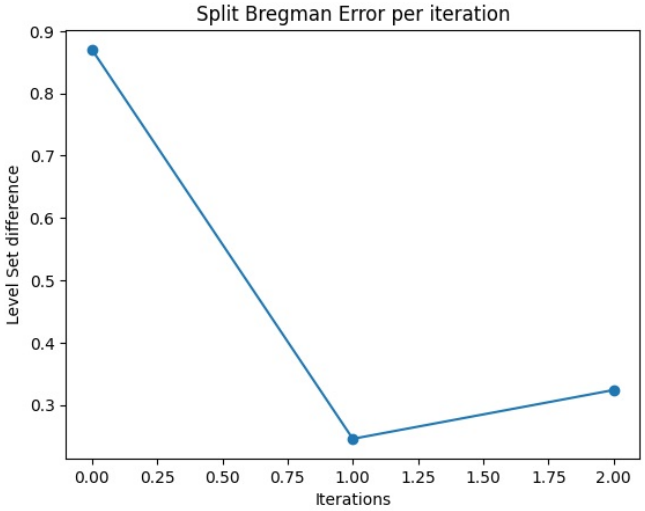
Using Energy 2

In [34]:
```
force2 = Force2(image)
segmentator = SplitBregmanGCS(
    force2,
    lambda_value=lambda_value,
    nu_value=nu_value,
    epsilon_value=epsilon_value,
    gs_error=gs_error,
    mode=NormalizationMode.FirstImageParameters,
    debug=True)

last_level_set, last_level_set_no_normalized = segmentator.run(initial_level_set)
```

```
---------- Iteration error 0.8700236246608009 ----------
Gauss Seidel Iteration:   0%|          | 0/10000 [00:00<?, ?it/s]Gauss Seidel Iteration:  46%|████      | 4566/10000 [00:33<00:40, 135.62it/s]
The solution converged after 4566 iterations
---------- Iteration error 0.24554837801444837 ----------
Gauss Seidel Iteration:  41%|███       | 4064/10000 [00:31<00:46, 128.94it/s]
```

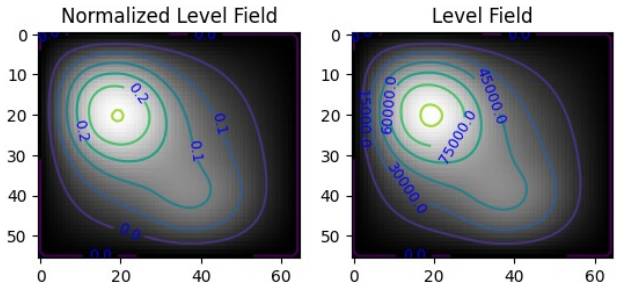
```
The solution converged after 4064 iterations
---------------- Iteration error 0.3239076957516847 ------------------------
Gauss Seidel Iteration:  41%|████      | 4064/10000 [00:29<00:43, 137.37it/s]
The solution converged after 4064 iterations
Converged with an error 8.436565703678474e-07
```





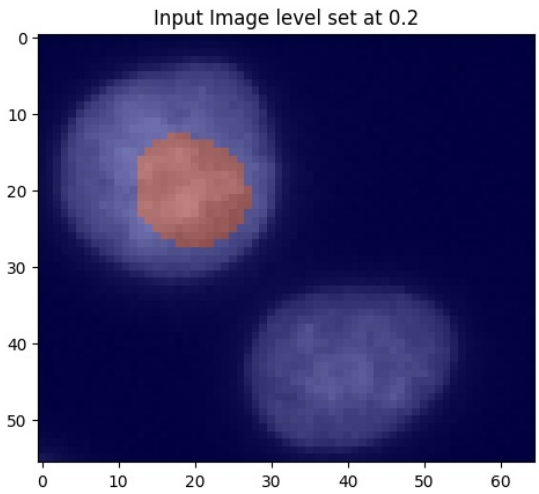
In [35]:
```
fig, ax = plt.subplots(1,2)
ax[0].set_title('Normalized Level Field')
ax[0].imshow(last_level_set, 'gray')
cs = ax[0].contour(last_level_set)
ax[0].clabel(cs, fmt='%2.1f', colors='blue', fontsize=9)
ax[1].set_title('Level Field')
ax[1].imshow(last_level_set_no_normalized, 'gray')
cs = ax[1].contour(last_level_set_no_normalized)
ax[1].clabel(cs, fmt='%2.1f', colors='blue', fontsize=9)
```

