MECE 5397 Scientific Computing for Mechanical Engineers

Spring 2019

Final Project

APc1-6

 Solution to the Poisson Equation

By Leon Marchant

For Amit Amritkar & Andrea Prosperetti

 Due: 05/05/19

Submitted On: 05/05/19

**Abstract**

The purpose of this project it to expose students to the entire process of solving a partial differential equation. The Poisson equation was chosen to be solved numerically using two methods, the Gauss-Seidel method and the Successive Over Relaxation method. The mathematical statemen was defined, and with this information, the equations needed to be discretized. The Dirichlet and Neumann boundary conditions required to be implemented correctly in their own ways. A code was written for each method in order to compare their processing speed and precision. This code was analyzed in great detail to display the thought process behind every decision made whilst coding. The program used checkpoints to save data during large number of iterations so that data from the last checkpoint could be used to continue the calculation in case of failure. The results of several cases were visualized and analyzed, and the two methods were compared using the same variables. Throughout the entire process, source control was used to have the project traceable and reproduceable. A list of the computer’s specifications was also supplied.

The mathematical statement for the project can be seen below in figure 1.

A close up of text on a white background

Description automatically generated

Figure 1: Mathematical statement of the problem

The problem is broken down into three parts. The two-dimensional Poisson equation, the domain in which the problem is to be solved, and the boundary conditions that encompass the problem. There are two Dirichlet boundary conditions occurring on the top and bottom of the domain, these are denoted as phi for the bottom and psi for the top. On the left and right side however, there are two Neumann boundary conditions, these are more challenging to implement since they require the use of ghost nodes. The reason ghost nodes are needed will become clear upon discretizing the problem.

The domain needs to be divided into a finite number of elements so that the Poisson equation may be solved numerically. Figure 2 and 3 below show how the Poisson equation and the Neumann boundary conditions were discretized.

A close up of text on a white background

Description automatically generatedA close up of text on a white background

Description automatically generated

Figure 2: Discretization of the Poisson Equation Figure 3: Discretization of Neumann Boundary Conditions

In order to utilize the Neumann boundary conditions on the left and right side of the domain, the information that the left and right points of the boundary are equal needs to be placed into the Poisson equation at the first and last collumns.

The psudocode below,written in Matlab, outlines the steps taken to solve the Poisson equation numerically, as well as the comments outlining the logic behind each decision:

Regular Housekeeping commands.

clc

clear

close

Next the timer is set to start, the checkpoint file is created, and the domain of interest is defined.

tic

checkpoint='checkpoint.mat';

ax=-pi;

bx=pi;

ay=-pi;

by=pi;

The domain is discretized into nx and ny increments, these values can be changed in order to have a finer mesh, which will take longer to process, or a coarser one. The lengths of the of the increments are then calculated with the domain in mind as hx and hy.

nx=99;

ny=99;

hx = (bx-ax)/(nx-1);

hy = (by-ay)/(ny-1);

The matrices of x, y, F and u are now pre-allocated to improve processing time. The size of the matrix is determined by the number of increments.

x=zeros(1,nx);

y=zeros(1,ny);

F=zeros(ny,nx);

u=zeros(ny,nx);

For loops are then used to fill the x and y grid and place the Dirichlet boundary conditions on the top and bottom of the u matrix.

for k = 1:nx

x(k) = ax+hx\*(k-1);

u(ny,k) = (x(k)-ax)\*(x(k)-ax)\*sin(pi\*(x(k)-ax)/(2\*(bx-ax)));

u(1,k) = (cos(pi\*(x(k)-ax))-1)\*cosh(bx-x(k));

end

for j = 1:ny

y(j) = ay+hy\*(j-1);

end

The initial value for error e, checkpointing count t, and iteration count i are set. The value of the wanted magnitude of error is also established and can be altered to improve or worsen the precision of the code.

E=1;

Ewanted=10^-7;

t=0;

i=0;

The two methods now separate and must use their individual code.

For the Gauss-Seidel method:

The while loop is used to have the script continue to iterate until the desired magnitude of error is reached.

while E > Ewanted

GSu is now saved as the prior u going into the next set of for loops. This is a crucial part of the Gauss-Seidel method to improve processing times over the regular Jacobi method.

GSu = u;

The two for loops are set up to repeat the calculation of the u value at every increment along the x and y grid. Every column and all but the first and last rows will be calculated. This is because if and else if statements take care of the Neumann boundary conditions at those locations.

for k = 1:nx

for j = 2:ny-1

The value of F is calculated for subsequent use in the equation for u.

F(j,k) = sin(pi\*(x(k)-ax)/(bx-ax)).\*cos(pi\*(2\*(y(j)-ay)/(by-ay)+1)/2);

The if statement for k equals 1 calculates the value of u at the left boundary, and for k equals nx it calculates the right one. Within the discretized equation at the boundaries, the ghost node values are replaced with the values within the domain. At all other positions, the regular discretized Poisson equation is used. Per the Gauss-Seidel method, the values from the prior iteration are used where possible to hasten the process.

if k==1

u(j,k) = (2\*u(j,k+1)+u(j-1,k)+GSu(j+1,k) +F(j,k)\*hy\*hx)/4

elseif k==nx

u(j,k) = (2\*u(j,k-1)+u(j-1,k)+GSu(j+1,k)+F(j,k)\*hy\*hx)/4;

else

u(j,k) = (u(j,k-1)+GSu(j,k+1)+u(j-1,k)+GSu(j+1,k)… +F(j,k)\*hy\*hx)/4;

end

end

end

The next segment of code functions to save the checkpointed data to be able to begin from a saved iteration if the code fails. It is set to save and tell the user every 1000 iterations.

t=t+1;

if t == 1000

disp('Checkpointing program, 1000 iterations have occurred');

save(checkpoint);

t=0;

end

The last part of the while loop serves to count iterations and calculate whether a new iteration is needed. The error is calculated using the L infinite error equation. This process finds the difference between iterations.

i=i+1;

E = max(max(abs(GSu-u)));

End

The next if statement was put in place to aid with visualization of the surface plots. If the number of nodes exceed 100, the grid lines are so concentrated that the color in the plots is obscured. However, at lower values, the grid lines serve as a good tool to visualize the mesh size. If either the nx or ny values are larger than 99, there is an additional line added to remove the grid lines. The graph has axis labels and a title with their fonts made larger to improve readability. A color bar is present for improved visualization, and the color map is altered to “cool” since this scale is considerate towards people whom may be color blind, and also does well to display changes in magnitude.

r = max(nx,ny);

if r<100

graph = surf(x,y,u);

xlabel('x','Fontsize',16);

ylabel('y','Fontsize',16);

zlabel('u(x,y)','Fontsize',16);

title('Solution to the Poisson Equation Using the Gauss-Seidel Method','Fontsize',16);

colorbar('vertical')

colormap('cool')

else

graph = surf(x,y,u);

xlabel('x','Fontsize',16);

ylabel('y','Fontsize',16);

zlabel('u(x,y)','Fontsize',16);

title('Solution to the Poisson Equation Using the Gauss-Seidel Method','Fontsize',16);

colorbar('vertical')

colormap('cool')

set(graph,'edgecolor','none')

end

Lastly, the timer is ended the user is displayed the total time taken for the calculation to complete.

toc

The difference in the Successive Over Relaxation method lies with the parameter omega. Omega was found using the equation shown below. This equation was referenced from a report titled “The Optimal Relaxation Parameter for the SOR Method Applied to the Poisson Equation in Any Space Dimensions” and was written by Shiming Yang and Matthias K. Gobbert. They outline that the parameter’s optimal value changes depending on the number of nodes.

wopt = 2/(1+sin(pi\*hx));

while E > Ewanted

Similarly to the Gauss-Seidel method the prior value is saved for use in the next iteration. The two if statements concerning the boundary conditions have GSu replaced with SOR. The equation for u(j,k) changes significantly. The previous value for u is multiplied by 1-wopt and the rest is multiplied by wopt. Omega is between 1 and 2, and accelerates the processing speed by increasing the amount that u(x,y) changes by each iteration.

