Solving Poisson's Equation Serially

Carlos Vazquez Gomez

January 27, 2019

1 Algorithm

Algorithm 1: Successive Approximation (Gauss-Seidel)

Input : 1. Matrix T with pre-set boundary values and internal values 0,
2. Matrix S with pre-set heat sink values
3. length x and
4. width y of rectangular area
5. $TRESH$ = error threshold
6. $UPDATE\_RATE$ = amount of grid updates between error re-calculations

Output: 1. Matrix T with all values solved

Function `NewPointVal`(x, y)**:**
  return
  $$\frac{-S^2_{x,y} + (point_{x-1,y} + point_{x+1,y})\Delta y^2 + (point_{x,y-1} + point_{x,y+1})\Delta x^2}{2(\Delta y^2 + \Delta x^2)};$$
end

$max\_error \leftarrow THRESH + 1$;
$count \leftarrow 0$;
$cur\_point \leftarrow firstinteriorpoint$;
while $max\_error > THRESH$ **do**
  $cur\_point.value =$ `NewPointVal(`$cur\_point.coord$`)`;
  $cur\_point = cur\_point.next\_interior\_point$;
  ; `/* Update max_error if necessary */`
  if $++count\%(UPDATE\_RATE * num\_points\_interior) == 0$ **then**
    $max\_error \leftarrow 0$;
    for *point in grid interior* **do**
      $this\_error \leftarrow |point.value$`NewPointVal(`$point.coord$`)`$|$;
      if $this\_error > max\_error$ **then**
        $max\_error = this\_error$;
    end for
end while

2 Verification

2.1 Second Order Accuracy

It was successfully verified that the algorithm is second order accurate. We plotted discrepancies (infinity normed) between the analytic and computed solution for a series of grids: 5x5, 11x11, 21x21, 41x41, and 51x51, and with error threshold 1e-12. The grid had dimensions X=2 and Y=1, with boundary conditions $T_{x,y} = xe^y, \forall$ (x,y) in interior, and source terms $S_{x,y} = xe^y$. These are shown in the blue line of Figure 1. Next, we plotted $(\Delta x)^2$ for each of the grids.

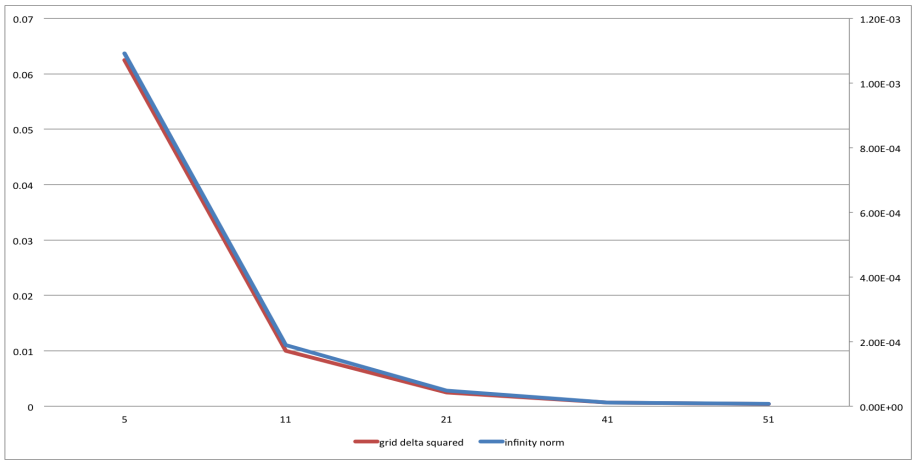

Figure 1: Comparing $(\Delta x)^2$ (red) with the maximum error (blue) at different grid divisions

Observe how in Figure 1, the y-axes for both series of data start at zero. Then, by geometric similarity, it becomes clear that the ratio of any two points in one line is almost identical to the ratio of the corresponding two points in the other line. In other words, $\frac{e_1}{e_2} \approx \frac{(\Delta x)^2}{(\Delta y)^2}$. Figure 2 provides another look at the normed ratios of both lines (the ratio is scaled so that the case 21x21 equals 1).

2.2 Possibilities for Grid Divisions

The code successfully converged for the following grids: 5x5, 11x11, 21x21, 41x41, 51x51, and 101x101. However, 201x201 began taking too much time (more than 5 minutes), and was therefore not considered successful. The code also works for heterogeneous divisions: 5x11, 5x21, 11x41, 21x101, and 201x5. Therefore, the only failure encountered (which was due to slow performance) was that of the grid 201x201.