Out[35]:
```
<a list of 8 text.Text objects>
```



In [36]:
```
alpha = 0.2
level_set_at_alpha = last_level_set > alpha
fig, ax = plt.subplots(1,1)
ax.set_title(f'Input Image level set at {alpha}')


ax.imshow(image, cmap='gray')
ax.imshow(level_set_at_alpha, 'jet', interpolation='none', alpha=0.5)
```

Out[36]: <matplotlib.image.AxesImage at 0x1a27be8bb50>



Observation

In this example we don't see un improvement over using energy 3. But for this case we had to lower the threshold α.

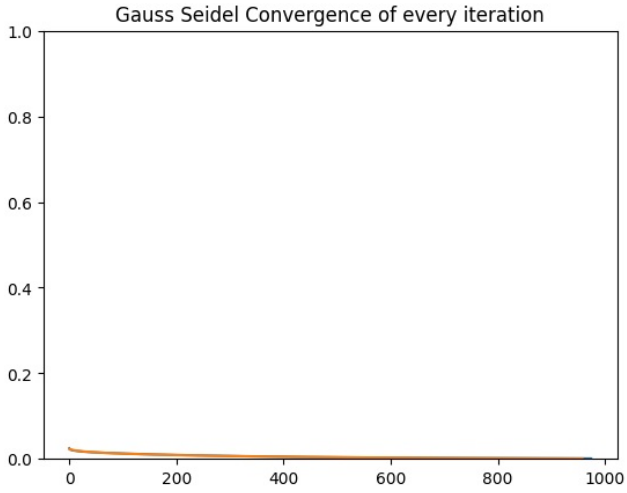
Using Energy 3

In [29]:
```
sigma = 5
k0 = 1
k1= 1
```

In [30]:
```
force3 = Force3(image, k0=k0, k1=k1, sigma=sigma)
segmentator = SplitBregmanGCS(
    force3,
    lambda_value=lambda_value,
```

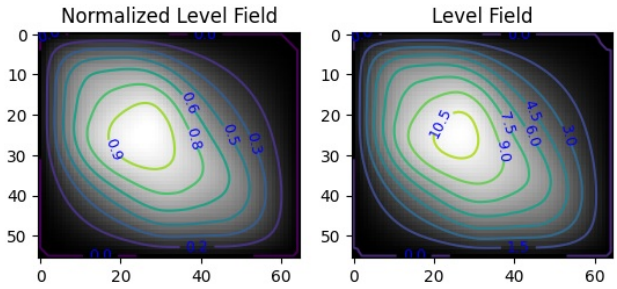
```
    nu_value=nu_value,
    epsilon_value=epsilon_value,
    gs_error=gs_error,
    mode=NormalizationMode.FirstImageParameters,
    debug=True)

last_level_set, last_level_set_no_normalized = segmentator.run(initial_level_set)
```

```
----------------- Iteration error 0.8700236246608009 ------------------------
Gauss Seidel Iteration:   0%|          | 17/10000 [00:00<00:59, 168.77it/s]Gauss Seidel Iteration:  10%|
| 975/10000 [00:04<00:37, 237.68it/s]
The solution converged after 975 iterations
----------------- Iteration error 0.33144708330427264 ------------------------
Gauss Seidel Iteration:  10%|         | 959/10000 [00:04<00:39, 227.36it/s]
The solution converged after 959 iterations
Converged with an error 0.030406511799383272
```