SORu = u;

for k = 1:nx

for j = 2:ny-1

F(j,k) = sin(pi\*(x(k)-ax)/(bx-ax)).\*cos(pi\*(2\*(y(j)-ay)/(by-ay)+1)/2);

if k==1

u(j,k) = (2\*u(j,k+1)+u(j-1,k)+SORu(j+1,k)+F(j,k)\*hy\*hx)/4

elseif k==nx

u(j,k) = (2\*u(j,k-1)+u(j-1,k)+SORu(j+1,k)+F(j,k)\*hy\*hx)/4;

else

u(j,k) = (1-wopt)\*SORu(j,k)+wopt\*(u(j,k-1)+SORu(j,k+1)+u(j-1,k)+… SORu(j+1,k)+F(j,k)\*hy\*hx)/4;

end

end

end

A detailed list of the specifications of the personal computer used to complete this project is listed below:

Operating System

Windows 10 Home 64-bit

Computer type: Desktop

CPU

Intel Core i7 7700K

Cores 4

Threads 8

Name Intel Core i7 7700K

Code Name Kaby Lake

Package Socket 1151 LGA

Technology 14nm

Specification Intel Core i7-7700K CPU @ 4.20GHz

Family 6

Extended Family 6

Model E

Extended Model 9E

Stepping 9

Revision B0

Instructions MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, Intel 64, NX, VMX, AES, AVX, AVX2, FMA3

Virtualization Supported, Enabled

Hyperthreading Supported, Enabled

Fan Speed 2662 RPM

Bus Speed 99.9 MHz

Stock Core Speed 4200 MHz

Stock Bus Speed 100 MHz

Average Temperature 48 °C

Caches

L1 Data Cache Size 4 x 32 Kbytes

L1 Instructions Cache Size 4 x 32 Kbytes

L2 Unified Cache Size 4 x 256 Kbytes

L3 Unified Cache Size 8192 Kbytes

Cores

Core 0

Core Speed 4196.9 MHz

Multiplier x 42.0

Bus Speed 99.9 MHz

Temperature 50 °C

Threads APIC ID: 0, 1

Core 1

Core Speed 4196.9 MHz

Multiplier x 42.0

Bus Speed 99.9 MHz

Temperature 46 °C

Threads APIC ID: 2, 3

Core 2

Core Speed 4196.9 MHz

Multiplier x 42.0

Bus Speed 99.9 MHz

Temperature 47 °C

Threads APIC ID: 4, 5

Core 3

Core Speed 4196.9 MHz

Multiplier x 42.0

Bus Speed 99.9 MHz

Temperature 48 °C

Threads APIC ID: 6, 7

RAM

Memory slots

Total memory slots 4

Used memory slots 2

Free memory slots 2

Memory

Type Unknown

Size 16384 Mbytes

Channels # Dual

DRAM Frequency 1199.2 MHz

CAS# Latency (CL) 16 clocks

RAS# to CAS# Delay (tRCD) 16 clocks

RAS# Precharge (tRP) 16 clocks

Cycle Time (tRAS) 39 clocks

Command Rate (CR) 2T

Physical Memory

Memory Usage 46 %

Total Physical 16 GB

Available Physical 8.57 GB

Total Virtual 32 GB

Available Virtual 19 GB

SPD

Number Of SPD Modules 2

Slot #1

Type Unknown

Size 8192 Mbytes

Manufacturer Corsair

Max Bandwidth DDR4-2400 (1200 MHz)

Part Number CMK16GX4M2A2400C16

SPD Ext. XMP

Slot #2

Type Unknown

Size 8192 Mbytes

Manufacturer Corsair

Max Bandwidth DDR4-2400 (1200 MHz)

Part Number CMK16GX4M2A2400C16

SPD Ext. XMP

Motherboard

Manufacturer MSI

Model Z270 GAMING PLUS (MS-7A75) (U3E1)

Version 1.0

Chipset Vendor Intel

Chipset Model Kaby Lake

Chipset Revision 05

Southbridge Vendor Intel

Southbridge Model Z270

Southbridge Revision 00

System Temperature 39 °C

Graphics

NVIDIA GeForce GTX 970

Manufacturer NVIDIA

Model GeForce GTX 970

Device ID 10DE-13C2

Revision A2

Subvendor MSI (1462)

Current Performance Level Level 0

Voltage 0.856 V

Technology 28 nm

Bus Interface PCI Express x16

Temperature 42 °C

Driver version 26.21.14.3039

BIOS Version 84.04.36.00.f1

Memory 4095 MB

Count of performance levels : 1

Level 1 - "Perf Level 0"

Storage

Hard drives

ADATA SU650 (SSD)

Heads 16

Cylinders 29,185

Tracks 7,442,175

Sectors 468,857,025

SATA type SATA-III 6.0Gb/s

Device type Fixed

ATA Standard ACS2

Serial Number 2H4220032186

Firmware Version Number Q0831C0

LBA Size 48-bit LBA

Power On Count 628 times

Power On Time 175.5 days

Speed Not used (SSD Drive)

Features S.M.A.R.T., APM, NCQ, TRIM, SSD

Max. Transfer Mode SATA III 6.0Gb/s

Used Transfer Mode SATA III 6.0Gb/s

Interface SATA

Capacity 223 GB

Real size 240,057,409,536 bytes

RAID Type None

SanDisk SDSSDA240G (SSD)

Manufacturer SanDisk

Heads 16

Cylinders 29,185

Tracks 7,442,175

Sectors 468,857,025

SATA type SATA-III 6.0Gb/s

Device type Fixed

ATA Standard ACS2

Serial Number 160697403140

Firmware Version Number U21010RL

LBA Size 48-bit LBA

Power On Count 1549 times

Power On Time 259.1 days

Speed Not used (SSD Drive)

Features S.M.A.R.T., APM, NCQ, TRIM, SSD

Max. Transfer Mode SATA III 6.0Gb/s

Used Transfer Mode SATA III 6.0Gb/s

Interface SATA

Capacity 223 GB

Real size 240,057,409,536 bytes

RAID Type None

SanDisk SDSSDA960G (SSD)

Manufacturer SanDisk

Heads 16

Cylinders 116,737

Tracks 29,767,935

Sectors 1,875,379,905

SATA type SATA-III 6.0Gb/s

Device type Fixed

ATA Standard ACS2

Serial Number 180905800400

Firmware Version Number Z33130RL

LBA Size 48-bit LBA

Power On Count 194 times

Power On Time 77.9 days

Speed Not used (SSD Drive)

