

The goal of this project is to study the solutions of various partial differential equation and implement them on simulated data. In the first section we discuss the solutions to Heat Flow equations using partial differential equations. In the next section we observe and implement the solution to similar flow problems using the level sets method which is a more robust approach. We conclude this assignment by discussing some of the areas in computer vision research where level sets have been used.

1. Heat Equations

(a) The 1D Heat Equations

We are analysing a typical 1-dimensional heat equation here. We assume arbitrary values for x . This is assumed to be the distribution of heat along a 1-dimensional conducting rod of 0 thickness. As time t progresses linearly, we would observe how the heat reaches equilibrium. The following plot gives an insight to the scenario.

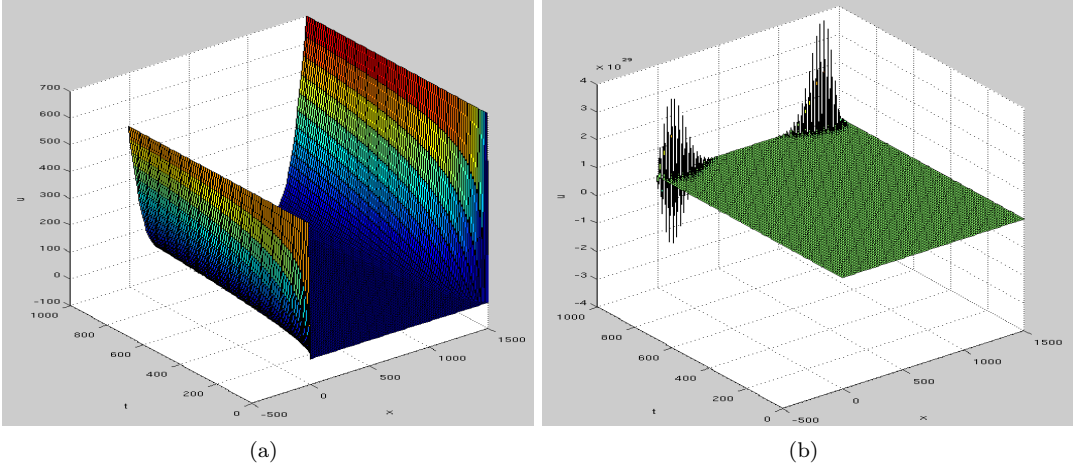


Figure 1: (a) Plot demonstrating the initial heat distribution on a surface (b) The solution demonstrating the distribution of the heat after $t = 1000$.

In order to study how an error in measurement of the distribution affects the estimation of the heat in successive time intervals, we deliberately introduce an error in the measurement in the form of a spike. We demonstrate the simulation results through different plots.

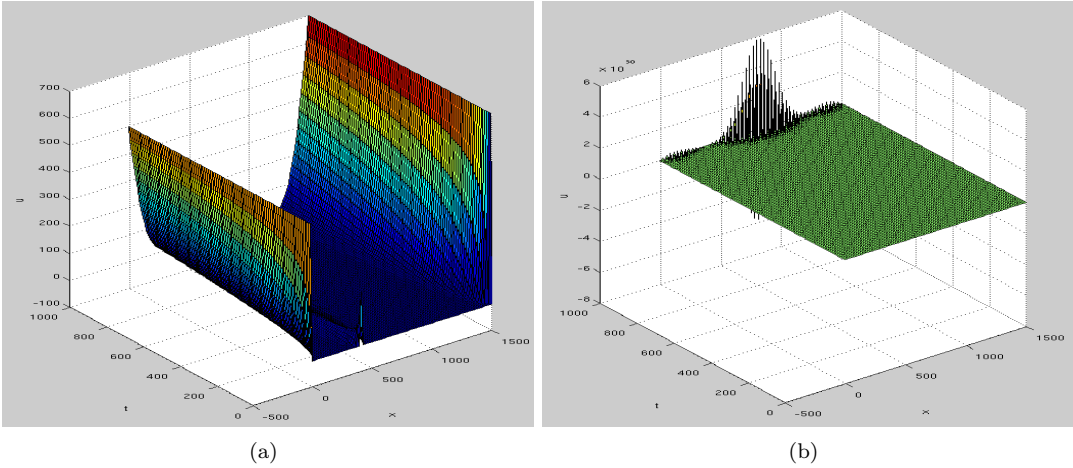


Figure 2: Results with the introduction of an error in measurement: (a) Initial heat distribution with introduced error (b) Plot demonstrating how equilibrium is achieved with error.