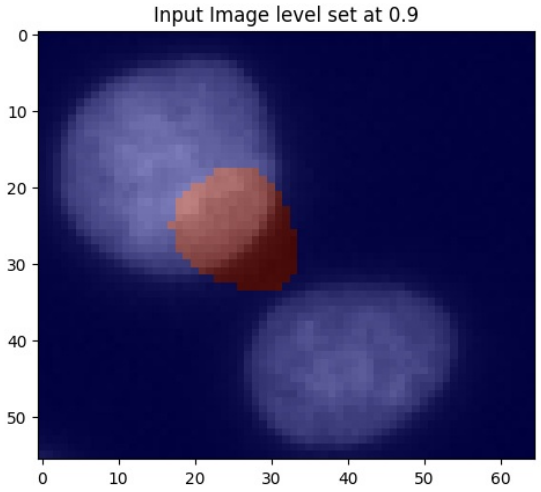
In [31]:
```
fig, ax = plt.subplots(1,2)
ax[0].set_title('Normalized Level Field')
ax[0].imshow(last_level_set, 'gray')
cs = ax[0].contour(last_level_set)
ax[0].clabel(cs, fmt='%2.1f', colors='blue', fontsize=9)
ax[1].set_title('Level Field')
ax[1].imshow(last_level_set_no_normalized, 'gray')
cs = ax[1].contour(last_level_set_no_normalized)
ax[1].clabel(cs, fmt='%2.1f', colors='blue', fontsize=9)
```

Out[31]:
```
<a list of 10 text.Text objects>
```



In [32]:
```
alpha = 0.9
level_set_at_alpha = last_level_set > alpha
fig, ax = plt.subplots(1,1)
ax.set_title(f'Input Image level set at {alpha}')


ax.imshow(image, cmap='gray')
ax.imshow(level_set_at_alpha, 'jet', interpolation='none', alpha=0.5)
```

Out[32]: <matplotlib.image.AxesImage at 0x2db70aac2d0>



Observation

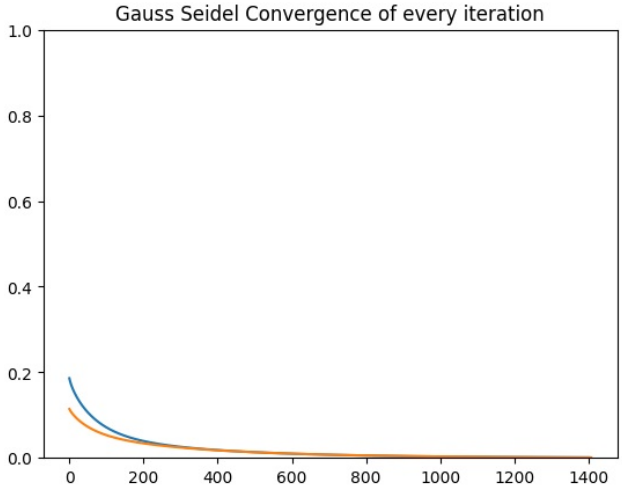
In all three cases, the cell segmentation is not functioning properly. Each energy should be capable of individually segmenting the cells. It is evident that there is an error in the implementation of the energy definition or convergence algorithm.

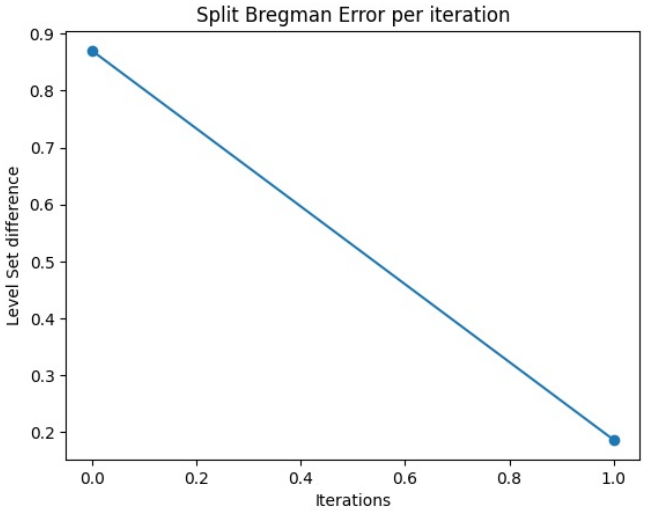
In [41]:
```
sigma = 5
k0 = 1
k1= 1
# Default parameters for method
lambda_value = 1
nu_value = 0.5
epsilon_value=0.1
gs_error=1e-3
```

