Numerical Methods

The most common numerical approach for Laplace's equation is the relaxation method, examined in detail here. Also used for the Poisson/Laplace eqn's are

1) Finite element method — good for engineering problems with complicated boundaries. Idea is to use a simple analytical approximate solution locally, then adjust boundaries to minimize errors.
2) Spectral methods — similar to Fourier series expansions of Sect. 3.3.1 of Griffiths.
3) Specialized methods for particular solutions.

Relaxation method

Start with mean value theorem,

$$V(\vec{r}) = \frac{1}{4\pi R^2} \oint_{\text{sphere}} V(\vec{r}')\,da' \quad \text{for a sphere at } \vec{r}.$$

Break up volume (or in 2D, area) into a regular mesh, so

$$V(\vec{r}) \rightarrow V_{i,j} \quad (2D)$$

$$\rightarrow V_{i,j,k} \quad (3D)$$

Approximate the MVT by requiring that at each point, the potential V should equal the average of its nearest neighbors.

Thus we require that

2-D: $V_{i,j} = \frac{1}{4}\left(V_{i+1,j} + V_{i,j+1} + V_{i-1,j} + V_{i,j-1}\right)$

3-D: $V_{i,j,k} = \frac{1}{6}\left(V_{i+1,j,k} + V_{i,j+1,k} + V_{i,j,k+1} + V_{i-1,j,k} + V_{i,j-1,k} + V_{i,j,k-1}\right)$

(Assumes a square grid, but can also be done in other coordinates.)

To get a useful numerical method from this, we approach this solution by successive approximations:

1) Set entire grid to zero.
2) Define boundary conditions by setting potentials as needed at the edges. (it's only a little harder if $E^{\perp}$ is specified instead of V.)
3) Given this (terrible) initial guess, cycle through all the points, taking averages as in the equations above. Repeat the procedure to get better and better guesses.

Pseudocode for 2-D problem, for n'th iteration:

```
Repeat as needed {
FOR i = first_i to last_i - 1        ; skip boundaries
    FOR j = first_j + last_j - 1
        $V^{n+1}_{i,j} = \frac{1}{4}\left[V^n_{i+1,j} + V^n_{i,j+1} + V^n_{i-1,j} + V^n_{i,j-1}\right]$
}
```

This is <u>Jacobi's method</u>. In practice it is simpler to put the results back into the original array as soon as we have them. This also converges faster. It means that points with i-1, j-1, or k-1 come from the current iteration, not the previous one.

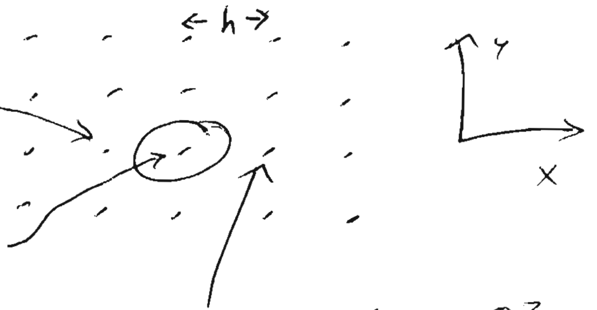
This variation is the Gauss-Seidel method:

$$V_{i,j}^{n+1} = \frac{1}{4}\left[V_{i+1,j}^{n} + V_{i,j+1}^{n} + V_{i-1,j}^{n+1} + V_{i,j-1}^{n+1}\right]$$

(in 2-D)

Another way to see where this stuff comes from is to consider the Taylor expansion of V at an arbitrary point on a 2-D mesh:

$$V_{i-1,j} \simeq V_{i,j} + \frac{\partial V}{\partial x}(-h) + \frac{1}{2}\frac{\partial^2 V}{\partial x^2}(-h)^2 + \frac{1}{3!}\frac{\partial^3 V}{\partial x^3}(-h)^3 \ldots$$



$$V_{i+1,j} \simeq V_{i,j} + \frac{\partial V}{\partial x}h + \frac{1}{2}\frac{\partial^2 V}{\partial x^2}h^2 \ldots$$

Adding up the two horizontal neighbors plus the two vertical ones as well,

$$\sum_{4\text{ neighbors}} V = 4V_{i,j} + \underbrace{(\nabla^2 V)}_{=0\text{ by Laplace's eq'n.}} h^2 + \mathcal{O}(h^4)$$

So $V_{i,j} = \frac{1}{4}\sum_{\text{neighbors}} V + \mathcal{O}(h^4)$.