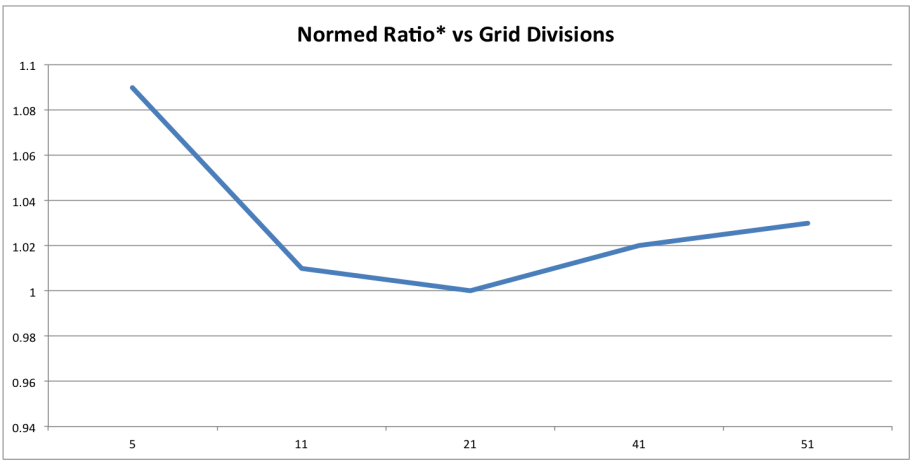

Figure 2: **Normed ratio* is the ratio of the errors to the $(\Delta x)^2$ for each grid, scaled so that the middle point is 1. Notice how close each ratio is to 1, which implies second order accuracy.

3 Problem Solving

With this algorithm, we were able to solve the following heat flow problem: Boundary temperatures = 0. Heat sink value = -0.2 everywhere.