In [42]:
```
segmentator = SplitBregmanGCS(
    force1,
    lambda_value=lambda_value,
    nu_value=nu_value,
    epsilon_value=epsilon_value,
    gs_error=gs_error,
    mode=NormalizationMode.FirstImageParameters,
    debug=True)

last_level_set_1, _ = segmentator.run(initial_level_set)
```

```
----------------- Iteration error 0.8700236246608009 ------------------------
```

```
Gauss Seidel Iteration:   0%|          | 5/10000 [00:00<03:45, 44.37it/s]Gauss Seidel Iteration:  14%|
| 1402/10000 [00:30<03:07, 45.96it/s]
The solution converged after 1402 iterations
---------- Iteration error 0.18679710819589943 ----------
Gauss Seidel Iteration:  14%|         | 1408/10000 [00:33<03:25, 41.85it/s]
The solution converged after 1408 iterations
Converged with an error 0.0911956154497904
```

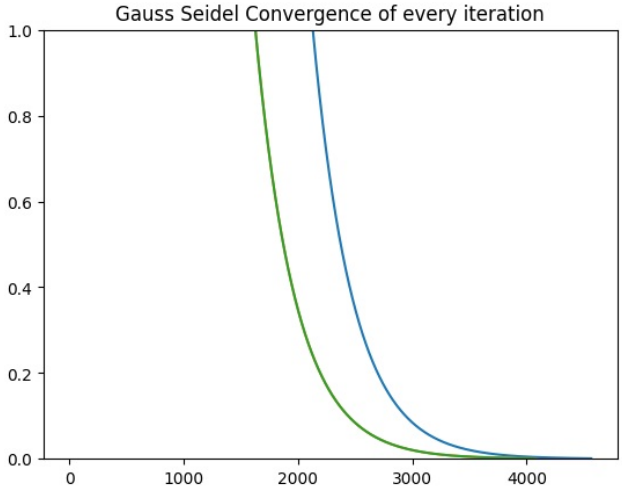




In [43]:
```
segmentator = SplitBregmanGCS(
    force2,
    lambda_value=lambda_value,
    nu_value=nu_value,
    epsilon_value=epsilon_value,
    gs_error=gs_error,
    mode=NormalizationMode.FirstImageParameters,
    debug=True)

last_level_set_2, _ = segmentator.run(initial_level_set)
```

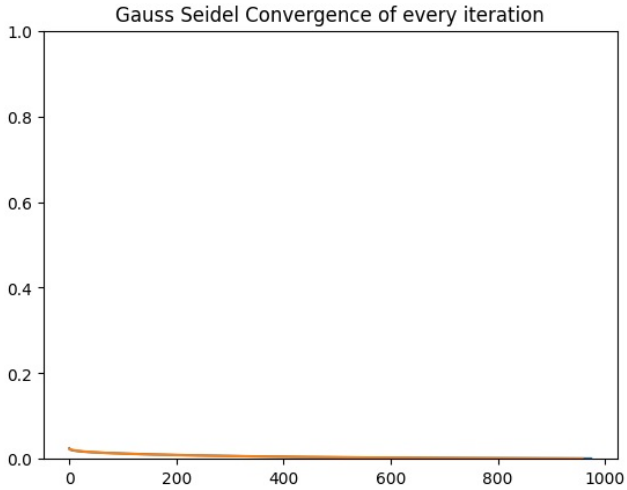
```
---------- Iteration error 0.8700236246608009 ----------
Gauss Seidel Iteration:   0%|          | 5/10000 [00:00<03:43, 44.76it/s]Gauss Seidel Iteration:  46%|
| 4566/10000 [01:17<01:32, 58.57it/s]
The solution converged after 4566 iterations
---------- Iteration error 0.24554837801444837 ----------
Gauss Seidel Iteration:  41%|      | 4064/10000 [01:20<01:57, 50.55it/s]
The solution converged after 4064 iterations
---------- Iteration error 0.3239076957516847 ----------
Gauss Seidel Iteration:  41%|      | 4064/10000 [01:20<01:56, 50.74it/s]
```