$\uparrow$ mesh can be fairly coarse and still give good results!

<u>Overrelaxation</u> (Couch potato method ??)

To improve convergence when still far from the correct solution, can intentionally overcorrect by a factor $w > 1$. To avoid multiplying by w, just subtract off $(w-1)$(old result). This gives the preferred integration formula,

$$\boxed{V_{i,j}^{n+1} = \frac{w}{4}\left[V_{i+1,j}^{n} + V_{i,j+1}^{n} + V_{i-1,j}^{n} + V_{i,j-1}^{n}\right] + (1-w)\,V_{i,j}^{n}}$$

Gauss-Seidel with overrelaxation (2-D)

In 3-D, similar with 6 terms and $w/6$.

For $w > 2$ this is unstable. Typical optimum values are 1.4 - 1.8. Can check by monitoring convergence, looking at point with max. change between n'th and n+1'st iteration.

"Program" relax.mcd, following, implements this for a simple rectangular region, using Mathcad.

There is also a C-language sample program, "relax.c", available in the Resources section of the P3201 web page.

Relax.mcd

Solves Laplace's equation for a square array in 2-D by the relaxation method, including an "overrelaxation" parameter. Last modified 10/21/09 by E. Eyler

Define the "left" (x=0) and right (x=SIZE-1) boundary conditions using Kronecker delta functions, setting all other points to V=0.. Just for fun, set the left border to a sinusoidal potential, but the right border to a constant 2V.

$$\mathrm{SIZE} := 25$$

$$f(x,y) := \delta(x,0)\cdot\left(1+\cos\left(\frac{2\cdot\pi y}{\mathrm{SIZE}}\right)\right) + 2\,\delta(x,\mathrm{SIZE}-1)$$

Now create a square matrix with these boundary conditions.

$$V := \mathrm{matrix}(\mathrm{SIZE},\mathrm{SIZE},f)$$

Define the "bottom" (y=0) and top boundaries to be at ground, 0 V. Adding them to the expression above would have caused unsightly 'spikes' in the corners, although it would not have affected the actual results of the calculation. This time we will use a different method, the Mathcad range variable, to set the values:

$$x := 0\,..\,\mathrm{SIZE}-1$$

$$V_{x,0} := 0 \qquad\qquad V_{x,\mathrm{SIZE}-1} := 0$$

Display the corner of the matrix near the origin, noting rows are indexed by x, and columns by y:

V =

	0	1	2	3	4	5
0	0	1.969	1.876	1.729	1.536	1.309
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0
7	0	0	0	0	0	0

Define the relaxation calculation. Define it as a function that operates on an arbitrary array Φ, returning both the result and the number of iterations required to achieve the desired convergence level, tol, between adjacent iterations. The overrelaxation parameter is w. Note that for an angled right boundary, the looping condition on y would require modification.

```
relax(Φ, tol, w) :=
    iters ← 0
    maxdiff ← $1 \times 10^6$
    while  maxdiff > tol
        iters ← iters + 1
        maxdiff ← 0
        for  $x \in 1 .. \text{rows}(\Phi) - 2$
            for  $y \in 1 .. \text{cols}(\Phi) - 2$
                newval ← $\frac{w}{4}\left(\Phi_{x-1,y} + \Phi_{x,y-1} + \Phi_{x+1,y} + \Phi_{x,y+1}\right) + (1-w)\cdot\Phi_{x,y}$
                maxdiff ← $\left|newval - \Phi_{x,y}\right|$  if  $\left|newval - \Phi_{x,y}\right| > maxdiff$
                $\Phi_{x,y}$ ← newval
    return $\begin{pmatrix}\Phi \\ iters\end{pmatrix}$
```

Finally, carry out the calculation on the initialized array, V, and print part of the result.