Listing 1: Program demonstrating heat flow in 1-dimension

```

1 % x is vector of values and is not restricted in the interval [0,1], init
2 % has the same length of the vector x.
3
4 % lambda = k*dt/(dx^2);
5 % for lambda ≤ 1/2; k ≥ (1/2)*(dx^2/dt);
6 % The spike is for the introduction of error
7 clear all; close all; clc;
8
9 spike = 200;
10 t = linspace(0, 1000, 100);
11 x = linspace(-5, 1500, 100);
12 bdry = [500 700];
13 k = 2;
14
15 init = x(1);
16 u = heat(k, t, x, init, bdry);
17
18 figure; subplot(1, 2, 1);
19 surf(x, t, u);
20 xlabel x; ylabel t; zlabel u;
21
22 err = zeros(size(x));
23 % Select a random position where we could introduce the spike.
24 spikeloc = ceil(length(t)*rand(1));
25 err(1, spikeloc) = spike;
26
27 u = heat(2, t, x, init + err, bdry);
28 subplot(1, 2, 2); surf(x, t, u);
29 xlabel x; ylabel t; zlabel u;
30
31 % For this data t = [0,1000] and x = [-5, 1500], lambda = 0.2980 and k=2.
32 % So for lambda = 0.75, with the same x and t, we have k = 5.0336.
33
34 spike = 200;
35 % In order to make lambda=3/4
36 k = 5.0336;
37 u = heat(k, t, x, init, bdry);
38
39 figure; subplot(1, 2, 1);
40 surf(x, t, u);
41 xlabel x; ylabel t; zlabel u;
42
43 err = zeros(size(t));
44 spikeloc = ceil(length(t)*rand(1));
45 err(1, spikeloc) = spike;
46
47 u = heat(k, t, x, init + err, bdry);
48 subplot(1, 2, 2); surf(x, t, u);
49 xlabel x; ylabel t; zlabel u;

```

Listing 2: Implementation of the heat equation

```

1 function u = heat(k, x, t, init, bdry)
2 % solve the 1D heat equation on the rectangle described by
3 % vectors x and t with u(x, t(1)) = init and Dirichlet boundary
4 % conditions u(x(1), t) = bdry(1), u(x(end), t) = bdry(2).
5 J = length(x);
6 N = length(t);
7 dx = mean(diff(x));
8 dt = mean(diff(t));
9 s = k*dt/dx^2

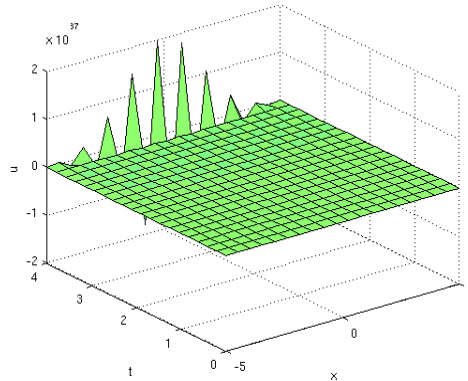
```

```

10 u = zeros(N,J);
11
12 u(1,:) = init;
13 for n = 1:N-1
14     u(n+1,2:J-1) = s*(u(n,3:J) + u(n,1:J-2))+(1-2*s)*u(n, 2:J-1);
15     u(n+1,1) = bdry(1);
16     u(n+1,J) = bdry(2);
17 end

```

We test our programs further by using some fixed inputs as specified in the assignment. The fixed inputs are the dirichlet boundary conditions and $k = 2$, $-5 \geq x \geq 5$ and $0 \geq t \geq 4$.