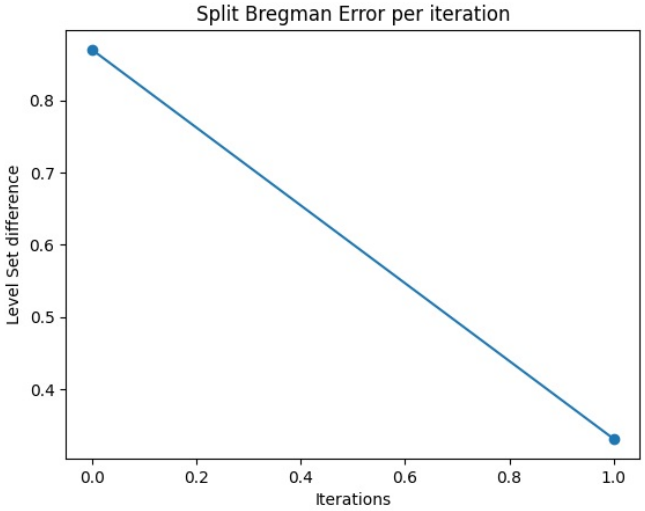
```
The solution converged after 4064 iterations
Converged with an error 8.436565703678474e-07
```



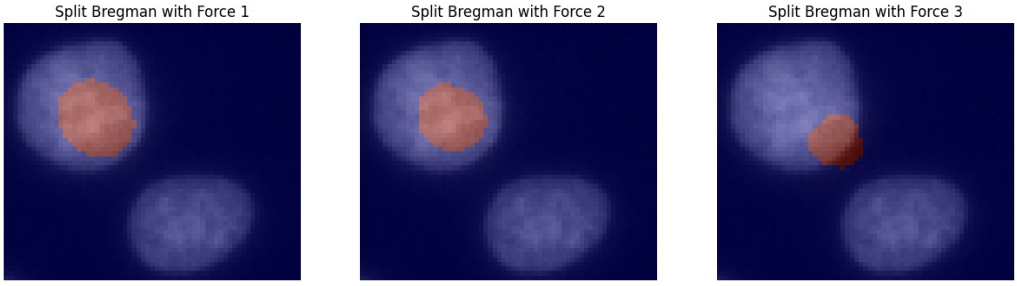
In [45]:
```
force3 = Force3(image, k0=k0, k1=k1, sigma=sigma)
segmentator = SplitBregmanGCS(
    force3,
    lambda_value=lambda_value,
    nu_value=nu_value,
    epsilon_value=epsilon_value,
    gs_error=gs_error,
    mode=NormalizationMode.FirstImageParameters,
    debug=True)

last_level_set_3, _ = segmentator.run(initial_level_set)
```

```
----------------- Iteration error 0.8700236246608009 ------------------------
Gauss Seidel Iteration:   0%|          | 9/10000 [00:00<06:15, 26.62it/s]Gauss Seidel Iteration:  10%|
| 975/10000 [00:27<04:12, 35.72it/s]
The solution converged after 975 iterations
----------------- Iteration error 0.33144708330427264 ------------------------
Gauss Seidel Iteration:  10%|          | 959/10000 [00:23<03:42, 40.67it/s]
The solution converged after 959 iterations
Converged with an error 0.030406511799383272
```





In [53]:
```
alpha_1 = 0.65
alpha_2 = 0.2
alpha_3 = 0.95
fig, ax = plt.subplots(1,3, figsize=(15,8))

ax[0].set_title(f'Split Bregman with Force 1')
ax[0].imshow(image, cmap='gray')
ax[0].imshow(last_level_set_1 > alpha_1, 'jet', interpolation='none', alpha=0.5)
ax[0].set_axis_off()

ax[1].set_title('Split Bregman with Force 2')
ax[1].imshow(image, cmap='gray')
ax[1].imshow(last_level_set_2 > alpha_2, 'jet', interpolation='none', alpha=0.5)
ax[1].set_axis_off()

ax[2].set_title('Split Bregman with Force 3')
ax[2].imshow(image, cmap='gray')
ax[2].imshow(last_level_set_3 > alpha_3, 'jet', interpolation='none', alpha=0.5)
ax[2].set_axis_off()
```



