The Gauss-Seidel method

1. Use four steps of the Gauss-Seidel method to approximate a solution to a system of linear equations $A\mathbf{u} = \mathbf{v}$ where

$$A = \begin{pmatrix} 5 & 1 & -2 \\ 1 & 10 & 2 \\ -2 & 2 & 10 \end{pmatrix} \text{ and } \mathbf{v} = \begin{pmatrix} -1.5 \\ 0.2 \\ -1.0 \end{pmatrix}.$$

Answer: $\mathbf{u}_0 = A_{\text{diag}}^{-1}\mathbf{v} = \begin{pmatrix} -0.3 \\ 0.02 \\ -0.1 \end{pmatrix}$, and $\mathbf{u}_1 = \begin{pmatrix} -0.344 \\ 0.0744 \\ -0.18368 \end{pmatrix}$, $\mathbf{u}_2 = \begin{pmatrix} -0.388352 \\ 0.0955712 \\ -0.19678464 \end{pmatrix}$,

$\mathbf{u}_3 = \begin{pmatrix} -0.397828096 \\ 0.0991397376 \\ -0.19939356672 \end{pmatrix}$, $\mathbf{u}_4 = \begin{pmatrix} -0.399585374208 \\ 0.0998372507648 \\ -0.19988452499456 \end{pmatrix}$.

2. The solution to Question 1 is the vector $\mathbf{u} = \begin{pmatrix} -0.4 \\ 0.1 \\ -0.2 \end{pmatrix}$. What is $\|\mathbf{u} - \mathbf{u}_k\|_2$ for each of these approximations?

Answer: 0.1625, 0.06370, 0.01287, 0.002413, 0.0004601

3. The errors in each approximation in Question 2 seem to drop by approximately a constant with each step. What would be your estimate as to the reduction in this error?

Answer: The appears to drop by a value around 5 and 5.5.

4. Verify your response to Question 3 by running the following Matlab code:

```
A = [5 1 -2; 1 10 2; -2 2 10];
v = [-1.5 0.2 -1.0]';
u = [-0.4 0.1 -0.2]';           # The exact solution to A*u = v
Adiag = diag(diag(A));
Aoff  = A - Adiag;
InvAdiag = Adiag^-1;
u1 = InvAdiag*v;

for i = 1:30
    u0 = u1;

    for j = 1:3
        u1(j) = InvAdiag(j,j)*(v(j) - Aoff(j,:)*u1);
    end

    norm( u0 - u )/norm( u1 - u )
end
```

5. What is happening in the last few steps of the for loop in Question 4?

6. What are the efficiencies introduced into the following code when compared to that in Question 4?

```
A = [5 1 -2; 1 10 2; -2 2 10];
v = [-1.5 0.2 -1.0]';
u = [-0.4 0.1 -0.2]';           # The exact solution to A*u = v
Aoff  = A;
u1 = v;

for j = 1:3
    Aoff(j, j) = 0.0;
    u1(j) = u1(j)/A(j,j);
end

for i = 1:30
    u0 = u1;

    for j = 1:3
        u1(j) = (v(j) - Aoff(j,:)*u1)/A(j,j);
    end

    norm( u0 - u )/norm( u1 - u )
end
```

Answer: There are no unnecessary intermediate data structures (vectors or arrays) and thus resulting in an O(n^2) saving in memory. Also, there is no unnecessary calculation of the inverse, and therefore a savings in run time of O(n^3). There are other efficiencies as well, such as dividing by the diagonal entries as opposed to multiplying by the inverse, which is an O(n^2) savings with each iteration of the loop.