Features S.M.A.R.T., APM, NCQ, TRIM, SSD

Max. Transfer Mode SATA III 6.0Gb/s

Used Transfer Mode SATA III 6.0Gb/s

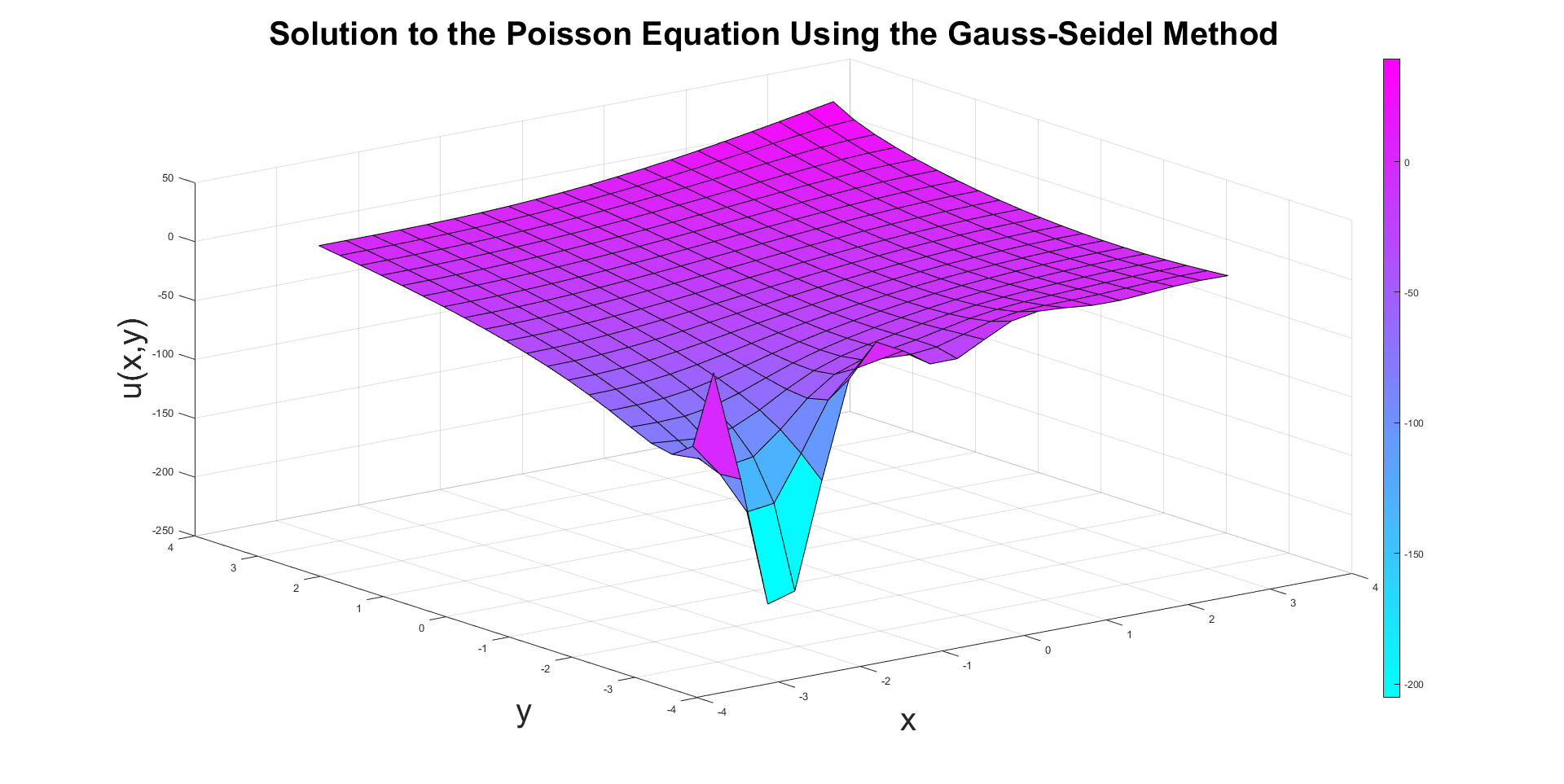
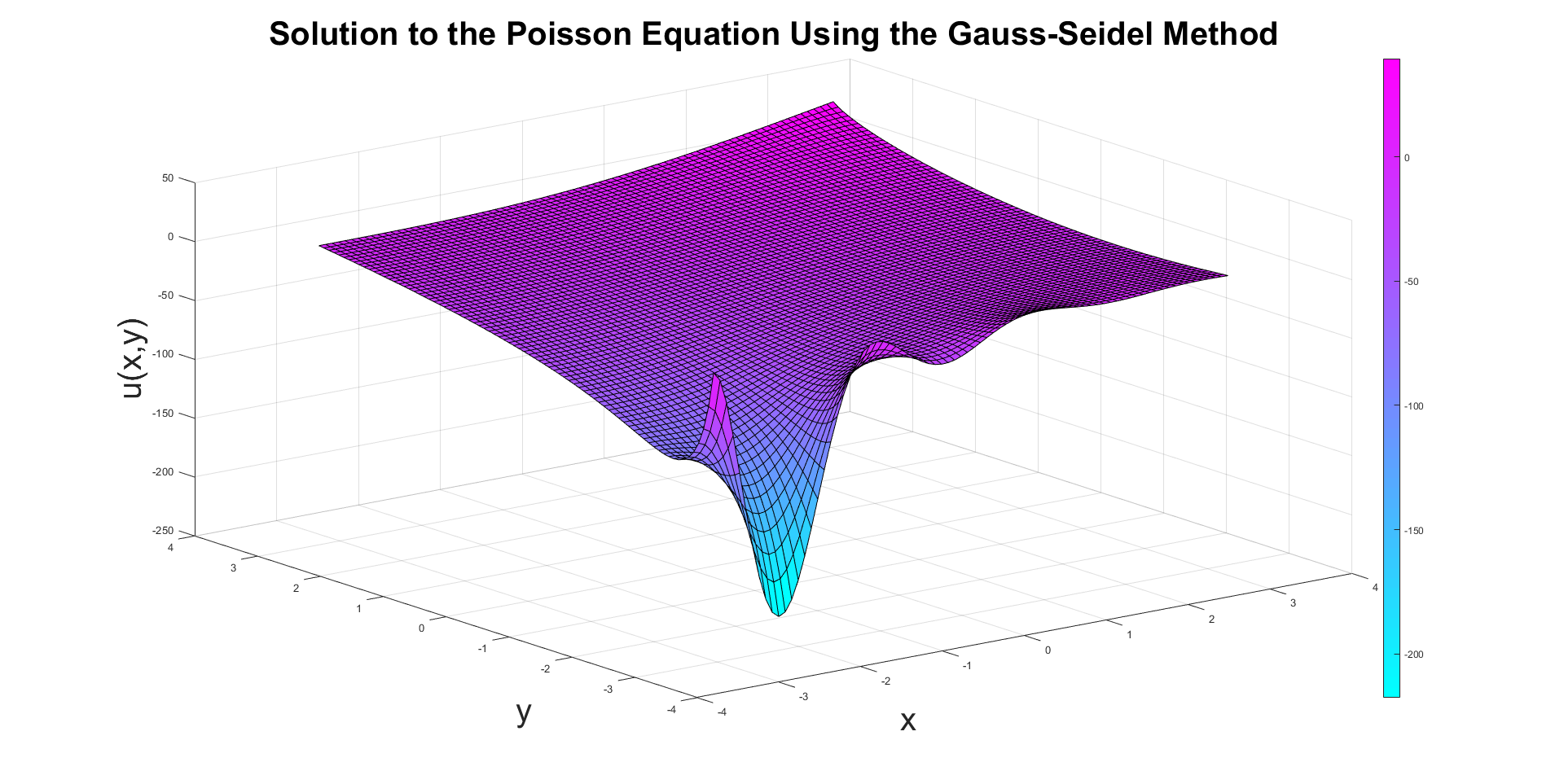
Interface SATA

Capacity 894 GB

Real size 960,197,124,096 bytes

RAID Type None

The surface plots of several grid sizes are shown below, comparisons between the mesh sizes and between the Gauss-Seidel method and the Successive Over Relaxation method will be made.