A.6 Alternative Implementation Notebook

In [1]:
```
import matplotlib.pyplot as plt
import cv2 as cv
import numpy as np

from source.split_bregman_gcs import SplitBregmanGCS, NormalizationMode
from source.utils import normalization_automatic
from source.image_force import Force1, Force2, Force3
```

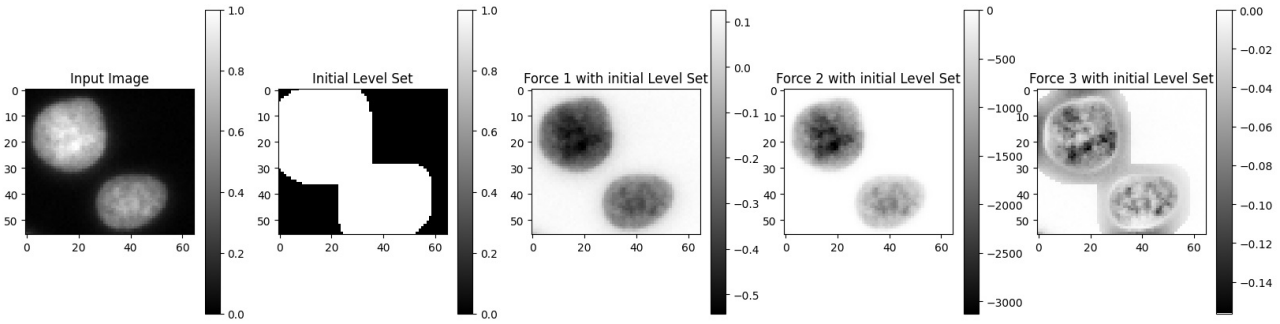
Evaluation Energy Functions

In [2]:
```
initial_level_set = cv.imread('test_images/simplify_cells_initial_mask_2.png', cv.CV_8U)
initial_level_set = normalization_automatic(initial_level_set)
image = cv.imread('test_images/simplify_cells.tif', cv.CV_16U)
image = normalization_automatic(image)
```

In [20]:
```
k0 = 1
k1 = 1
sigma = 8
force1 = Force1(image, k0=k0, k1=k1)
r1 = force1.compute_force(initial_level_set>0)
force2 = Force2(image)
r2 = force2.compute_force(initial_level_set>0)
force3 = Force3(image, k0=k0, k1=k1, sigma=sigma)
r3 = force3.compute_force(initial_level_set>0)
```

In [ ]:
```
```

In [21]:
```
fig, ax = plt.subplots(1,5, figsize=(20,5))
ax[0].set_title('Input Image')
pos = ax[0].imshow(image, 'gray')
fig.colorbar(pos, ax=ax[0])

ax[1].set_title('Initial Level Set')
pos = ax[1].imshow(initial_level_set, 'gray')
fig.colorbar(pos, ax=ax[1])

ax[2].set_title('Force 1 with initial Level Set')
pos = ax[2].imshow(r1, 'gray')
fig.colorbar(pos, ax=ax[2])


ax[3].set_title('Force 2 with initial Level Set')
pos = ax[3].imshow(r2, 'gray')
fig.colorbar(pos, ax=ax[3])


ax[4].set_title('Force 3 with initial Level Set')
pos = ax[4].imshow(r3, 'gray')
fig.colorbar(pos, ax=ax[4])
plt.show()
```



Alternative PDE Solution

In [22]:
```
# Default parameters for method
lambda_value = 1
nu_value = 0.5
epsilon_value=0.1
gs_error=1e-3
```


Checking with every force indepently

In [23]:
```
from source.pde_solver import PDESolver
segmentator_1 = PDESolver(force1, lambda_value, epsilon_value)
last_level_set_1 = segmentator_1.run(initial_level_set)
```

```
----------------- Iteration error 0.7019015716797695 ------------------------
Convergence with 0.7019015716797695
```

