MATH 561: Numerical Analysis I

Instructor: Prof. Wolfgang Bangerth
bangerth@colostate.edu

Answers for homework assignment 2

Problem 1 (LU decomposition). If you follow the steps for computing the LU decomposition discussed in class, you will end up with the following two factors:

$$A = LU, \qquad L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{12} & 0 & 0 \\ \frac{1}{3} & \frac{1}{12} & \frac{1}{180} & 0 \\ \frac{1}{4} & \frac{3}{40} & \frac{1}{120} & \frac{1}{2800} \end{pmatrix}, \qquad U = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ 0 & 1 & 1 & \frac{9}{10} \\ 0 & 0 & 1 & \frac{3}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

To solve the linear system $Ax = b$, one would first solve $Ly = b$ through forward substitution. This yields $y = (1, 18, 210, 1820)^T$. With this, we can solve $Ux = y$ by backward substitution, and this yields $x = (-64, 900, -2520, 1820)^T$.

It is easy to verify that Ax is indeed equal to b.

Problem 2 (LU decomposition). A program that implements the steps will of course lead to the exact same results as above. Being able to compute an LU decomposition by hand, as required for Problem 1, is useful when debugging an implementation, because you can follow what your program does step-by-step, and verify on paper that it is correct.

Problem 3 (Positive definite matrices). The key to understand the solution of this problem is to realize that

$$x^T A x = \sum_i \sum_j x_i A_{ij} x_j = \sum_i \sum_j x_j A_{ij} x_i = \sum_j \sum_i x_j A_{ij} x_i = \sum_j \sum_i x_j (A^T)_{ji} x_i = x^T A^T x$$

This is true for *any* matrix.

To show the first part of the problem, assume that A is positive definite. Then $x^T A x > 0$ for any vector $x \neq 0$. But then obviously, given the above identity, we also have that $x^T A^T x > 0$. This means that $x^T A^s x = \frac{1}{2}(x^T A x + x^T A^T x) > 0$. In other words, A^s is also positive definite.

The other way around is no more complicated. Assume that A^s is positive definite, i.e., $x^T A^s x > 0$. Using the definition, we know that $\frac{1}{2}(x^T A x + x^T A^T x) > 0$. But the definition above tells us that $x^T A x = x^T A^T x$, so the two terms in the sum $\frac{1}{2}(x^T A x + x^T A^T x)$ are really the same, i.e., $0 < \frac{1}{2}(x^T A x + x^T A^T x) = \frac{1}{2}(x^T A x + x^T A x) = x^T A x$. Consequently, if A^s is positive definite, then so is A.

Problem 4 (Norms on $\mathbb{R}^n$). To be a norm, a functional has to satisfy all three of the listed conditions. On the other hand, if it violates at least one of them, it is not a norm. It is important to realize that each of these conditions speaks about "for all vectors" (in part a, the norm has to be nonzero for all nonzero vectors), so the condition is already violated if we can find only a *single* vector (or pair of vectors) for which it is not true. If we can show that a functional violates one of the three conditions, we do not need to check the other conditions, since we are already sure that it can't be a norm.

What this means is that if you suspect that something is not a norm, then it comes down to trying to find a cleverly chosen vector or pair of vectors that violate the (in)equality you think isn't true. The cases below are examples of this approach.

a) $\|x\| = \max\{|x_1|, |x_2|, |x_3|, \ldots, |x_n|\}$ is indeed a norm. This is the l_∞ norm and for which we have shown the norm axioms in class.

b) $\|x\| = \max\{|x_2|, |x_3|, |x_4|, \ldots, |x_n|\}$ is not a norm, since $\|x\| = 0$ for the vector $x = (1, 0, 0, \ldots, 0)$, even though $x \neq 0$. It therefore violates condition 1. What this shows is that a norm really needs to somehow evaluate *all* elements of a vector; if it doesn't, you can choose a vector that is nonzero only in the one element that the norm doesn't evaluate, and you would still get a zero norm.