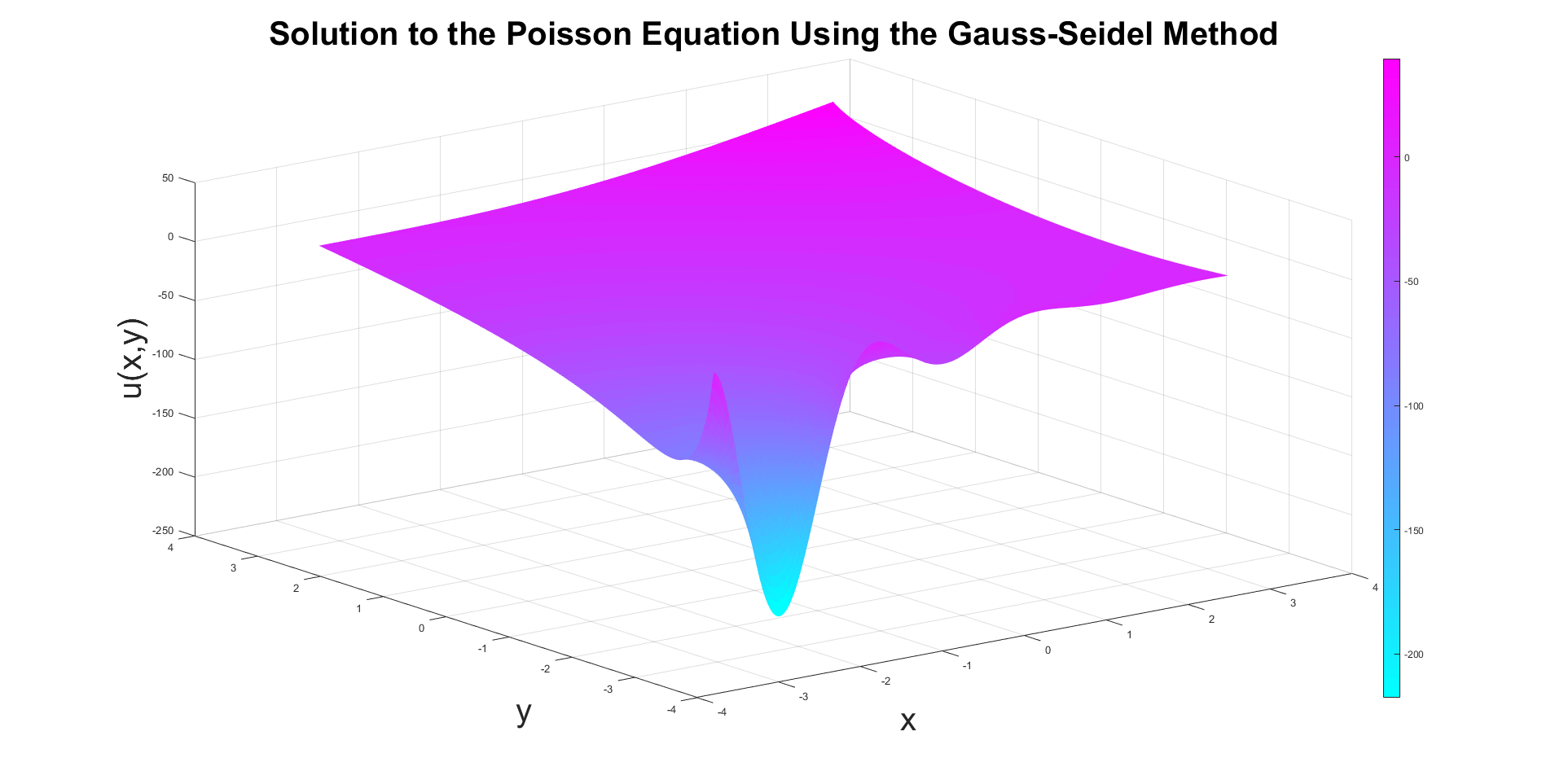
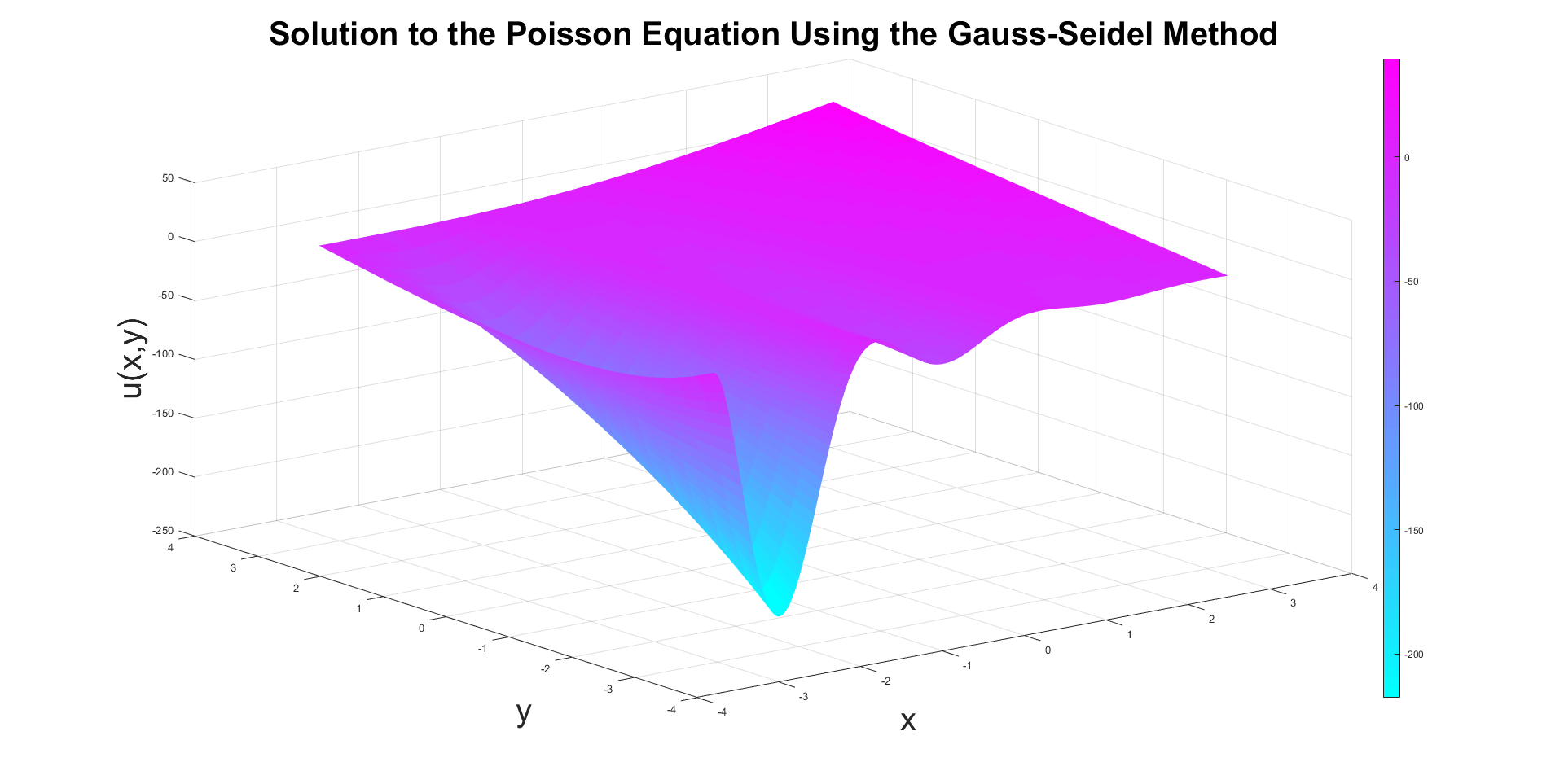
 