(a)

Figure 3: Results with predefined set of inputs

Listing 3: Program demonstrating heat flow in 1-dimension with pre-defined initial and boundary value conditions

```

1 % This is the code for problem 1 b. We have been already given the values
2 % for x= (-5, 5), t = (0, 4), bdry = [10, 20] and init: t =10 for x <0 and
3 % t=20 for x>0, k =2
4
5 clear all; close all; clc;
6 % Creating the vectors for both temperature and surface co-ordinate
7 % distribution
8 x = linspace(-5, 5, 20);
9 t = linspace(0, 4, 20);
10 % Selecting boundary conditions
11 bdry = [10 20];
12 k = 2;
13 % Selecting the initial values
14 init = x;
15 init(find(init >0)) = 20;
16 init(find(init <0)) = 10;
17
18 u = heat(k, t, x, init, bdry);
19 figure; surf(x, t, u);
20 xlabel x; ylabel t; zlabel u;

```

- (b) **Extension to the 2D Heat equation** Based on the knowledge of the behavior of 1 Dimensional heat flow equations, we make an attempt to understand 2-dimensional heat equations. We carry out several numerical experiments and discuss the results.

We implement the 2-Dimensional heat equation.

Listing 4: Implementation of the 2 dimensional heat equation

```

1 function u = heat2d(k, mu, init, x, y, t)
2 % a,b,c are parameters that control the boundary conditions
3 % As x and y are selected to be vectors of equal length, lx = ly
4 lx = length(x);
5 ly = length(y);
6 lt = length(t);
7
8 h = mean(diff(x)); %mean(diff(y)) = dy =dx
9 dt = mean(diff(t));
10 % Calculate lambda =s
11 s = k*dt/(h^2);
12 s
13 %u(0, x, y) = f(x, y),
14 %u(t, 0, y) = 0(y),
15 %u(t, 1, y) = 1(y),
16 %u(t, x, 0) = 0(x),
17 %u(t, x, 1) = 1(x)
18
19 % Initialize u
20 u = init;
21 % for each unit time interval, calculate the heat propagation over the
22 % surface
23 for kent = 1:lt-1
24     for jcnt = 2:ly-1
25         for icnt = 2:lx-1
26             u(icnt, jcnt, kent+1) = (1-4*s)*u(icnt, jcnt, kent) + ...
27                                     s*mu*(u(icnt+1, jcnt, kent) + ...
28                                     u(icnt-1, jcnt, kent) + ...
29                                     u(icnt, jcnt+1, kent) + ...
30                                     u(icnt, jcnt-1, kent)) + ...
31                                     s*(1-mu)* (u(icnt+1, jcnt+1, kent) + ...
32                                     u(icnt-1, jcnt+1, kent) + ...
33                                     u(icnt-1, jcnt-1, kent) + ...
34                                     u(icnt+1, jcnt-1, kent));
35         end
36     end
37 end
38 return;

```

We perform experiments with the 2-D heat equation implemented as above with 2 different values of μ , $\mu = 0$ and $\mu = 3$. However we do not notice any significant difference in the output. The following piece of code demonstrates the use of the 2-D heat equation.

Listing 5: Using the 2-Dimensional heat equation

```

1 % This is the code for problem 2 b,c,d . We are required to test the 2D Heat
2 % equation with the help of this script.
3
4 clear all; close all; clc;
5 % Creating the vectors for both temperature and surface co-ordinate
6 % distribution
7 x = linspace(-0.5, 0.5, 50);
8 y = linspace(-0.4, 0.8, 50);
9 t = linspace(0, 3, 20);
10
11 lx = length(x);
12 ly = length(y);
13 lt = length(t);
14
15 % Selecting boundary conditions
16 bdry = [7, 12, 30];