c) $\sum_{i=1}^{n} |x_i|^3$ looks deceptively like the l_3 norm (which would take the third root of the whole expression), but without the third root it is not a norm since $\|\lambda x\| = |\lambda|^3 \|x\| \neq |\lambda| \|x\|$, i.e. the second condition is violated.

d) $\left(\sum_{i=1}^{n} |x_i|^{1/2}\right)^2$ is not a norm, since it violates the triangle inequality. If it was, we would call it the $l_{1/2}$ norm and it would fit into the l_p norm family defined by $\|x\| = (\sum_i |x_i|^p)^{1/p}$; in reality, people do indeed call it the $l_{1/2}$ norm, but everyone who uses this terms is keenly aware of the fact that this is not in fact a proper use of the term. In particular, it can be shown that the l_p norms only satisfy the norm conditions for $1 \leq p \leq \infty$, but not for $p < 1$.

Many pairs of vectors x, y can be chosen for which the triangle inequality does not hold. One simple example is $x = (0, 1), y = (1, 0)$, for which $\|x + y\| = (\sqrt{0+1} + \sqrt{1+0})^2 = 4$, but $\|x\| + \|y\| = (\sqrt{0} + \sqrt{1})^2 + (\sqrt{1} + \sqrt{0})^2 = 2 < \|x + y\|$.

e) $\max\{|x_1 - x_2|, |x_1 + x_2|, |x_3|, |x_4|, \ldots, |x_n|\}$ is a norm. To show this, we have to verify all three conditions:

(i) Positivity: If the maximum of all these (non-negative) expressions is zero, then each of the individual expressions must be zero. From $|x_1 - x_2| = 0, |x_1 + x_2| = 0$, we can conclude that $x_1 = x_2, x_1 = -x_2$, which can only be true if $x_1 = x_2 = 0$. That the other elements of x are all zero is obvious. This condition is therefore satisfied.

(ii) Linearity:

$$\begin{aligned}
\|\lambda x\| &= \max\{|\lambda x_1 - \lambda x_2|, |\lambda x_1 + \lambda x_2|, |\lambda x_3|, |\lambda x_4|, \ldots, |\lambda x_n|\} \\
&= \max\{|\lambda||x_1 - x_2|, |\lambda||x_1 + x_2|, |\lambda||x_3|, |\lambda||x_4|, \ldots, |\lambda||x_n|\} \\
&= |\lambda| \max\{|x_1 - x_2|, |x_1 + x_2|, |x_3|, |x_4|, \ldots, |x_n|\} \\
&= |\lambda| \|x\|.
\end{aligned}$$

This condition is therefore also satisfied.

(iii) Triangle inequality:

$$\|x + y\| = \max\{|x_1 + y_1 - x_2 - y_2|, |x_1 + y_1 + x_2 + y_2|, |x_3 + y_3|, |x_4 + y_4|, \ldots, |x_n + y_n|\}.$$

Consider the first term: since the triangle inequality also holds for the magnitude of a single scalar value, we have $|x_1 + y_1 - x_2 - y_2| < |x_1 - x_2| + |y_1 - y_2|$. Similary for the second term. For all the other terms, we have $|x_i + y_i| < |x_i| + |y_i|$. We therefore have

$$\begin{aligned}
\|x + y\| \leq \max\{&|x_1 - x_2| + |y_1 - y_2|, |x_1 + x_2| + |y_1 y_2|, \\
&|x_3| + |y_3|, |x_4| + |y_4|, \ldots, |x_n| + |y_n|\}.
\end{aligned}$$

Finally, it is important to realize that

$$\max\{a_1 + b_1, a_2 + b_2\} \leq \max\{a_1, a_2\} + \max\{b_1, b_2\},$$

(and similarly if the maximum is taken over more than two elements). This may sound obvious, but isn't really (take for example $a_1 = 2, a_2 = 1, b_1 = 1, b_2 = 2$ to see that the two sides are really not equal). With this, we can establish that

$$\|x + y\| \leq \|x\| + \|y\|.$$

Consequently, this is a norm.