Grid size nx = 20 ny = 20 Grid size nx = 80 ny = 80

Elapsed time is 0.143996 seconds Elapsed time is 4.590366 seconds

Iterations performed 1076 Iterations performed 15093

Magnitude of error 10^-7 Magnitude of error 10^-7

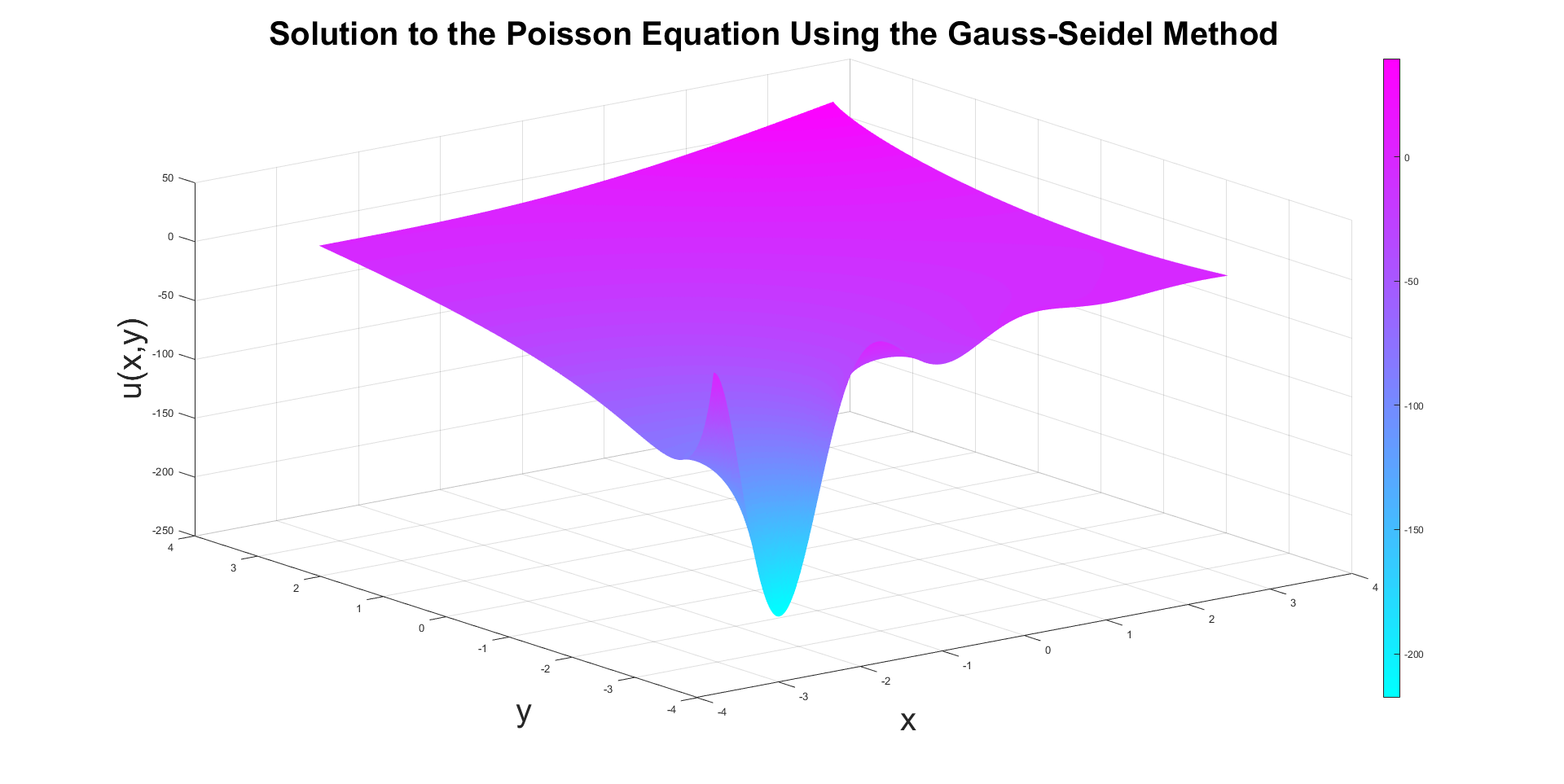
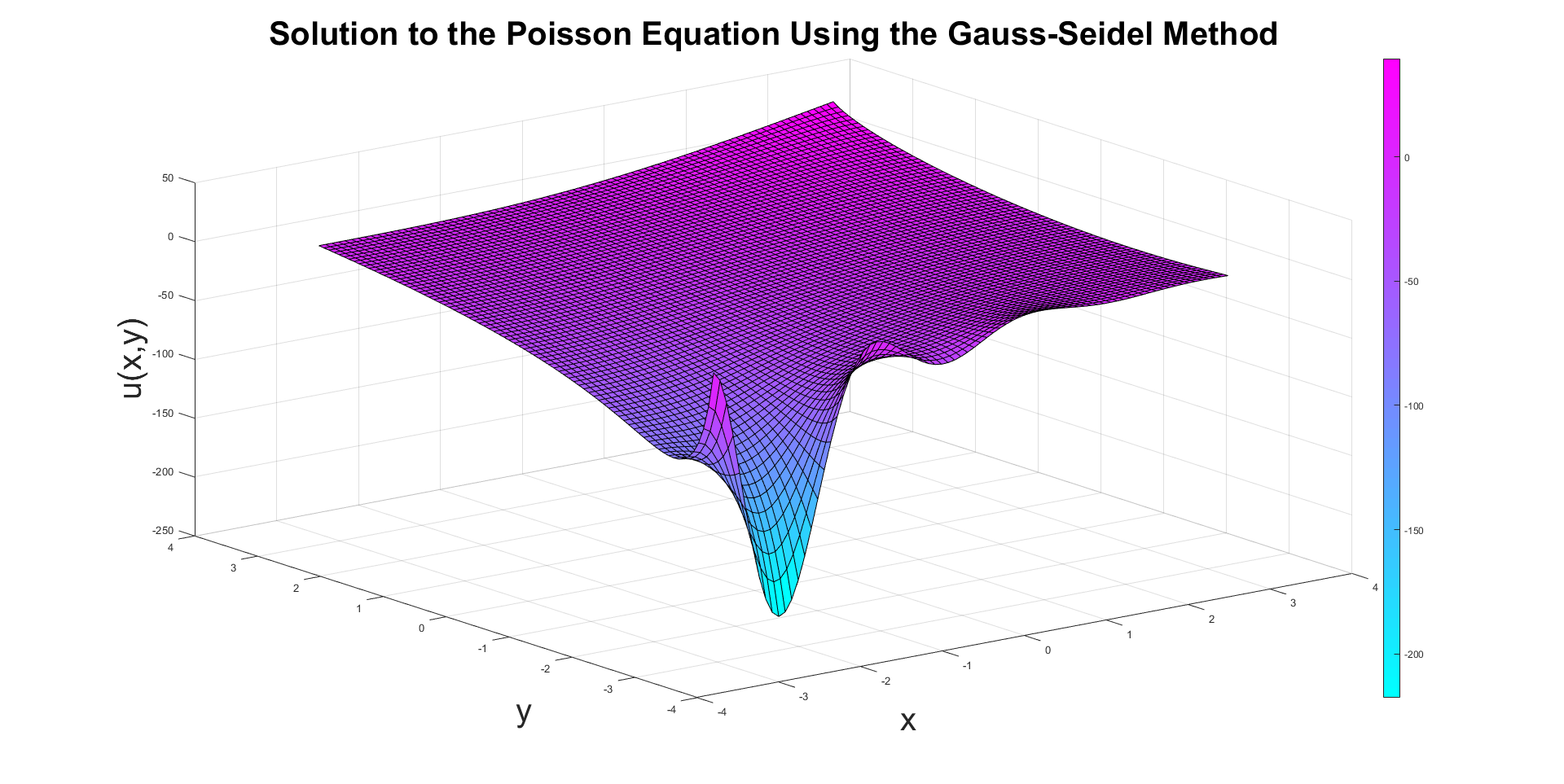
 