```

```

17 k = 0.001;
18 mu = 0.33;
19
20 init(1:lx, 1:ly, 1:lt) = bdry(1);
21 for kc=1:lt
22     for jc=1:ly
23         init(1, jc, kc) = bdry(3)*(kc<bdry(2)) + bdry(1);
24     end
25 end
26
27 u = heat2d(k, mu, init, x, y, t);
28
29 figure;
30 for ic=1:lt
31     surf(x, y, u(:, :, ic));
32     xlabel x; ylabel y; zlabel t;
33     drawnow;
34 end
35 hold off;

```

We observe that apart from μ , there is another factor λ that affects the heat equation directly. The value of λ depends on k which is selected by us. If we select a bigger value of k , we end up getting a bigger λ and it leads to unstabilities. After several trials, we select $k = 0.001$ so that $\lambda = 0.3791$, we get results that depict stability in the solution.

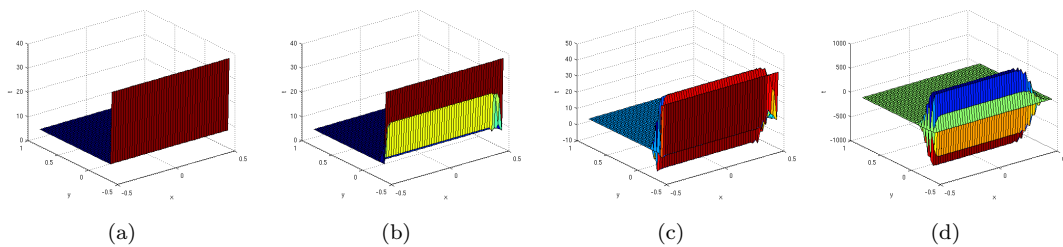


Figure 4: Plots at different number of iteration (i) leading to stable solution (a) $i = 1$, (b) $i = 5$, (c) $i = 10$, and finally (d) $i = 20$.

In order to get a better understanding of the heat flow along a 2-dimensional surface, we use a black and white image as our input with the pixel values depicting the distribution of the heat across the surface. We observe how the heat distributes after a finite period of time. The net effect of an iteration in heat is equivalent to a convolution of the image with a Gaussian filter. Since a Gaussian filter could be used to suppress the high frequency/low frequency components in an image and turn it blurry/sharp depending on the parameters of the Gaussian kernel, this operation could be used to create the same effect.

Listing 6: Using the 2-Dimensional heat equation

```

1 % This is the code for problem 2 b,c,d . We are required to test the 2D Heat
2 % equation with the help of this script.
3
4 clear all; close all; clc;
5 % Creating the vectors for both temperature and surface co-ordinate
6 % distribution
7 x = linspace(-0.5, 0.5, 50);
8 y = linspace(-0.4, 0.8, 50);
9 t = linspace(0, 3, 20);
10
11 lx = length(x);
12 ly = length(y);
13 lt = length(t);
14
15 % Selecting boundary conditions