In [24]:
```
segmentator_2 = PDESolver(force2, lambda_value, epsilon_value)
last_level_set_2 = segmentator_2.run(initial_level_set)
```

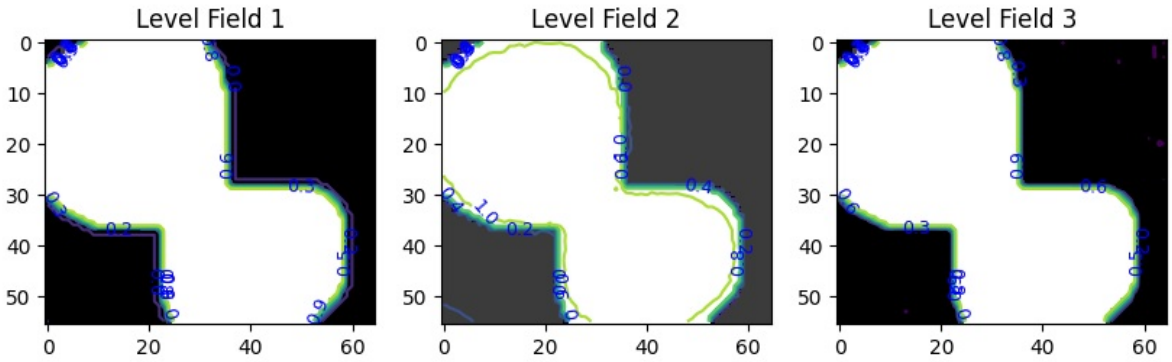
```
----------------- Iteration error 0.7019015716797695 ------------------------
Convergence with 0.7019015716797695
```

In [25]:
```
segmentator_3 = PDESolver(force3, lambda_value, epsilon_value)
last_level_set_3 = segmentator_3.run(initial_level_set)
```

```
----------------- Iteration error 0.7019015716797695 ------------------------
Convergence with 0.7019015716797695
```

In [26]:
```
fig, ax = plt.subplots(1,3, figsize=(10,5))
ax[0].set_title('Level Field 1')
ax[0].imshow(last_level_set_1, 'gray')
cs = ax[0].contour(last_level_set_1)
ax[0].clabel(cs, fmt='%2.1f', colors='blue', fontsize=9)

ax[1].set_title('Level Field 2')
ax[1].imshow(last_level_set_2, 'gray')
cs = ax[1].contour(last_level_set_2)
ax[1].clabel(cs, fmt='%2.1f', colors='blue', fontsize=9)

ax[2].set_title('Level Field 3')
ax[2].imshow(last_level_set_3, 'gray')
cs = ax[2].contour(last_level_set_3)
ax[2].clabel(cs, fmt='%2.1f', colors='blue', fontsize=9)
```

Out[26]:
```
<a list of 18 text.Text objects>
```

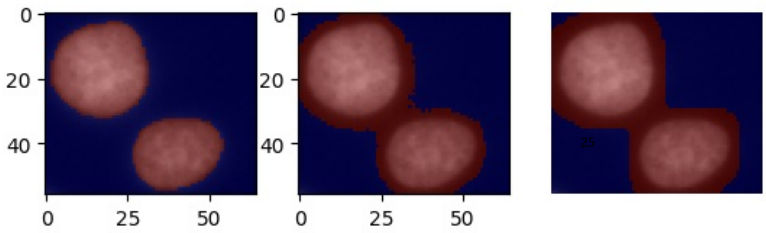


Observation

Now we are able to do a correct segmentation as expected.