Grid size nx = 200 ny = 200 Grid size nx = 200 ny = 20

Elapsed time is 145.808788 seconds Elapsed time is 0.282525 seconds

Iterations performed 81061 Iterations performed 1185

Magnitude of error 10^-7 Magnitude of error 10^-7

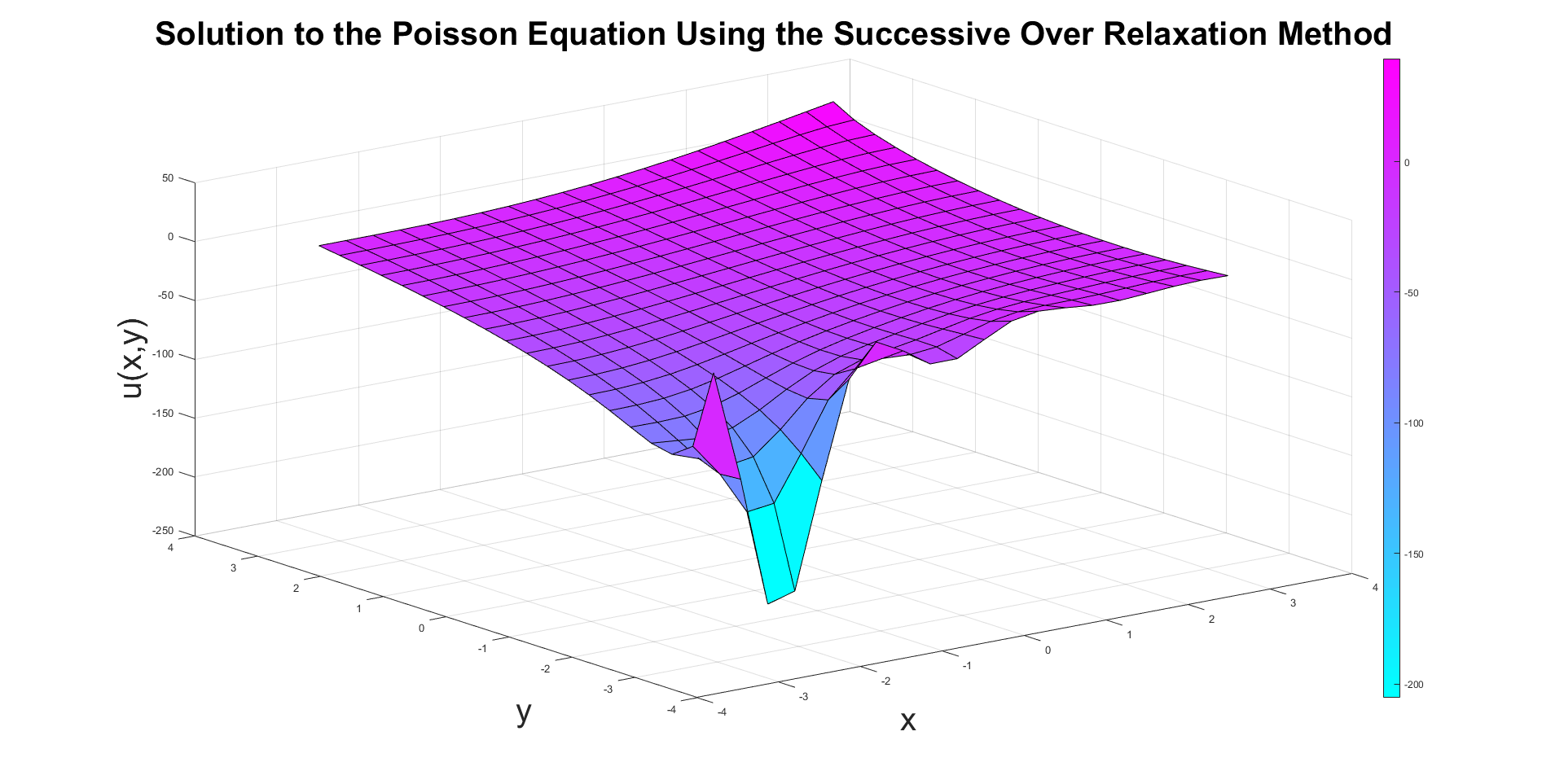
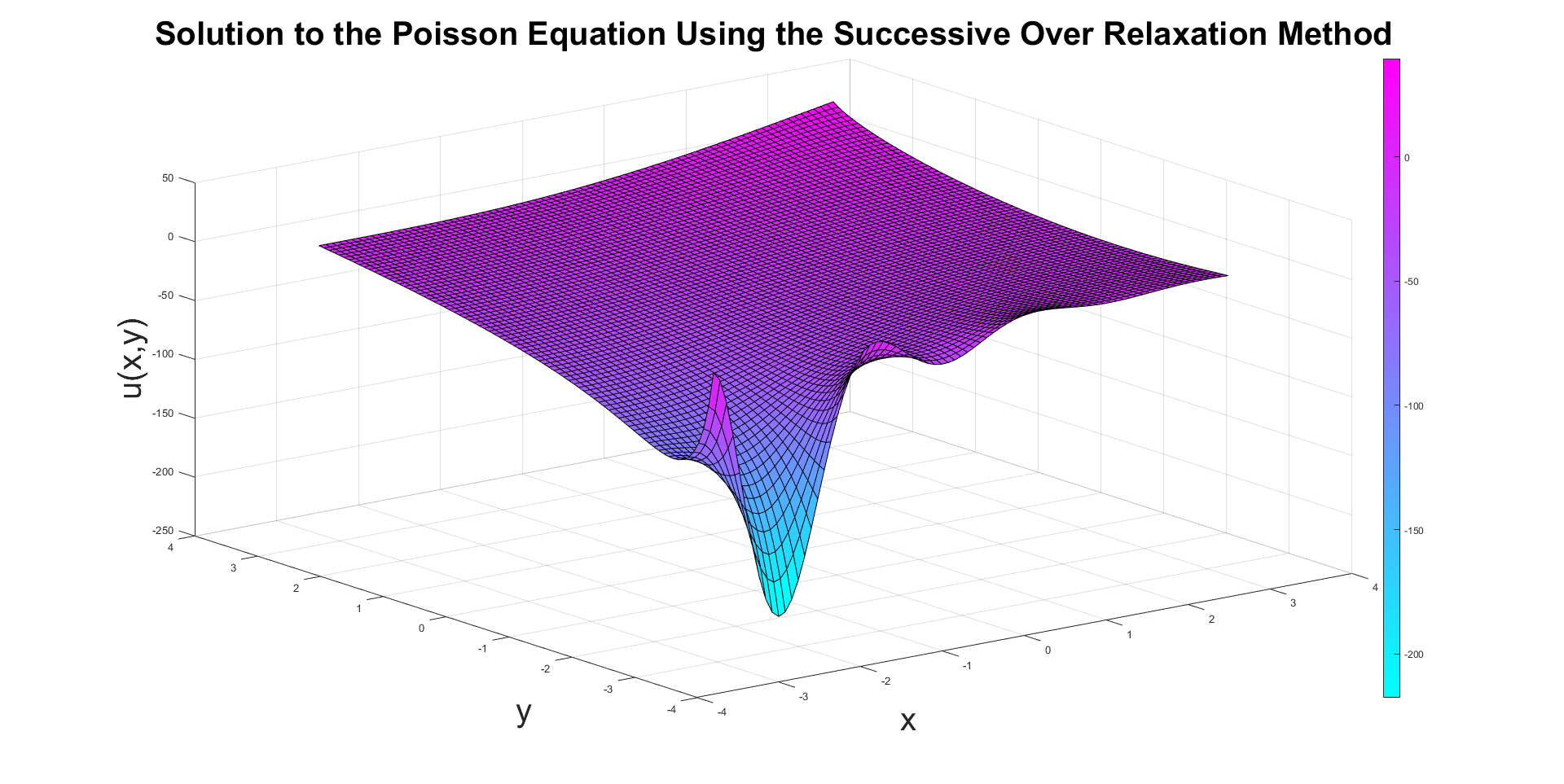
 

