Code submission notes: Submit code for C1 and C2 to Sakai, following the usual instructions. You must include the `gs`, `mult_by_A` and `cg` functions, along with a main script with a working example (not necessarily the one used to answer the questions).

Note: If pressed for time before midterms, C2 is the problem to omit; it is a practical exercise and not part of the core material on the exam.

Book problems

- Chapter 7: 0 part (d), 3, 4ab (skip the SOR part)

Other problems

P1. (Prob. 7.9 in the book, adjusted). Consider the iteration

\[ x_{k+1} = x_k + \alpha (b - Ax_k) \]

where \( \alpha > 0 \) is a fixed value and \( A \) is symmetric positive definite with \( 0 < \lambda_1 < \cdots < \lambda_n \).

a) Write the iteration in the form

\[ x_{k+1} = Tx_k + c \]

and identify the iteration matrix \( T \).

b) Derive a condition on \( \alpha \) that guarantees the iteration converges for any initial guess \( x_0 \). You can, of course, use the theorems from class here.

c) Show that the optimal choice of \( \alpha \) is \( 2/((\lambda_1 + \lambda_n)) \) and that the optimal convergence rate is

\[ \begin{align*}
\kappa_2(A) - 1 \\
\kappa_2(A) + 1
\end{align*} \]

where \( \kappa_2(A) = ||A||_2||A^{-1}||_2 \) is the condition number in the 2-norm.

\[ ^1 \text{This is an example of gradient descent, a minimization algorithm, applied to } f(x) = \frac{1}{2} x^T A x - b \cdot x. \]

The method seeks a minimum of \( f(x) \) by following the negative gradient \( -\nabla f \), the direction of steepest descent down the function surface.
P2 (miscellaneous norm-related questions).

a) Let $x \in \mathbb{R}^n$. Show that $\|x\|_2^2 = x^T x$. Note: the notation is unfortunate here; the subscript indicates the 2-norm, and the superscript is an exponent.

b) Fix some integer $n \geq 2$. Suppose $x \in \mathbb{R}^n$ and $p \in [1, \infty)$. Show that for all $x \in \mathbb{R}^n$, $\|x\|_\infty \leq \|x\|_p \leq n^{1/p}\|x\|_\infty$.

c) Find $\kappa_1$, $\kappa_\infty$ and $\kappa_2$ for the matrix

$$A = \begin{bmatrix} 1 & 2 \\ -3 & 4 \end{bmatrix}.$$ 

d) For the matrix in (c), find a vector $x$ with $\|x\|_\infty = 1$ such that $\|Ax\|_\infty = \|A\|_\infty$ (i.e. the maximum in the definition of the norm is achieved).

P3 (estimating rate of convergence without the solution). Show that if a sequence of scalars $x_k$ converges to $x^*$ and

$$x_k \sim x^* + cr^k$$

for some $r \in (0, 1)$ and constant $c$ then

$$\lim_{k \to \infty} \frac{x_{k+1} - x_k}{x_k - x_{k-1}} = r.$$ 

Correction: removed the second part.
Computational problems

C1. Consider the tridiagonal $n \times n$ matrix $A$ with

$$a_{ii} = 2 \text{ for all } i, \quad a_{ij} = -1 \text{ for } |i - j| = 1$$

and $a_{ij} = 0$ otherwise. Written out:

$$A = \begin{bmatrix}
2 & -1 & 0 & \ldots & 0 \\
-1 & 2 & -1 & \ddots & \vdots \\
0 & \ddots & \ddots & \ddots & 0 \\
\vdots & \ddots & -1 & 2 & -1 \\
0 & \cdots & 0 & -1 & 2
\end{bmatrix}.$$  

The goal of this problem (and the next) is to write an algorithm that can handle a large $n$, where the full matrix is too large to be constructed, and study convergence numerically.

a) Write an algorithm $\text{gs}(x_0, b, n, \text{tol})$ that solves $Ax = b$ using the Gauss-Seidel method without ever constructing the matrix $A$. Use

$$||x^{k+1} - x^k|| < \text{tol}||x^k||$$

as a stopping criterion (your choice of norm).

b) Let $\rho_n$ be the spectral radius for the iteration matrix. Using the right hand side\(^2\)

$$b_i = \frac{1}{(n+1)^2} f\left(\frac{i}{n+1}\right), \quad i = 1, 2, \ldots, n.$$  

and $f(x) = x$, estimate $\rho_n$ from the output of the algorithm. Hint: use P3; just look at one component of the vector. (Updated hint)

Create a plot of $\rho_n$ vs. $n$ and make a conjecture about how $\rho_n$ depends on $n$. Provide evidence using your output and/or plots. *Note: more precise is better, but you don’t have to get a perfect answer here.*

c) What is $\rho_n$ when $n = 100$? When $n = 1000$? Is the method suitable for large $n$?

Coding notes:

- When $n$ is large, you may need to stop before it gets close to the solution (set a max. number of iterations!); you can still estimate the rate from this output.

- As per usual, you may need to output a vector of data to answer (a), even though this may not be efficient to store. Hint: avoid saving the whole vector for all iterations!

- Your algorithm will be specific to the matrix $A$ given. The generalization would be to take in a general **sparse matrix** (see Section 5.6 of the textbook) but you do not have to worry about that here.

- The calculations give a numerical solution to the boundary value problem for $u(t)$,

$$-u_{tt} = f(t), \quad u(0) = u(1) = 0.$$  

\(^2\)The right hand side doesn’t really matter here; this particular $b$ is chosen because it should ensure that the $x_i$’s are not too large in magnitude.
C2 (Conjugate gradient).

**Context:** In this problem, you will use a better method (used often for real problems!) for the problem in C1. Think of the method as a 'black box' of sorts that produces a sequence of iterates converging to the answer. One of the points of the exercise is to implement and study the algorithm numerically without having the theory, which tends to happen when you need an algorithm and copy it from somewhere or use existing software.

a) Write a function \( y = \text{mult\_by\_A}(x) \) that takes an input vector \( x \) and outputs \( Ax \) where

\[
A = \begin{bmatrix}
2 & -1 & 0 & \ldots & 0 \\
-1 & 2 & -1 & \ddots & \\
0 & \ddots & \ddots & \ddots & 0 \\
\vdots & \ddots & -1 & 2 & -1 \\
0 & \cdots & 0 & -1 & 2
\end{bmatrix}
\]

That is, it takes in a function like the one in (a) (for whatever matrix). Note that this gets around explicitly forming \( A \); the algorithm only needs to know how to multiply \( A \) by a vector.

b) Implement the **conjugate gradient** (CG) algorithm in Section 7.4 (page 184), as

\[
x, \cdots = \text{cg}(\text{mult\_by\_A}, x_0, b, \text{tol})
\]

so that if the user needs to solve \( Ax = b \), a 'multiply-by-\( A \)' function must be written and supplied, but \( A \) itself does not need to be provided.

The CG method is a good choice for solving \( Ax = b \) where \( A \) is a symmetric positive definite matrix (it must be SPD, which is the main limitation). You can, of course, ignore all of the theory; just use the pseudocode. See code notes for details.

c) Test your code on \( Ax = b \) (for any \( b \), e.g. the one from C2) with \( A \) as in part (a) for various values of \( n \). Verify that after \( n \) iterations, the method terminates even when \( \text{tol} \) is quite small (e.g. \( 10^{-12} \)).

d) Plot the error \( e_k = ||x_k - x^*|| \) vs \( k \). If \( n \) is large (e.g. \( n = 1000 \)), does the method perform better than the one in C2? Discuss any interesting features that you see. **Hint:** To get the 'exact' solution \( x^* \), run the algorithm with a small tolerance and take that as the exact solution. In practice, one would have a solution from another source.

**Coding notes:**

- If using Matlab, remember that to pass a function argument you must use \texttt{@func\_name} rather than \texttt{func\_name}, which tells Matlab you mean the function variable and not 'evaluate this function'.
- Write the code **without** storing the variables for each \( k \) (e.g. use \( r \) instead of \( r(k) \) ).
  You will need a 'previous' value for at least one of them.
- The \( \langle x, y \rangle \) notation means \( x \cdot y \) (dot product). The vectors \( x_k \) and \( r_k \) are the iterates (converging to the solution) and the approximate residual.
- To debug, you can use Example 7.9, which lists detailed output.