```

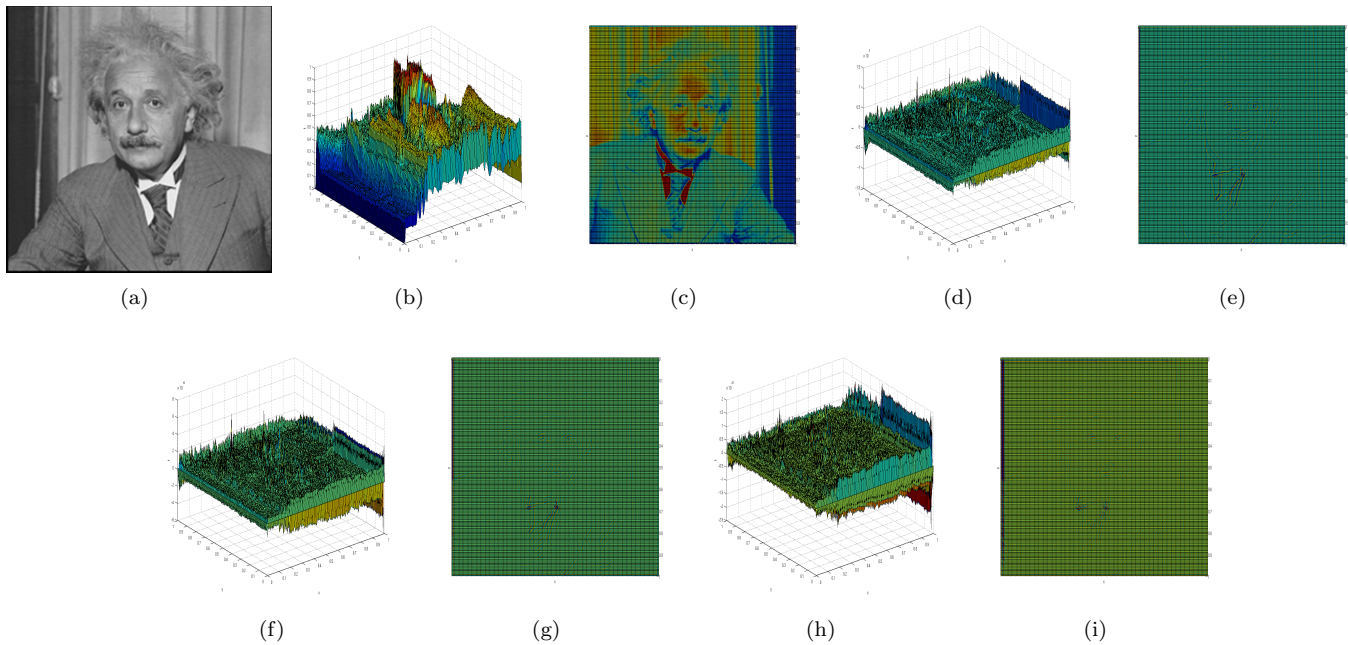


Figure 5: Effect of 2-dimensional heat flow on a surface represented by image: (a) Original iteration, Heat flow at iteration, i (b),(c) $i = 1$, (d),(e) $i = 3$, (f),(g) $i = 5$ and finally (h),(i) $i = 10$

```

16 bdry = [7, 12, 30];
17 k = 0.001;
18 mu = 0.33;
19
20 init(1:lx, 1:ly, 1:lt) = bdry(1);
21 for kent=1:lt
22     for jcnt=1:ly
23         init(1, jcnt, kent) = bdry(3)*(kent<bdry(2)) + bdry(1);
24     end
25 end
26
27 u = heat2d(k, mu, init, x, y, t);
28
29 figure;
30 for icnt=1:lt
31     surf(x, y, u(:, :, icnt));
32     xlabel x; ylabel y; zlabel t;
33     drawnow;
34 end
35 hold off;

```

2. **Level Sets Method** In Computer Vision research, a frequently used tool in tracking is through active contour model. The active contours or snakes technique is used to predict the next state of a moving contour given its previous state based on optical flow analysis. We attempt to solve the active contour problem through a more robust method known as the method of Level Sets where we use the concept of forward and backward differences.

(a) **An accurate and robust implementation** We observe two methods popularly known as the naive method and the upwind scheme to evolve two kinds of simple curves.

The goal of this experiment is to choose a simple curve and evolve it using the level set based approximation technique. We use the figure of eight for this experiment. We plot the curve by using the points generated by its parametric equation for simplicity. We use the naive method in this context.