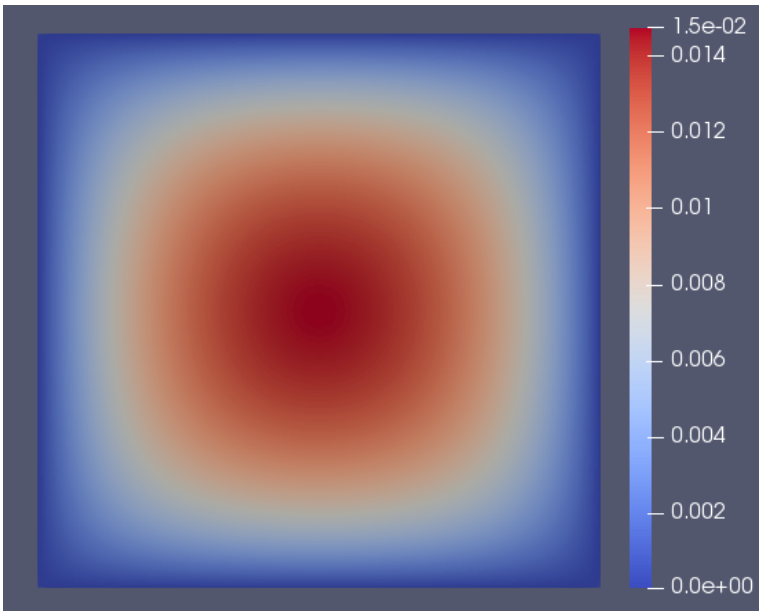

Figure 3: ParaView rendering of the temperatures

3.1 Convergence of Grid Point Calculations

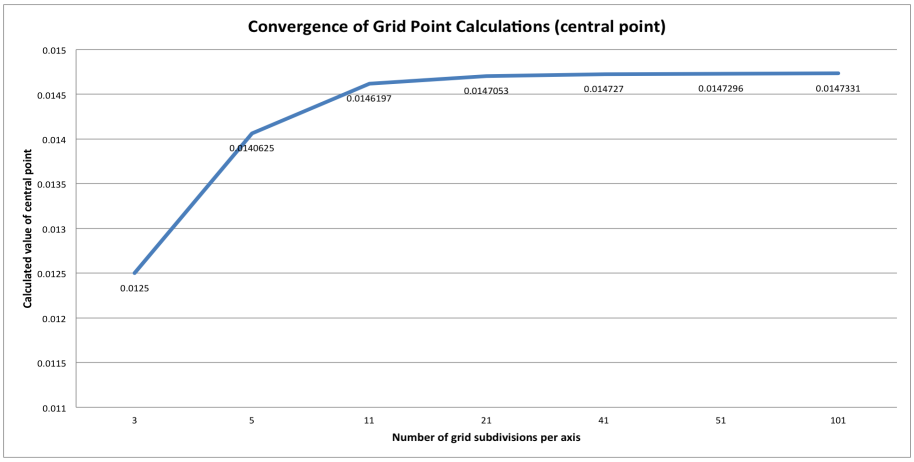

Figure 4: As the resolution of the grid gets finer, the temperature at the center approaches the neighborhood of 0.01475°

4 Performance

After waiting over 13 minutes for our 201x201 grid to converge to within error threshold 1e-12, we finally obtained the following data:

4.1 Speed versus Points

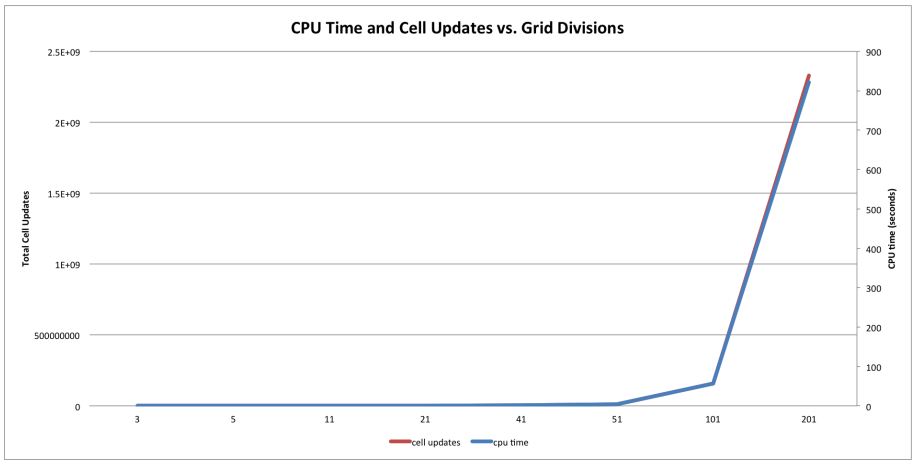

Figure 5: CPU time (blue) and total cell updates (red) are directly related. It is hard to tell, but there are two lines here!

This type of converging algorithm is very difficult to subject to complexity analysis, as it is not obvious how many updates it should take for values to converge. In order to get a better sense of the time complexity of our code, we plot the log time, as we have a hunch that it will be exponential time complexity.

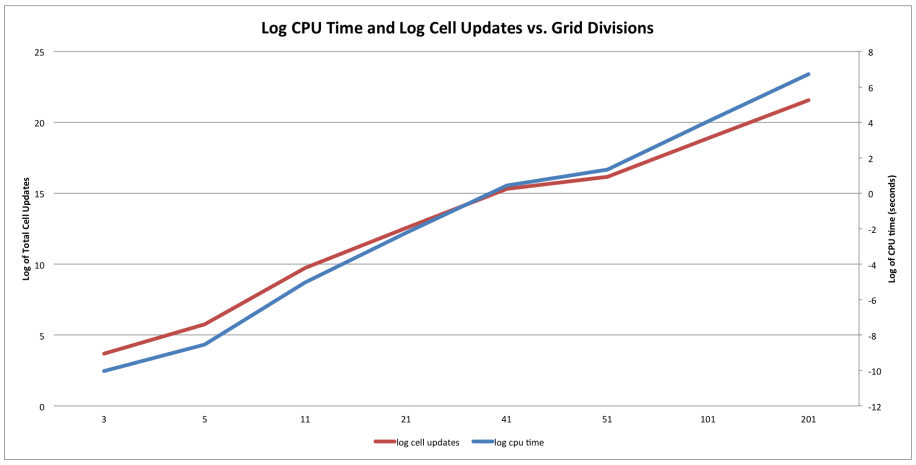

Figure 6: Log of time (blue) and log of cell updates (red) as functions of the cell divisions per axis.

From Figure 6, we can see that the algorithm is exponential in the number of subdivisions per axis, or approximately the square root of the number of points.