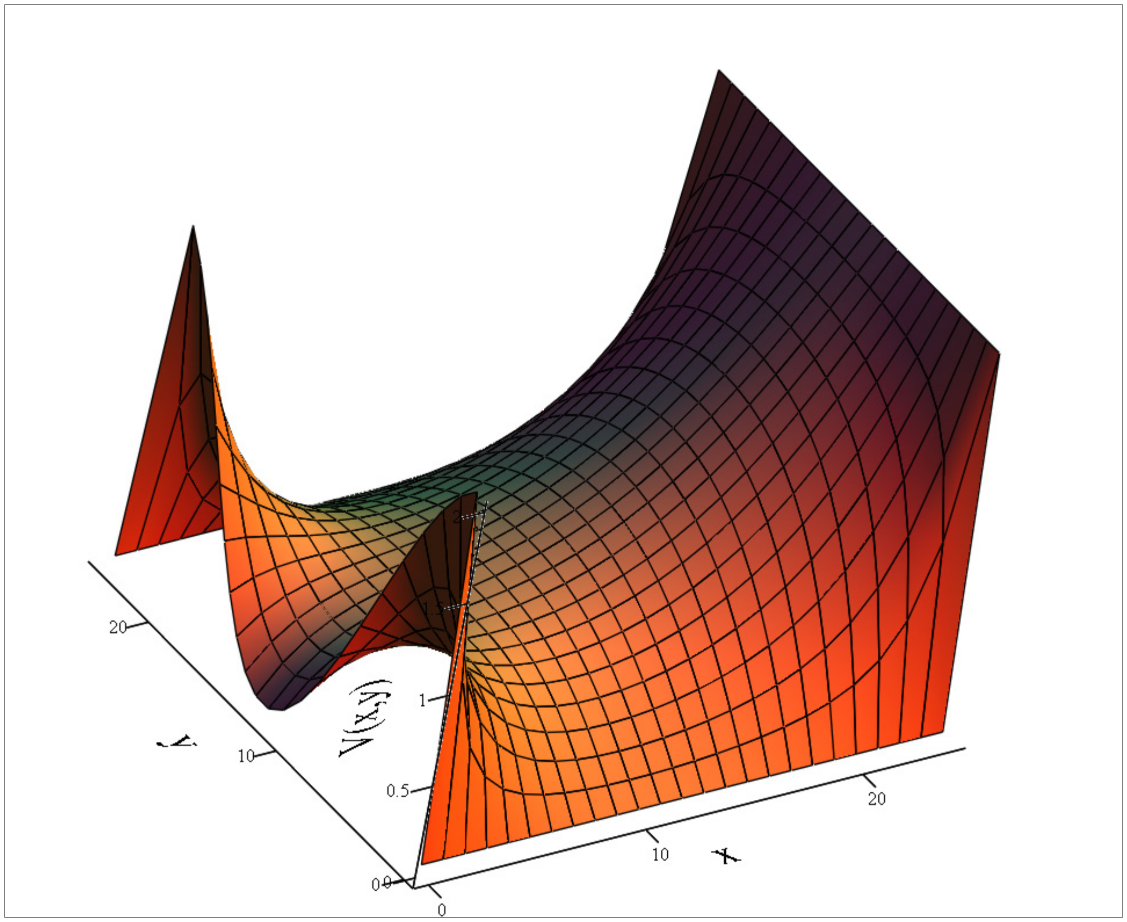
$$\begin{pmatrix} V \\ iters \end{pmatrix} := \text{relax}\left(V, 10^{-4}, 1.5\right)$$

$iters = 125$ Numer of iterations that were required.

V =

	0	1	2	3	4	5
0	0	1.969	1.876	1.729	1.536	1.309
1	0	0.924	1.217	1.268	1.205	1.08
2	0	0.511	0.8	0.921	0.936	0.884
3	0	0.321	0.55	0.681	0.733	0.727
4	0	0.221	0.4	0.52	0.586	0.609

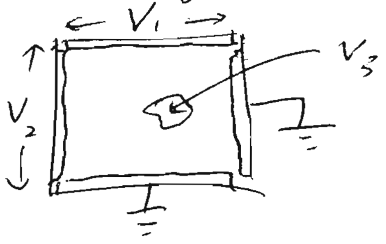
Finally, plot the results.



V

Additivity property of solutions

Say we have an apparatus with several conducting electrodes that can be set at various voltages:



(always same physical bdrys, but values change.)

If $V^{(a)}(\vec{r})$ and $V^{(b)}(\vec{r})$ are solutions, so is ~~a~~ $\alpha V^{(a)} + \beta V^{(b)}$.

This suggests a very efficient approach:

1) Set $V_1 = 1V$, all others to zero. Solve, call this $V^{(1)}$.

2) Set $V_2 = 1V$, all others to zero. Call solution $V^{(2)}$.

Now, if $V_1 = 13V$, $V_2 = 0$, and $V_3 = 459V$, solution is just

$$V^{13,0,459} = 13\,V^{(1)} + 0\,V^{(2)} + 459\,V^{(3)}$$

Justification:

$13\,V^{(1)}$ sets $V_1 = 13V$, all other bdry's to 0.

$459\,V^{(3)}$ sets $V_3 = 459V$, all others zero

$\Rightarrow$ sum does just what we want.