Grid size nx = 500 ny = 500 Grid size nx = 80 ny = 80 F(x,y) = 0

Elapsed time is 2152.465093 seconds Elapsed time is 1.328990 seconds

Iterations performed 187760 Iterations performed 14910

Magnitude of error 10^-5 Magnitude of error 10^-7

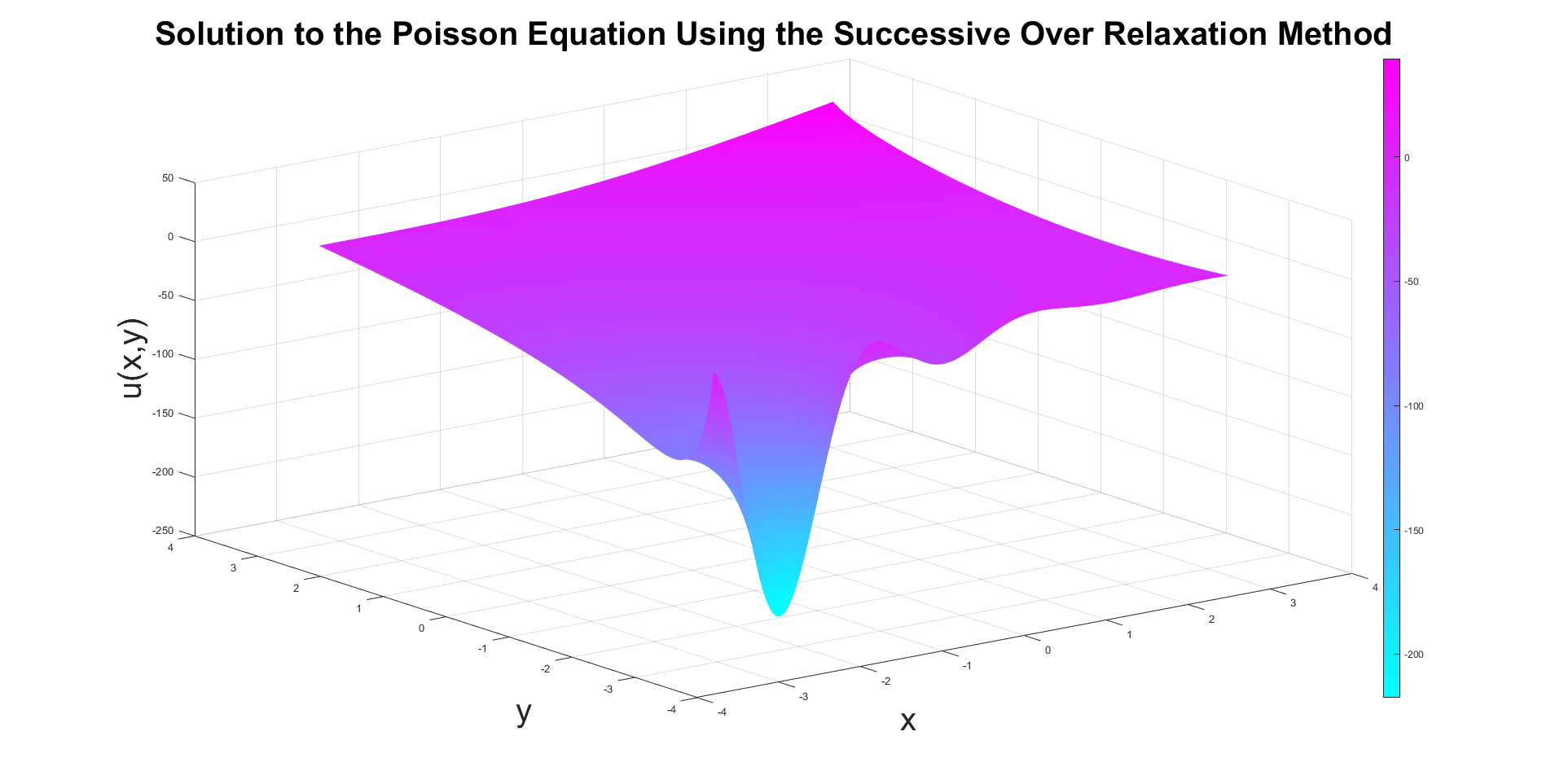
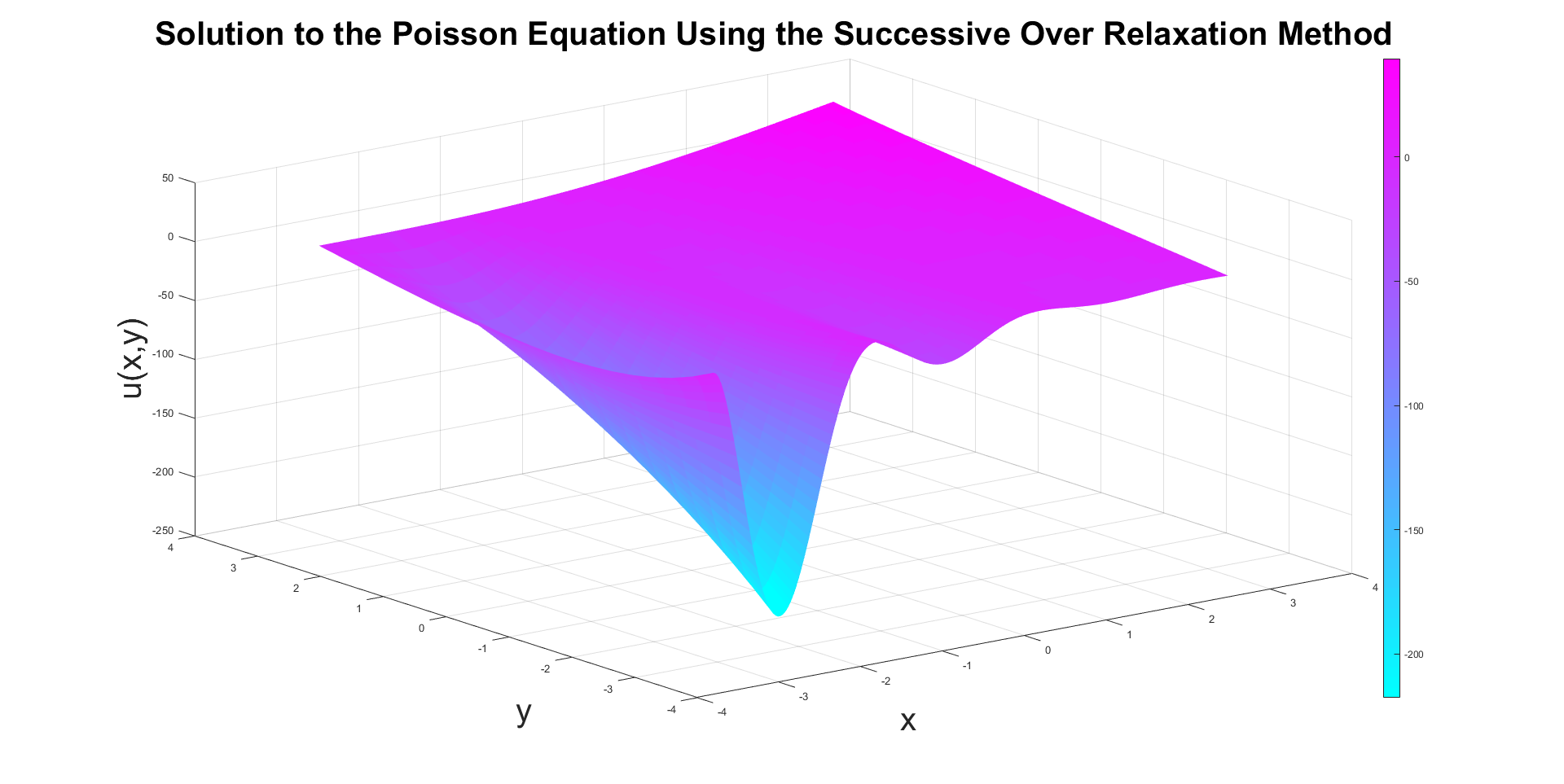
 

Grid size nx = 20 ny = 20 Grid size nx = 80 ny = 80

Elapsed time is 0.124487 seconds Elapsed time is 1.487479 seconds

Iterations performed 943 Iterations performed 4298

Magnitude of error 10^-7 Magnitude of error 10^-7

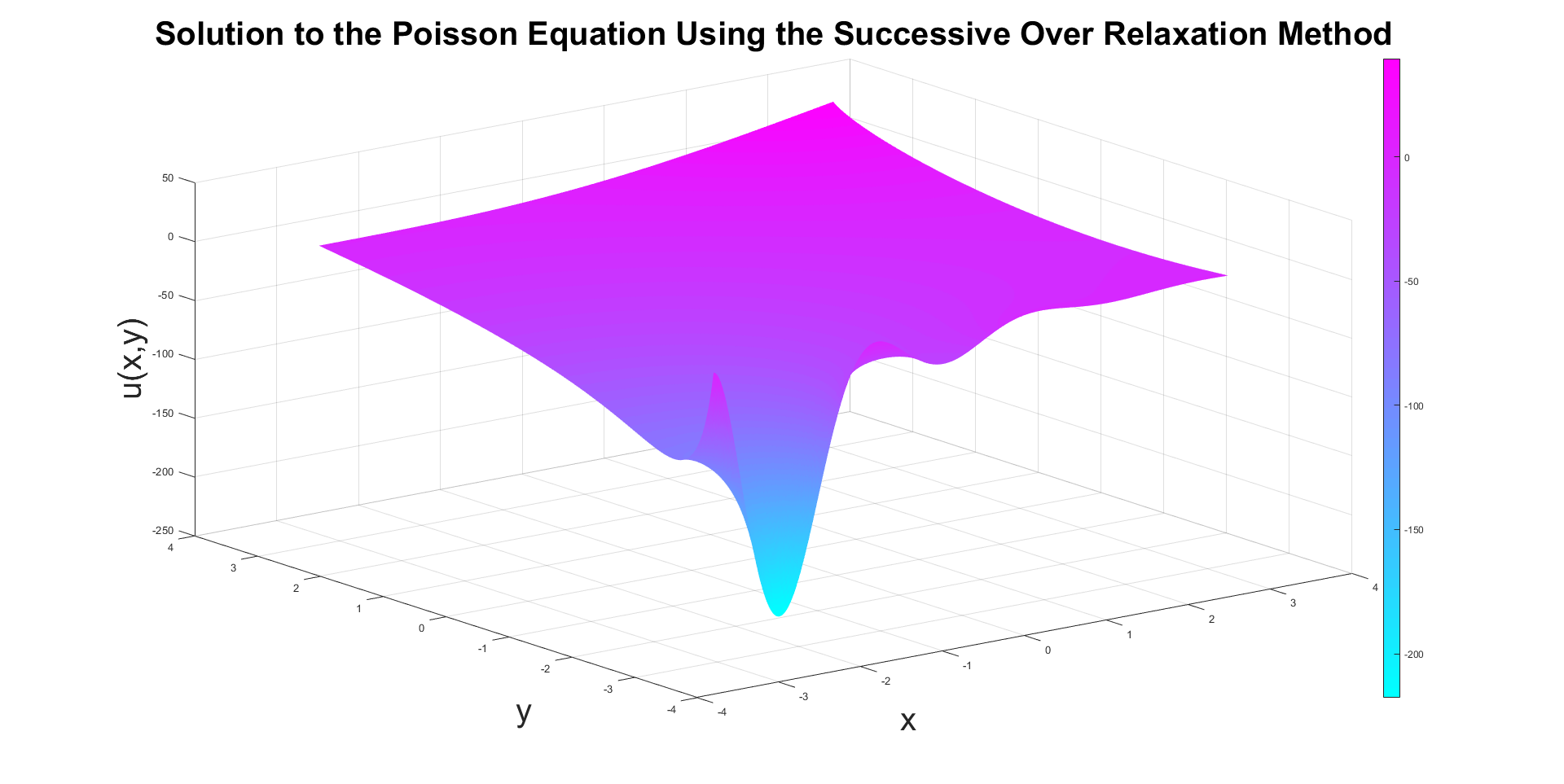
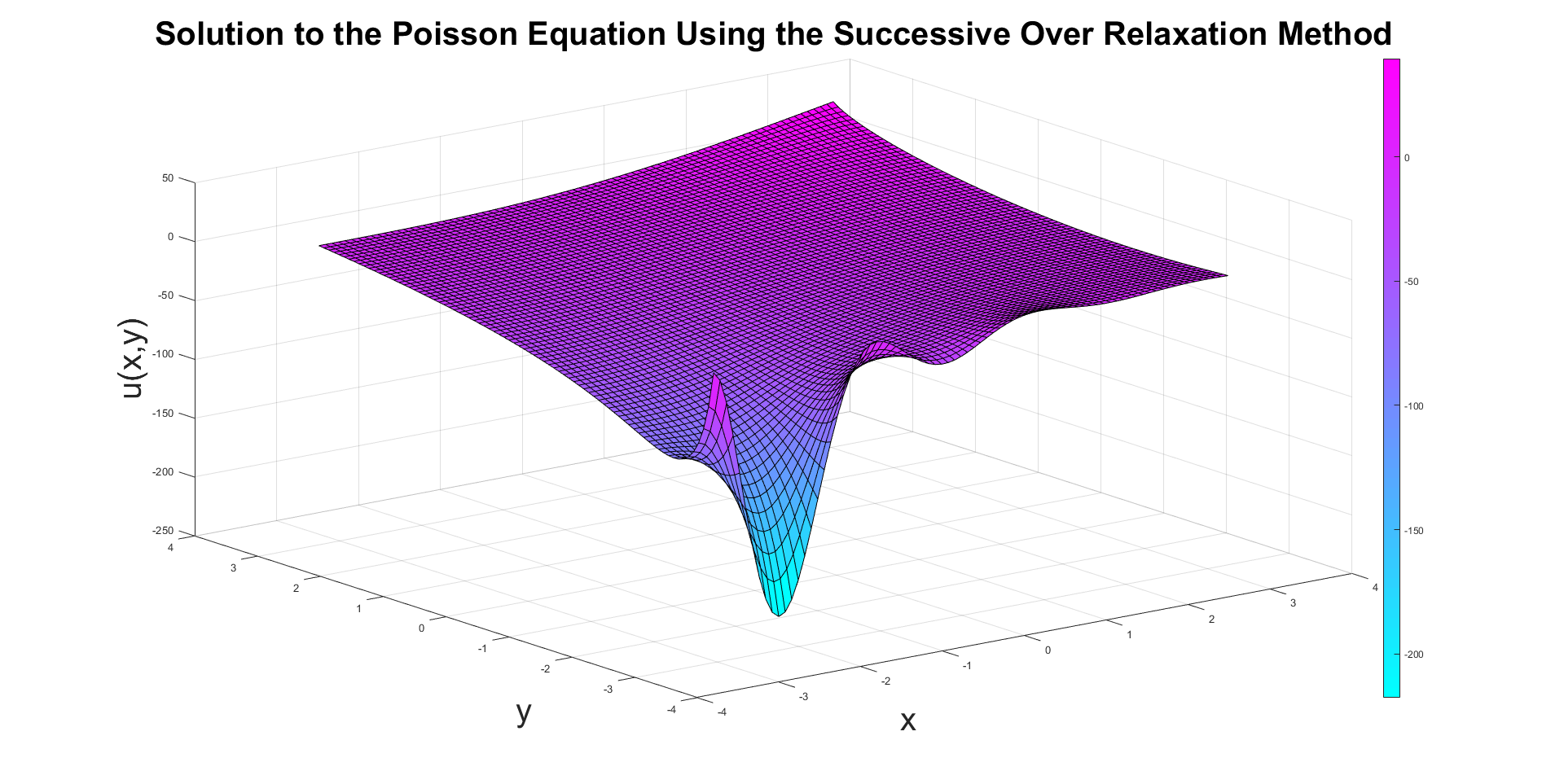
 

Grid size nx = 200 ny = 200 Grid size nx = 200 ny = 20

Elapsed time is 19.311346 seconds Elapsed time is 0.167798 seconds

Iterations performed 10226 Iterations performed 116

Magnitude of error 10^-7 Magnitude of error 10^-7

Grid size nx = 500 ny = 500 Grid size nx = 80 ny = 80 F(x,y) = 0

Elapsed time is 171.133197 seconds Elapsed time is 0.549643 seconds

Iterations performed 14212 Iterations performed 4245

Magnitude of error 10^-5 Magnitude of error 10^-7

As expected, the finer the mesh, the longer the calculation took to perform. Interestingly, the process for the nx=200, ny=20 was much faster than the one with nx=80, ny=80, however, the visual displays that it is not accurate, even compared with the grid of 20. The results for u(x,y) seemed to be almost identical between grids of 200 and 500. The solution also seemed to be identical visually between the two grids of 80 , one with F(x,y) = 0. This displays that the function slows the calculations, but did not impart any information on the surface. The benefits of Successive Over Relaxation are clear over Gauss-Seidel. Every calculation performed was significantly faster with the use of SOR, especially with finer meshes.