f) $\sum_{i=1}^{n} 2^{-i}|x_i|$ is indeed a norm. Positivity and linearity are easily established and not shown here. As for the triangle inequality, the following argument holds:

$$\begin{aligned}\|x+y\| &= \sum_{i=1}^{n} 2^{-i}|x_i+y_i| < \sum_{i=1}^{n} 2^{-i}(|x_i|+|y_i|) \\ &= \sum_{i=1}^{n} 2^{-i}|x_i| + \sum_{i=1}^{n} 2^{-i}|y_i| = \|x\| + \|y\|.\end{aligned}$$

Problem 5 (Equivalence of norms on $\mathbb{R}^n$). Let us show the left inequality first. Let v_p be the element of v with the biggest magnitude, i.e. $|v_p| \geq |v_i|, i = 1, \ldots, n$. Then

$$\|v\|_1 = \sum_{i=1}^{n} |v_i| = |v_p| + \sum_{i=1, i\neq p}^{n} |v_i|.$$

The sum on the right only adds up positive or zero elements, so $\|v\|_1 \geq |v_p|$. However, since v_p is the element with the biggest magnitude, we have that $\|v\|_\infty = |v_p|$, and therefore

$$\|v\|_1 \geq \|v\|_\infty.$$

This establishes the left inequality with a constant $c = 1$.

As for the right inequality, we use that $|v_i| \leq |v_p|$.

$$\|v\|_1 = \sum_{i=1}^{n} |v_i| \leq \sum_{i=1}^{n} |v_p|.$$

On the right, we sum n times over the same value, $|v_p|$ that does not depend on the counting index i at all. We therefore get:

$$\|v\|_1 \leq n|v_p|.$$

This establishes the right inequality with a constant $C = n$.

Note: In some of your homeworks, there were descriptions of "constants" that depended in various ways on the vector v. That was not the purpose of the questions: we are looking for constants c, C such that the inequality holds *for all* vectors v. One way to state this is to reformulate the equations and to ask whether there are constants c, C such that

$$\begin{aligned}\max_{v\in\mathbb{R}^n} \frac{\|v\|_\infty}{\|v\|_1} &\leq \frac{1}{c}, \\ \max_{v\in\mathbb{R}^n} \frac{\|v\|_1}{\|v\|_\infty} &\leq C.\end{aligned}$$

Taking the maximum over all possible vectors v makes it clear that the right hand side of the equations can't depend on the particular entries of v.

Problem 6 (Alternative vector norms). If A is a symmetric and positive definite matrix, then this in particular implies that (i) all eigenvalues are positive, (ii) the eigenvectors form a complete set of orthogonal and normalized vectors, and (iii) $x^T Ax > 0$ for any vector $x \neq 0$.

Using these, we then have to verify all three conditions:

- Non-negativity: This one is easy. Because $x^T Ax > 0$ for any vector $x \neq 0$, we obviously have that $\|x\|_A = \sqrt{x^T Ax} > 0$ for any vector $x \neq 0$. Furthermore, for $x = 0$, it is clear that we also get $\|x\|_A = 0$.

- Scalability: This one is easy as well: $\|\lambda x\|_A = \sqrt{(\lambda x)^T A(\lambda x)} = \sqrt{\lambda^2 x^T A x}|\lambda|\sqrt{x^T A x} = |\lambda|\|x\|_A$.
- Triangle inequality: This one is a bit trickier. It uses the fact that the eigenvectors v_i of A form a complete, orthonormal basis of $\mathbb{R}^n$. Consequently, we can write any vector as a linear combination of these eigenvectors. Specifically, for given vectors x, y, let us write them as $x = \sum_{i=1}^n \xi_i v_i$ and $y = \sum_{i=1}^n \eta_i v_i$. Then we have the following equalities:

$$\begin{aligned}
\|x+y\|_A^2 &= (x+y)^T A(x+y) \\
&= \left(\sum_i \xi_i v_i + \sum_j \eta_j v_j\right)^T A \left(\sum_k \xi_k v_k + \sum_l \eta_j v_l\right) \\
&= \left(\sum_i \xi_i v_i + \sum_j \eta_j v_j\right)^T \left(\sum_k \xi_k A v_k + \sum_l \eta_j A v_l\right) \\
&= \left(\sum_i \xi_i v_i + \sum_j \eta_j v_j\right)^T \left(\sum_k \lambda_k \xi_k v_k + \sum_l \eta_j \lambda_l v_l\right) \\
&= \sum_i \sum_k \xi_i \lambda_k \xi_k v_i^T v_k + \sum_j \sum_k \eta_j \lambda_k \xi_k v_j^T v_k + \sum_i \sum_l \xi_i \lambda_l \eta_l v_i^T v_l + \sum_j \sum_l \eta_j \lambda_l \eta_l v_j^T v_l.
\end{aligned}$$

But the eigenvectors are orthonormal, so products of eigenvectors yield results such as $v_i^T v_k = \delta_{ik}$. This yields

$$\|x+y\|_A^2 = \sum_i \lambda_i \xi_i^2 + \sum_j \lambda_j \xi_j \eta_j + \sum_i \lambda_i \xi_i \eta_i + \sum_j \lambda_j \eta_j^2.$$

Let us introduce "scaled" values $\tilde{\xi}_i = \sqrt{\lambda_i}\xi_i$, $\tilde{\eta}_i = \sqrt{\lambda_i}\eta_i$. (We can do this because all of the eigenvalues λ_i are positive.) Then

$$\|x+y\|_A^2 = \sum_i \tilde{\xi}_i^2 + \sum_j \tilde{\xi}_j \tilde{\eta}_j + \sum_i \tilde{\xi}_i \tilde{\eta}_i + \sum_j \tilde{\eta}_j^2.$$

If we now group all of these $\tilde{\xi}_i$ into an n-dimensional vector $\tilde{\xi}$, and do the same for $\tilde{\eta}$, then it is easy to see that we in fact have

$$\|x+y\|_A^2 = \|\tilde{\xi} + \tilde{\eta}\|^2$$

where the right hand side uses the common l_2 norm. But we know that that is a norm, so

$$\|x+y\|_A \leq \|\tilde{\xi}\| + \|\tilde{\eta}\|.$$

It is easy to show, using the tools above, that in fact $\|\tilde{\xi}\| = \|x\|_A$ and $\|\tilde{\eta}\| = \|y\|_A$, which yields the desired result.

Problem 7 (Jacobi iteration). For this problem, we are asked to compute the following iteration:

$$x_{k+1} = x_k + D^{-1} r_k$$

where the residual r_k is defined as

$$r_k = b - A x_k,$$

In each step, we will therefore compute this residual and add the values of the residual scaled by the inverse elements of the diagonal of A to x_k to get x_{k+1}. Because we don't need x_k any more once we have computed the corresponding elements of x_{k+1}, we simply reuse the memory space for it.

In order to compute the residual, it is important to realize that we don't really have to store the matrix at all. We only need to know how to multiply it with x_k. Given the description of the elements of A, we have that

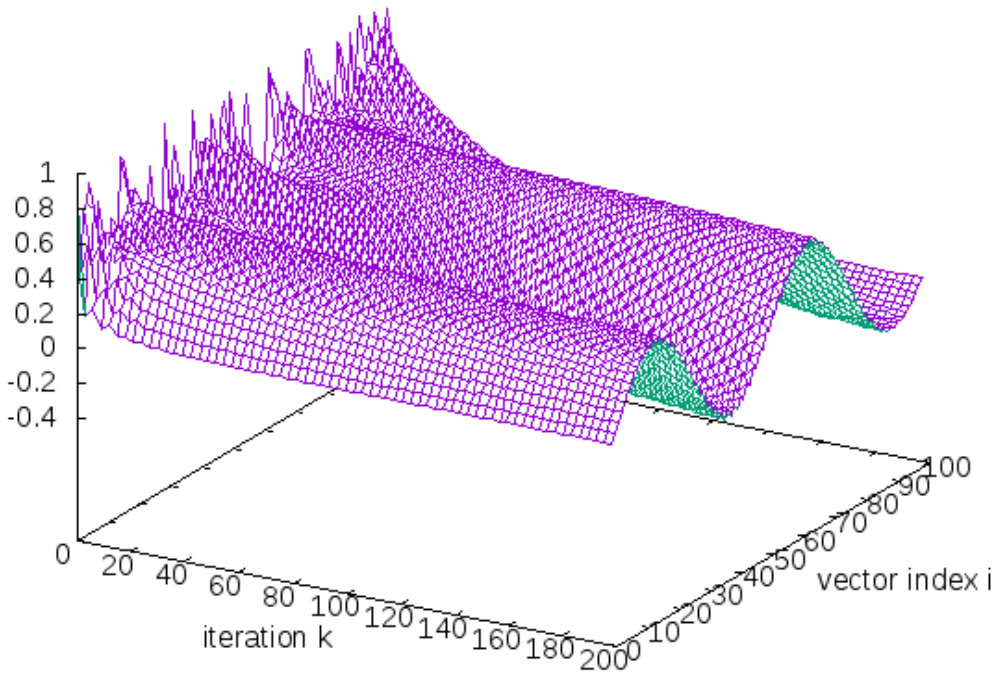
$$\begin{aligned}
(Ax_k)_1 &= 2.01(x_k)_1 - (x_k)_2, \\
(Ax_k)_i &= 2.01(x_k)_i - (x_k)_{i-1} - (x_k)_{i+1}, \qquad 2 \le i \le n, \\
(Ax_k)_n &= 2.01(x_k)_n - (x_k)_{n-1}.
\end{aligned}$$

Using this in a program then reads as follows:

```
#include <iostream>
#include <cmath>

const unsigned int N = 100;

double x[N];
double residual[N];

double b[N];


int main ()
{
  // initialize x with random values between
  // 0 and 1. set b to the values given in
  // the problem description
  for (unsigned int i=0; i<N; ++i)
    {
      x[i] = 1.*rand() / RAND_MAX;
      b[i] = std::sin(2.*3.14159265358*i/50)/100;
    }

  // do the iterations
  for (unsigned int it = 0; it<=200; ++it)
    {
      // every third iteration, produce some
      // output
      if (it % 3 == 0)
        {
          for (unsigned int i=0; i<N; ++i)
            std::cout << it << ' ' << i << ' '
                      << x[i]
                      << std::endl;
          std::cout << std::endl;
        }

      // compute the residual -(Ax-b) as
      // described above
      residual[0] = - (2.01*x[0] - x[1] - b[0]);
```

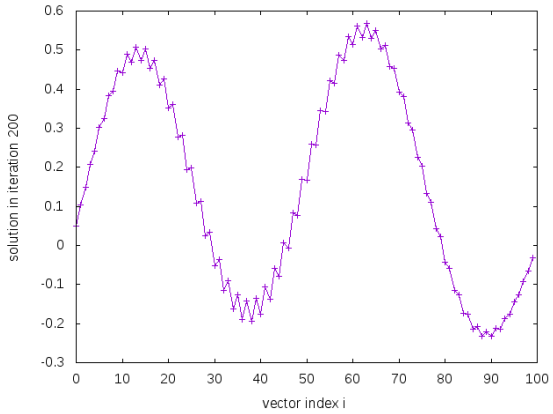
```
        for (unsigned int i=1; i<N-1; ++i)
          residual[i] = -(2.01*x[i] - x[i-1] - x[i+1] - b[i]);
        residual[N-1] = -(2.01*x[N-1] - x[N-2] - b[N-1]);

        // then update x
        for (unsigned int i=0; i<N; ++i)
          x[i] = x[i] + residual[i]/2.01;
    }
}
```

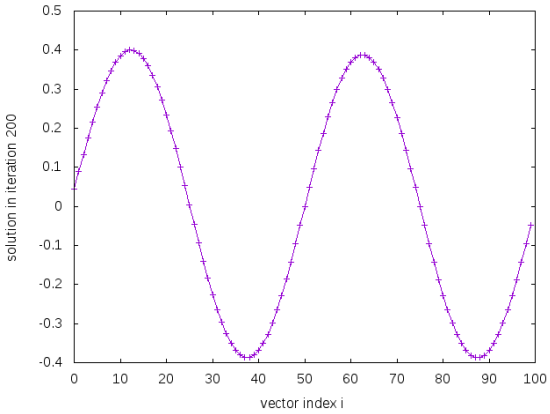
The program outputs the 100 elements of x_k for every third iteration k. If we plot them, we get this:



In other words, we start out with a completely random vector x_0 (at the left edge), but over time (moving from the left to the right) the solution converges to something smooth and sine-like. However, even at the last iteration x_{200}, there is still some jitter. It will go away and we will recover the final solution after significantly more than 200 iterations, but as was proven in class the iteration will at least converge.

As a sidenote, if you go through the code you will realize that in each iteration we perform exactly $2N$ scalar multiplications (or divisions, for that matter, but we usually count them together), where $N = 100$ is the vector size. Even if we did 2000 iterations, we would still have performed only $2000 \cdot 2N = 400,000$ multiplications. It is easy to check that the average update $|(\delta x_k)_i|$ to the vector element i in iteration k is around $1.6 \cdot 10^{-6}$ in the 2000th iteration (while it was 0.3 in the first iteration), so we can confidently say that after 2000 iterations the solution has mostly converged and is fairly accurate. The following two figures illustrate this, showing the elements of the solution vectors $x^{(200)}$ and $x^{(2000)}$:





Clearly, the solution after 200 iterations is "not quite there yet", as we can see after knowing how it looks after 2000 iterations. We do have to acknowledge that convergence is pretty slow – we shouldn't have to do 2000 iterations for linear systems with 100 elements.

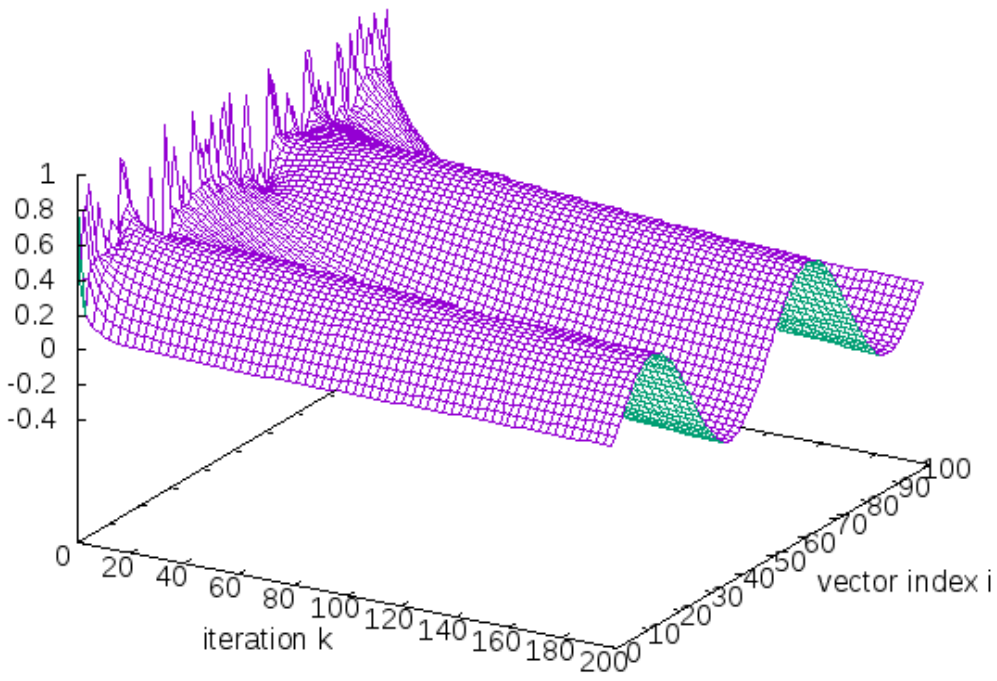
On the other hand, if we had used a Cholesky decomposition, we would have used roughly $\frac{1}{6}N^3 = 160,000$ multiplications. For this particular case, a "direct" solution using the Cholesky decomposition would therefore have been faster. This would probably have changed for larger vector sizes: even if we allow that the number of Jacobi iterations grows linearly with the number N of unknowns in x and that the number of multiplications per iteration is $2N$, we get a quadratic growth of the numerical work for the Jacobi iteration, while the work grows like N^3 for direct solvers like Cholesky or the LU decomposition. You are encouraged to make experiments how large your vectors need to be before Jacobi becomes faster than a Cholesky decomposition.

Problem 8 (Gauss-Seidel iteration). The exercise is easily repeated for the Gauss-Seidel method. Using the trick discussed in class where we simply overwrite the previous solution vector with the current one, it is easy to adjust the program from the previous problem to the current one:

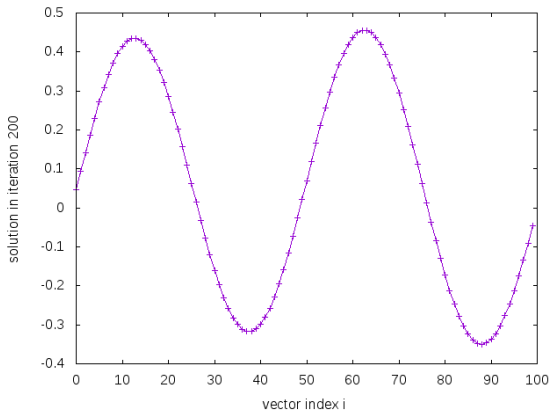
```
#include <iostream>
#include <cmath>

const unsigned int N = 100;

double x[N];
double residual[N];

double b[N];


int main ()
{
  // initialize x with random values between
  // 0 and 1. set b to the values given in
  // the problem description
  for (unsigned int i=0; i<N; ++i)
    {
      x[i] = 1.*rand() / RAND_MAX;
      b[i] = std::sin(2.*3.14159265358*i/50)/100;
    }
```

```
  // do the iterations
  for (unsigned int it = 0; it<=200; ++it)
    {
      // every third iteration, produce some
      // output
      if (it % 3 == 0)
        {
          for (unsigned int i=0; i<N; ++i)
            std::cout << it << ' ' << i << ' '
                      << x[i]
                      << std::endl;
          std::cout << std::endl;
        }

      // update the solution vector as discussed in class, by just
      // overwriting the previous value
      //
      // in the following, we need to recall that whenever we use the
      // element a_ii, we know that it is 2.01; similarly, the
      // *immediate* off-diagonal entries are -1, and all other matrix
      // entries are zero. this makes multiplying with the matrix
      // relatively straightforward, except that we have to pay
      // attention to the first and last row
      x[0]   = 1./2.01 * (b[0] - (-1 * x[1]));
      for (unsigned int i=1; i<N-1; ++i)
        x[i] = 1./2.01 * (b[i] - (-1 * x[i-1]) - (-1 * x[i+1]));
      x[N-1] = 1./2.01 * (b[N-1] - (-1 * x[N-2]));
    }
}
```

We can again produce the same graph as for the previous problem:



Looking at the solution after 200 iteration, we now see something that looks a lot more (in fact indistinguishable) like the solution of the Jacobi iteration after 2000 iterations:



This does not immediately quantify *how fast* the two methods converge, but it is clear that the Gauss-Seidel method has achieved a better solution after 200 iterations than the Jacobi method has.