Listing 7: Program solving the evolution of an eight-shaped curve using the level-set method

```

1 clear all; close all; clc;
2 % Problem 3. Generate group of simple curves. 8 shaped curve, oval shaped
3 % curve. Apply Naive method to evolve the curve for one iteration
4 % for different values of beta.
5 % Curve equations: Parametric form for the figure-8 curve.
6 %  $x = (t^2 - 1)/(t^2 + 1)$ ;  $y = 2t(t^2 - 1)/(t^2 + 1)^2$ 
7
8 rt = [-10, 9];
9 ints = 20;
10 % this t is not the same as the time variable, its just a parameter
11 t = linspace(rt(1), rt(2), ints);
12 x = (t.^2 - 1)./(t.^2 + 1);
13 y = 2*t.*(t.^2 - 1)./((t.^2 + 1).^2);
14
15 % Apply Naive method on this curve, evolve it for one iteration
16 % Computing initial U.
17 for icnt=1:ints
18     for jcnt=1:ints
19         u(icnt, :, 1) = x(icnt);
20         u(:, jcnt, 1) = y(jcnt);
21     end
22 end
23
24 dT = 1;
25 dx = mean(diff(x));
26 dy = mean(diff(y));
27 k = 0.01;
28 B = k;
29 niter = 20;
30
31 figure;
32 for kcnt =2: niter
33     for icnt = 2:ints-1
34         for jcnt = 2:ints-1
35             Dx = (u(icnt +1, jcnt, kcnt-1) - u(icnt, jcnt, kcnt-1))/dx;
36             Dy = (u(icnt, jcnt+1, kcnt-1) - u(icnt, jcnt, kcnt-1))/dy;
37             u(icnt, jcnt, kcnt) = u(icnt, jcnt, kcnt -1) + dT*B*sqrt(Dx^2+Dy^2);
38         end
39     end
40     kcnt
41     surf(x, y, u(:, :, kcnt));
42     xlabel x; ylabel y; zlabel u;
43     drawnow;
44 end

```

In this experiment we select an ellipse as our input curve which we are supposed to evolve using the upwind scheme. The results are discussed in the next section.

Listing 8: Program simulating evolution of an ellipse solved by using the upwind scheme

```

1 clear all; close all; clc;
2 % Problem 3. Generate group of simple curves. 8 shaped curve, oval shaped
3 % curve. Apply Sophisticated method to evolve the curve for one iteration
4 % for different values of beta.
5
6 rt = [-10, 9];
7 ints = 20;
8 % this t is not the same as the time variable, its just a parameter
9 t = linspace(rt(1), rt(2), ints);
10 p = [20, 10, 40, 30];
11 x = p(1) + p(2)*sin(t);
12 y = p(3) + p(4)*cos(t);
13

```

```

14 % Apply Sophisticated method on this curve, evolve it for one iteration
15 % Computing initial U.
16 for icnt=1:ints
17     for jcnt=1:ints
18         u(icnt, :, 1) = x(icnt);
19         u(:, jcnt, 1) = y(jcnt);
20     end
21 end
22
23 dT = 1;
24 dx = mean(diff(x));
25 dy = mean(diff(y));
26 k = 0.0001;
27 B = k;
28 niter = 100;
29
30 figure;
31 for kcnt =2: niter
32     for icnt = 2:ints-1
33         for jcnt = 2:ints-1
34             % Calculating Forward difference in both x and y directions
35             Dxp = (u(icnt +1, jcnt, kcnt-1) - u(icnt, jcnt, kcnt-1))/dx;
36             Dyp = (u(icnt, jcnt+1, kcnt-1) - u(icnt, jcnt, kcnt-1))/dy;
37             % Calculating Backward difference in both x and y directions
38             Dxm = (u(icnt, jcnt, kcnt-1) - u(icnt-1, jcnt, kcnt-1))/dx;
39             Dym = (u(icnt, jcnt, kcnt-1) - u(icnt, jcnt-1, kcnt-1))/dy;
40             delp = sqrt((max(Dxm,0))^2 + (min(Dxp,0))^2 + ...
41                 (max(Dym,0))^2 + (min(Dyp,0))^2);
42             delm = sqrt((max(Dxp,0))^2 + (min(Dxm,0))^2 + ...
43                 (max(Dyp,0))^2 + (min(Dym,0))^2);
44
45             u(icnt, jcnt, kcnt) = u(icnt, jcnt, kcnt -1) - ...
46                 dT*(max(-B,0)*delp + min(-B, 0)*delm);
47         end
48     end
49     kcnt
50     surf(x, y, u(:, :, kcnt));
51     xlabel x; ylabel y; zlabel u;
52     drawnow;
53 end

```

Both the experiments performed above have exploited different values of k . However we achieve the best results for $k = \beta$. We chose to display a wider interval of iteration for the experiment performed using the sophisticated method because the changes are hardly noticeable between closer iterations.

- (b) **Level Sets method in Research** This section involves the study of level sets methods in research, Computer Vision in particular [1].

i. We are required to derive the equation

$$\epsilon_x^{LBF}(\phi, f_1(x), f_2(x)) = \lambda_1 \int k_\sigma(x-y)|I(y)-f_1(x)|^2 H(\phi(y))dy + \lambda_2 \int k_\sigma(x-y)|I(y)-f_2(x)|^2 (1-H(\phi(y)))dy$$

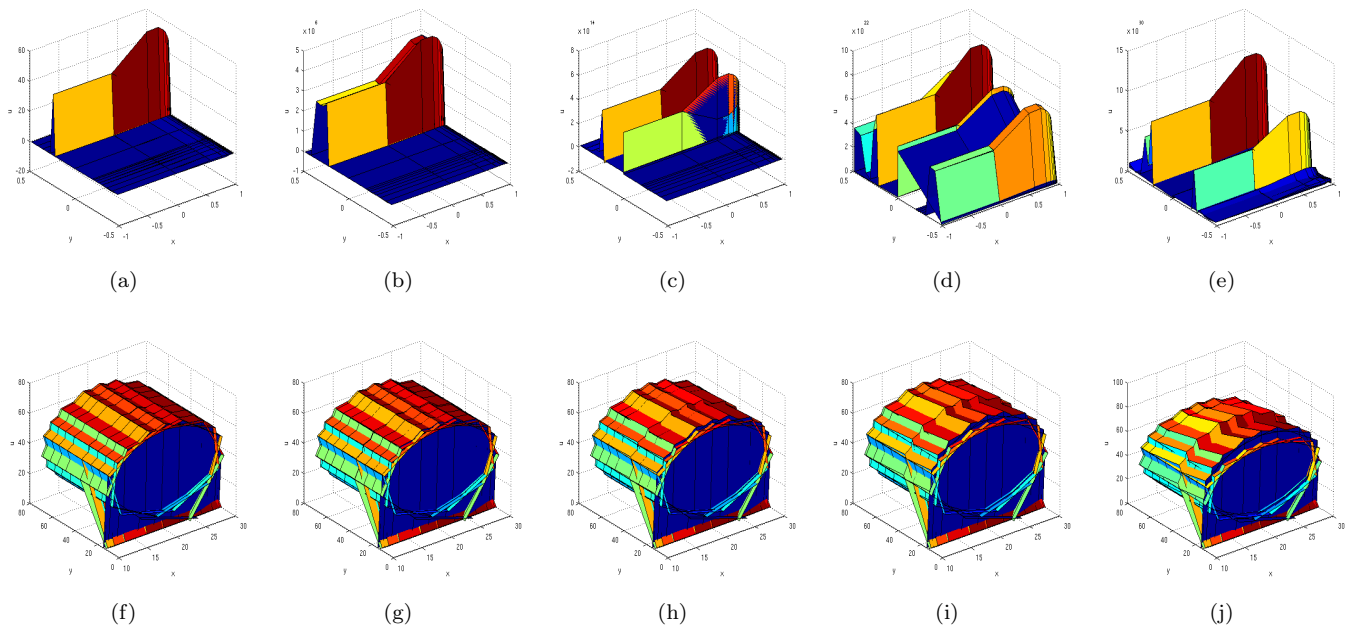


Figure 6: Evolving curves by use of level set method. The curve being a figure of 8 and the results are displayed at the end of iteration (a) $i = 2$, (b) $i = 5$, (c) $i = 10$, (d) $i = 15$, (e) $i = 20$. The second set of results are on an ellipse and the respective plots depict iteration (f) 2, (g) 10, (h) 25, (i) 50 and (j) 100.

Bibliography

- [1] Chumming Li, Chiu-Yen Kao, John C. Gore and Zhaohua Ding, “Implicit Active Contours Driven by Local Binary Fitting Energy”, CVPR 2007.