ALOP Autumn School
High performance solution of linear optimization problems

Numerical linear algebra

1. Load the matrix A from the file `bcsstk08.mat`: this defines an array `A`

   (a) Identify the dimension n of the matrix A

   (b) Generate `xstar` to represent a vector $\boldsymbol{x}^* \in \mathbb{R}^n$, all of whose entries are 1

   (c) Compute `b=A*xstar` to represent the vector $\boldsymbol{b} = A\boldsymbol{x}^*$

   (d) Find and report the time required to solve $A\boldsymbol{x} = \boldsymbol{b}$

      i. By computing A^{-1} as `Ainv` and then forming `Ainv*b`

      ii. Using `A\b`

      You should see that it is much faster to use "backslash" than form the inverse and multiply by it.

   (e) Compute and report the residual norm $\|A\boldsymbol{x} - \boldsymbol{b}\|_2$ and norm of the solution error $\|\boldsymbol{x} - \boldsymbol{x}^*\|_2$. Is one approach more accurate than the other?

   (f) Repeat using the files `nasa1824.mat` and `pds02.mat`. Do your observations change?

2. Download `Arrowmatrix.m` which, for a given value of n, returns $A = \begin{bmatrix} -1 & -1 & -1 & \dots & -1 & -1 \\ -1 & 1 & & & & \\ -1 & & 1 & & & \\ \vdots & & & \ddots & & \\ -1 & & & & 1 & \\ -1 & & & & & 1 \end{bmatrix}$

   (a) Convince yourself that forming the decomposition $PA = LU$ using Gaussian elimination with partial pivoting results in L and U having no zero entries below and above the diagonal respectively. Convince yourself that forming the decomposition $PAQ = LU$ by pivoting using the Martkowitz criterion leads to L and U having no more nonzero entries than the corresponding triangles of A.

   (b) Write a script (using the dimension `n=10` to develop your script) which performs the following

      i. Determine the number of nonzeros in A [using `nnz(A)`]

      ii. Determine the elapsed time to compute A^{-1} [using `inv(A)`]

      iii. Determine the total number of nonzeros in A^{-1}

      iv. Determine the elapsed time to form the decomposition $PA = LU$ [using `[L U P]=lu(A)`]

      v. Determine the total number of nonzeros in both L and U [ie `nnz(L)+nnz(U)-n`, see below for explanation of the subtraction of n]

      vi. Report the dimension of A, its number of nonzeros, the times for the matrix computations in (ii) and (iv), and the numbers of nonzeros in their results

   (c) The command `sparse(A)` returns the matrix A in a format which exploits its sparsity. Add commands to your script to perform the following

      i. Form the sparse version of A, referred to below as `SparseA`

      ii. Determine the elapsed time to form the (sparsity-exploiting) decomposition $PAQ = LU$ [using `[L U P Q]=lu(SparseA)`]

      iii. Determine the total number of nonzeros in both L and U

      Run your script for $n = 50 \times 2^k$ for $k = 1, \dots, 7$. You should observe that it is much more efficient (in terms of time and sparsity) to form $PAQ = LU$ than $PA = LU$ or A^{-1}.

(d) After a computation with a sparse matrix, the ratio of the number of nonzeros in the result to the number of nonzeros in the matrix is known as the *fill factor*. The best that can reasonably expected is a fill factor of 1. The density of the result of a matrix computation is the number of nonzeros in the result divided by the product of the dimensions of the matrix, and is usually expressed as a percentage.

Add commands to your script to compute and report the fill factor and density for the computed LU decompositions and inverse

Note

- Since the n diagonal entries of L are known structurally, n is subtracted from the total number of nonzeros in L and U to get the number of true nonzeros. Otherwise the (trivial) LU decomposition of the identity matrix would have a fill factor of 2.
- To force `sprintf` to display a "`%`" symbol, you need to put "`%%`" into the string. This is because a single "`%`" symbol is interpreted as a format statement.