In [33]:
```
alpha = 1
#level_set_at_alpha = last_level_set > alpha
fig, ax = plt.subplots(1,3)

ax[0].imshow(image, cmap='gray')
ax[0].imshow(last_level_set_1 > alpha, 'jet', interpolation='none', alpha=0.5)

ax[1].imshow(image, cmap='gray')
ax[1].imshow(last_level_set_2 > alpha, 'jet', interpolation='none', alpha=0.5)

ax[2].imshow(image, cmap='gray')
ax[2].imshow(last_level_set_3 > alpha, 'jet', interpolation='none', alpha=0.5)
```

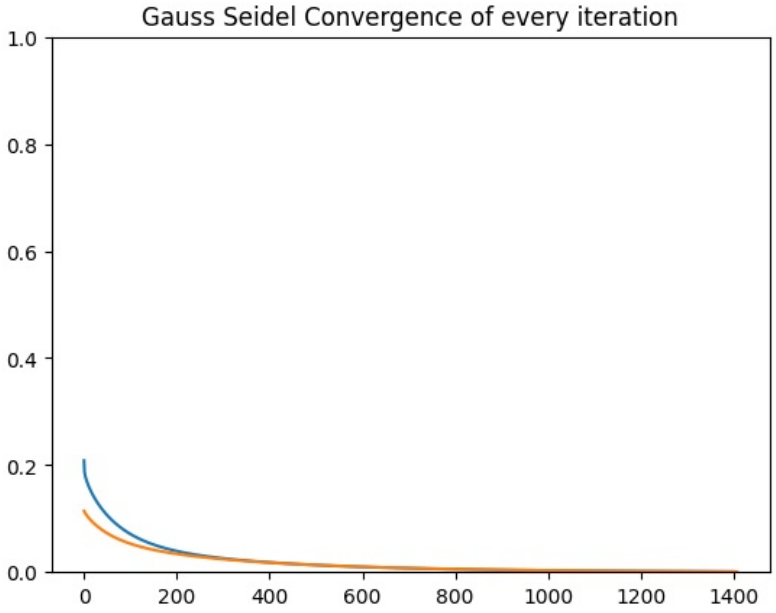
Out[33]:
```
<matplotlib.image.AxesImage at 0x21f8a47f1d0>
```



In [38]:
```
segmentator = SplitBregmanGCS(
```

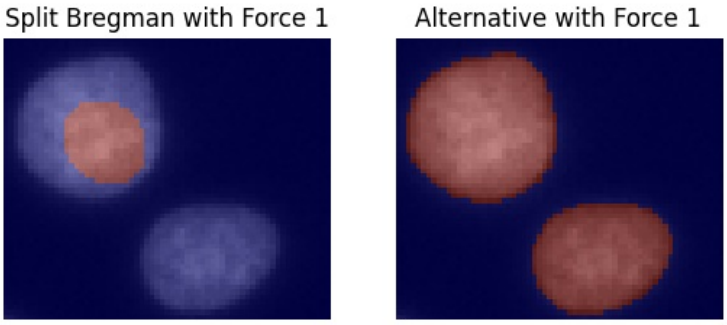
```
    force1,
    lambda_value=lambda_value,
    nu_value=nu_value,
    epsilon_value=epsilon_value,
    gs_error=gs_error,
    mode=NormalizationMode.FirstImageParameters,
    debug=True)

last_level_set_bregman, _ = segmentator.run(initial_level_set)
```

```
----------------- Iteration error 0.7019015716797695 ------------------------
Gauss Seidel Iteration:   0%|          | 0/10000 [00:00<?, ?it/s]Gauss Seidel Iteration:  14%|          | 1400/10000 [00:08<00:52, 162.71it/s]
The solution converged after 1400 iterations
----------------- Iteration error 0.4900529663099459 ------------------------
Gauss Seidel Iteration:  14%|          | 1408/10000 [00:07<00:47, 180.05it/s]
The solution converged after 1408 iterations
Converged with an error 0.09202629172143907
```



In [47]:
```
alpha_1 = 0.65
alpha_2 = 1
fig, ax = plt.subplots(1,2)

ax[0].set_title(f'Split Bregman with Force 1')
ax[0].imshow(image, cmap='gray')
ax[0].imshow(last_level_set_bregman > alpha_1, 'jet', interpolation='none', alpha=0.5)
ax[0].set_axis_off()

ax[1].set_title('Alternative with Force 1')
ax[1].imshow(image, cmap='gray')
```

```
ax[1].imshow(last_level_set_1 > alpha_2, 'jet', interpolation='none', alpha=0.5)
ax[1].set_axis_off()
```



In [ ]: