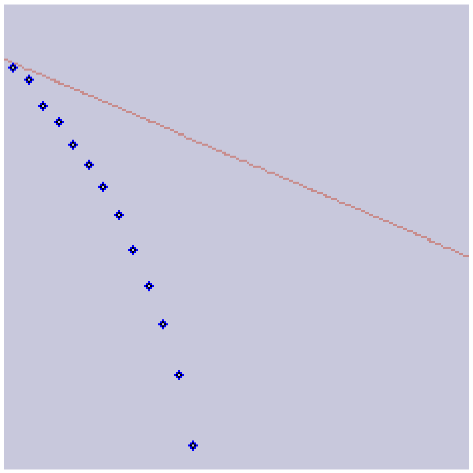
Jacobi Algorithm

The Jacobi algorithm finds the eigenvalues of a matrix A by producing increasingly diagonal similar matrices B_n until the off-diagonal entries are sufficiently small.

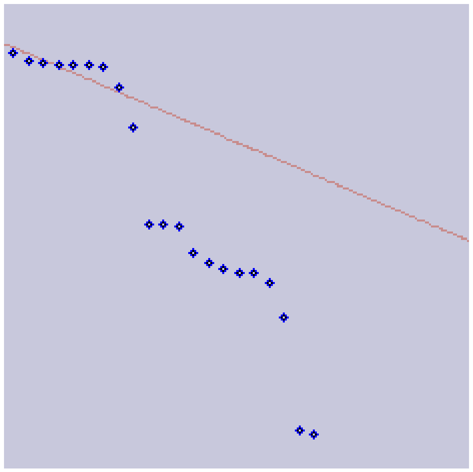
The progress of the algorithm at each step can be measured by $\text{Off}(B_n)$. The graphs produced by this application use the iteration count on the horizontal axis, and $\ln(\text{Off}(b_n))$ on the vertical. The red line shows the algorithm's theoretical maximum bound, given by $y = x \ln(9/10) + \ln(\text{Off}(A))$.

In optimizing the runtime, one consideration is how to select the off-diagonal entry that is to be reduced at each step. If the largest entry is always chosen, the algorithm will terminate in a minimum number of iterations. However, finding the largest entry can be time-consuming. The alternative is to simply run through each entry, regardless of its value.


Sorted: When the largest off-diagonal entry is always chosen, Off(A) decreases steadily.

	Sorted		*Unsorted*	
	Steps	*Time (ms)*	*Steps*	*Time (ms)*
Mean	24.64	104.48	33.84	104.56
St. dev.	1.68	5.64	2.66	5.75

Statistics generated from 25 trials on 5x5 matrices with randomly-generated entries between -10 and 10


Unsorted: Indiscriminantly sweeping through the off-diagonal entries yields less predicable results.

The latter unsorted method requires more steps to complete, but each step takes less time. The result is that both methods yield approximately equal running times, with the sorted method being slightly more consistent. Without sorting, the algorithm is not constrained to the same theoretical bound, and potentially has a much slower worst-case.

The sorted approach wastes processing time by finding the absolutely greatest entry, while the unsorted version is wasteful in running iterations on entries that are already small. The best solution would be to implement some heuristic to find large off-diagonal entries.