(e) Download `bcsstk08.mat`, `nasa2910.mat` and `c-36.mat`, and run your script for the matrix A which is loaded by the command `load('file.mat');` for each of the files `bcsstk08.mat`, `nasa2910.mat` and `c-36.mat`, observing the extreme density and fill-factor when forming A^{-1} and the relative efficiency of exploiting sparsity when forming the LU decomposition.

(f) Run your script for the file `pds02.mat`. Do you observe any odd behaviour?

3. Download `MyCG.m` which performs (at most n) iterations of the conjugate gradient method, starting from $\boldsymbol{x}^{(1)} = \mathbf{0}$, until $\|\boldsymbol{r}^{(k+1)}\|_2 \leq 10^{-8}$. As well as $\boldsymbol{x}$, the function returns the index m of the final iteration.

Conjugate gradient algorithm:

For $A\boldsymbol{x} = \boldsymbol{b}$, given some estimate $\boldsymbol{x}^{(1)}$ of $\boldsymbol{x}$, let $\boldsymbol{s}^{(1)} = \boldsymbol{r}^{(1)} = \boldsymbol{b} - A\boldsymbol{x}^{(1)}$. Then, for some tolerance ϵ, repeat for $k = 1, 2, \ldots$

$$\boldsymbol{w} = A\boldsymbol{s}^{(k)}$$

$$\alpha^{(k)} = \frac{\boldsymbol{r}^{(k)^T}\boldsymbol{s}^{(k)}}{\boldsymbol{w}^T\boldsymbol{s}^{(k)}}$$

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha^{(k)}\boldsymbol{s}^{(k)}$$

$$\boldsymbol{r}^{(k+1)} = \boldsymbol{r}^{(k)} - \alpha^{(k)}\boldsymbol{w}$$

If $\|\boldsymbol{r}^{(k+1)}\|_2 \leq \epsilon$ then stop

$$\beta^{(k)} = \frac{\|\boldsymbol{r}^{(k+1)}\|_2^2}{\|\boldsymbol{r}^{(k)}\|_2^2}$$

$$\boldsymbol{s}^{(k+1)} = \boldsymbol{r}^{(k+1)} + \beta^{(k)}\boldsymbol{s}^{(k)}$$

(a) The performance of the conjugate gradient method depends on the eigenvalues of the matrix A. Download `AnalyseEigs.m` and extend your script by calling `AnalyseEigs(A)` before performing CG. `AnalyseEigs(A)` counts and reports the number of complex and non-positive eigenvalues of A, as well as plotting the eigenvalues on a log scale.

Test the performance of CG using the following matrices, choosing the solution of $A\boldsymbol{x} = \boldsymbol{b}$ to be a random vector and forming $\boldsymbol{b} = A\boldsymbol{x}$.

i. $A = \lambda I$ of dimension $n = 1000$. Choose any nonzero value of λ that you want: CG should always terminate after one iteration

ii. The matrix A from `662_bus.mat`. [Remember to extract `n=size(A,1)`.]

iii. The matrix A from `bcsstk08.mat`. [Remember to extract `n=size(A,1)`.]

iv. The matrix $A = I + \epsilon E$ of dimension $n = 1000$, where E is the random sparse symmetric matrix with $\tau = 10n$ entries generated by the MATLAB function `SpSymmRand(n, tau)` which you will have to download. Use the sparse identity matrix given by `speye(n)`. Try $\epsilon = 10^{-k}$ for $k = 1, 2, 4, 8$.

v. $A = I + \boldsymbol{u}\boldsymbol{u}^T$ of dimension $n = 1000$ for a random vector $\boldsymbol{u}$ such that $\|\boldsymbol{u}\|_2 = 1$. This matrix has two distinct eigenvalues: $\lambda = 2$ (of multiplicity 1) and $\lambda = 1$ (of multiplicity $n - 1$).

(b) Compute and report the elapsed time of the CG iterations, together with the error measures $\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|_2$ and $\|A\hat{\boldsymbol{x}} - \boldsymbol{b}\|_2$, where $\hat{\boldsymbol{x}}$ is the solution computed by CG. Compare these with the same performance measures for `A\b`.

(c) **Extension:** A preconditioner for A is the **incomplete Cholesky decomposition** $A = \widetilde{L}\widetilde{L}^T$, where $\widetilde{L}$ is computed with values below a "drop tolerance" in magnitude set to zero. In MATLAB this is computed as

```
L = ichol(A,struct('type','ict','droptol',droptol));
```

i. Develop a function `MyPCG.m` which uses $\widetilde{L}$ as a preconditoner. Test this with a small problem such as `662_bus.mat`. As the drop tolerance reduces, so should the number of CG itertions required.

ii. For the larger problem `bodyy4.mat`, can you find a value for the drop tolerance such that the time required to compute $\widetilde{L}$ and use it in `MyPCG.m` is less than the time required by MATLAB's backslash operator?

Machine learning and data science applications

1. In the **support vector machine** problem, we are given a training dataset of n points of the form

$$(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n),$$

where $y_i \in \{-1, 1\}$ indicates the class to which the point $\boldsymbol{x}_i \in \mathbb{R}^p$ belongs. Let the rows of $X \in \mathbb{R}^{n \times p}$ be the vectors $\boldsymbol{x}_i$ and let $\boldsymbol{y} \in \{-1, 1\}^n$ be the vector of class indicators y_i. A linear support vector machine classifies a new point $\boldsymbol{x} \in \mathbb{R}^p$ by

$$\boldsymbol{x} \mapsto \operatorname{sign}(\boldsymbol{w}^T\boldsymbol{x} + w_0).$$

The normal vector $\boldsymbol{w} \in \mathbb{R}^p$ and the offset $w_0/\|\boldsymbol{w}\|$ define a hyperplane in $\mathbb{R}^p$ and need to be determined from the training set. To this end, we need to solve

$$\underset{\boldsymbol{w}, w_0}{\text{minimize}} \quad \frac{C}{n}\left[\sum_{i=1}^{n} \max\bigl(0, 1 - y_i(\boldsymbol{w}^T\boldsymbol{x}_i - w_0)\bigr)\right] + \|\boldsymbol{w}\|_2^2,$$

where $C > 0$ is a penalty parameter.

(a) By introducing variables

$$s_i = \max\bigl(0, 1 - y_i(\boldsymbol{w}^T\boldsymbol{x}_i - w_0)\bigr) \quad i = 1, \ldots, n$$

show that the problem may be formulated as the smooth, constrained QP problem

$$\underset{\boldsymbol{s}, \boldsymbol{w}, w_0}{\text{minimize}} \quad \boldsymbol{w}^T I \boldsymbol{w} + \frac{C}{n}\boldsymbol{e}^T\boldsymbol{s} \quad \text{subject to} \quad -YX\boldsymbol{w} + \boldsymbol{y}w_0 - \boldsymbol{s} \leq -\boldsymbol{e}, \quad \boldsymbol{s} \geq \boldsymbol{0},$$

where $\boldsymbol{e}$ is a vector of ones, Y is a diagonal matrix and $\boldsymbol{y}$ is a vector, both with entries y_i for $i = 1, \ldots, n$, and s_i is the i^{th} component of $\boldsymbol{s}$.

(b) Load `svm_training.mat` into you MATLAB session, which contains a set of 1000 training points in $\mathbb{R}^2$ and their class indicators. Visualize the points using the provided `svmplot.m` function.

(c) Solve the quadratic program for this data set and penalty parameter $C = 100$. The MATLAB function `quadprog` can be used to

$$\text{minimize} \quad \frac{1}{2}\boldsymbol{x}^T H\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} \quad \text{subject to} \quad A\boldsymbol{x} \leq \boldsymbol{b}$$

[**Note:** To pass a vector of lower bounds on the variables, the function is called with `x=quadprog(H,c,A,b,[],[],lb)`.]

(d) Add the hyperplane [line since $p = 2$] to the plot of points created by `svmplot.m`

2. The ball $B = \{\boldsymbol{x} \mid \|\boldsymbol{y} - \boldsymbol{x}\|_2 \leq r\}$ of centre $\boldsymbol{y} \in \mathbb{R}^n$ and radius r is the set of all points in $\mathbb{R}^n$ within a Euclidean distance r of $\boldsymbol{y}$. The **Chebychev centre problem** is that of finding the centre and radius of the largest ball within the polyhedron $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$, where $A \in \mathbb{R}^{m\times n}$.

(a) i. Show that points a distance r from $\boldsymbol{y} \in \mathbb{R}^n$ in the direction $\boldsymbol{a}_i$ are given by

$$\boldsymbol{x} = \boldsymbol{y} + \frac{r}{\|\boldsymbol{a}_i\|_2}\boldsymbol{a}_i$$

ii. Show that these points lie within the set $\boldsymbol{a}_i^T\boldsymbol{x} \leq b_i$ if

$$b_i \geq \boldsymbol{a}_i^T\boldsymbol{y} + \|\boldsymbol{a}_i\|_2 r$$

iii. Hence show that the Chebychev centre problem may be formulated as the LP problem

$$\underset{\boldsymbol{y},r}{\text{minimize}} \ -r \quad \text{subject to} \quad \boldsymbol{a}_i^T\boldsymbol{y} + \|\boldsymbol{a}_i\|_2 r \leq b_i, \ i = 1, \ldots, m$$

(b) Find the Chebychev centre and maximum radius when

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 0 & 1 \\ 0 & -1 & -1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 15 \\ 12 \\ -5 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

[**Note:** Calling `x=linprog(c,A,b)` in MATLAB finds the solution of the LP problem

$$\text{minimize } \boldsymbol{c}^T\boldsymbol{x} \quad \text{subject to} \quad A\boldsymbol{x} \leq \boldsymbol{b},$$

where arrays `A`, `b`, `c` and `x` correspond to a matrix A and vectors $\boldsymbol{b}$, $\boldsymbol{c}$ and $\boldsymbol{x}$.]

(c) Which of the hyperplanes does the maximum ball touch?
[Note that you should use a tolerance on values being zero (e. g. 10^{-6}) since the LP solution in (b) is computed using inexact arithmetic.

(d) How does a small change in the last component of $\boldsymbol{b}$ affect the radius?

[Adapted from D. Michaels, K. Ballerstein ETH Zürich, spring term 2011]

Interior point methods

1. **IPM Exercise:**

This is an opportuity to use a simple interior point solver written in MATLAB to solve small LP problems

(a) Download `ipexample.m`, `ipdriver.m`, `newtonsolve.m` and `newtonsolver.m`

- The function `ipdriver.m` solves the LP problem

$$\text{minimize } f = \boldsymbol{c}^T\boldsymbol{x} \quad \text{s. t.} \quad A\boldsymbol{x} = \boldsymbol{b} \quad \text{and} \quad \boldsymbol{x} \geq \boldsymbol{0}$$

- The script `ipexample.m` sets up `A`, `b` and `c` and calls `ipdriver.m` to solve

$$\begin{array}{lrcrcrcl} \min & f & = & 2x_1 & + & x_2 & & \\ \text{s. t.} & & & x_1 & + & x_2 & \geq & 3 \\ & & & 3x_1 & + & x_2 & \geq & 6 \\ & & & x_1 \geq 0 & & x_2 \geq 0 & & \end{array}$$

(b) Modify `ipexample.m` so that it solves

$$\begin{array}{lrcrcrcrcl} \max & f & = & x_1 & + & 2x_2 & + & 3x_3 & & \\ \text{s. t.} & & & x_1 & & & + & 2x_3 & \leq & 3 \\ & & & & & x_2 & + & 2x_3 & \leq & 2 \\ & & & x_1 \geq 0 & & x_2 \geq 0 & & x_3 \geq 0 & & \end{array}$$

(c) To use `ipdriver.m` to solve larger LP problems, the values of `A`, `b` and `c` can be read from `.mat` files. Download `AFIRO.mat` and `ADLITTLE.mat` then use `ipdriver.m` to solve them.

(d) **Extension:** Study `ipdriver.m`, `newtonsolve.m` and `newtonsolver.m` to see how the IPM is implemented in MATLAB

(e) **Extension:** Modify `ipdriver.m` and `newtonsolver.m` so that the IPM implementation solves QP problems.