ELEC 3035: Lab 1 — matrix computations in Matlab

Lecturer: Ivan Markovsky

1. *Computation time of a matrix–matrix product* For $n = 1$, 2, 3, and 4 do the following.

   (a) Using `randint`, generate two $n \times n$ random matrices, call them A and B, with integer elements between 0 and 9. Sample Matlab code for doing this is:

   ```
   for n = 1:4
      a{n} = randint(n,n,[0,9]);
      b{n} = randint(n,n,[0,9]);
      fprintf('n = %d:\n', n), A = a{n}, B = b{n},
   end
   ```

   (b) Find the product $C = AB$ by hand. Measure and record the time it takes you to do this computation.

   (c) Find the product $C = AB$ using Matlab and compare with your results. Use the commands `tic` and `toc` to measure the execution time for computing AB. Sample Matlab code for doing this is:

   ```
   for n = 1:4
      tic, c{n} = a{n} * b{n}; t(n) = toc;
      fprintf('n = %d: computation time %f sec\n', n, t(n))
   end
   ```

   (d) In order to have a better idea how long matrix product takes, repeat the experiment for n in the range from 500 to 2000 and plot the results. Sample Matlab code for doing this is:

   ```
   N = 10;
   n = round(linspace(500,2000,N));
   for i = 1:N
      ni = n(i);
      a = randint(ni,ni,[0,9]);
      b = randint(ni,ni,[0,9]);
      tic, c = a * b; t(i) = toc;
      fprintf('n = %4d: computation time %f sec\n', ni, t(i))
   end
   plot(n,t,'linewidth',2)
   ```

   Solution:

   (b) Not surprisingly, the comparison between the human and computer computation time shows that computers are much better than humans in doing computations. The conclusion is "let them do computations for us". We spend, however, a lot of time doing as exercises routine computations. As far as ELEC3035 Part 1 is concerned, this is the first and last time when you will be asked to do hand computations with matrices of size more than 3×3. Instead, you will be taught and asked to interpret and make sense of results.

   (c) The execution times reported by Matlab may differ on different computers. Moreover, the execution times may differ on the same computer, when the script is run at different moments of time. (Check this!) On my computer, in a particular run of the script, I obtained:

   ```
   n = 1: computation time 0.000010 sec
   n = 2: computation time 0.000161 sec
   n = 3: computation time 0.000028 sec
   n = 4: computation time 0.000014 sec
   ```

   It looks strange that for $n \geq 2$ the computation time drops as a function of n. (Of course, one expects it to go up.) The reason for this anomaly is that for such small matrices the execution time is actually meaningless; it is completely dominated by Matlab's overhead of calling FORTRAN libraries called BLAS (Basic Linear Algebra Subroutines) and LAPACK (Linear Algebra PACKage) that do the computation. (Note that

Matlab itself does not do the computations; it simply parses your input and calls the appropriate function from software libraries.) In addition, the computation time may vary depending on other processes running on the computer. There seems to be, however, a nonrandom feature of these computation times—for $n = 2$ the computation time is much bigger. This is due to the fact that Matlab needs more time to call a function for the first time. In case of functions written in Matlab code, this is due to compilation. In case of external functions, such as the BLAS and LAPACK functions, I am not sure about the precise reasons.

(d) The plot of the computation time vs matrix size is shown on Figure 1.

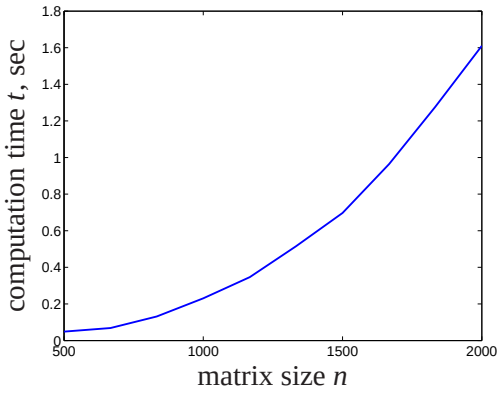

Figure 1: Computation time for square matrix-matrix product as a function of the matrix size.

□

2. *Solution of linear system of equations* Consider a linear system of equations $Ax = b$, where A is an $m \times n$ matrix and b is an $m \times 1$ vector, and let `a` and `b` be variables that represent, respectively, A and b in the computer's memory. Matlab has the handy notation `a\b`, for solving the system $Ax = b$. Solve the following systems of equations i) by hand and ii) using Matlab's `a\b`

(a) $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} x = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$

(b) $\begin{bmatrix} 1 & 2 \end{bmatrix} x = 5$

(c) $\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} x = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$

(d) $\begin{bmatrix} 1 \\ 3 \end{bmatrix} x = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$

(e) $\begin{bmatrix} 1 & 1 \\ 3 & 3 \end{bmatrix} x = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$

Comment on the the results.

Solution:

(a) The unique solution is $x_1 = -4$, $x_2 = 4.5$, because is full rank A. In this case, `a\b` returns *the* solution.

(b) A solution is nonunique. The general solution is $x_1 = 5 - 2\alpha$, $x_2 = \alpha$, where $\alpha \in \mathbb{R}$ is a free parameter. `a\b` returns *a* particular solution $x_1 = 0$, $x_2 = 2.5$. Note that A is full row rank,

(c) A solution is nonunique. The general solution is $x_1 = 5 - 2\alpha$, $x_2 = \alpha$, where $\alpha \in \mathbb{R}$ is a free parameter. In this case, A is not full row rank and `a\b` issues a warning message "Matrix is singular to working precision." and returns a vector of `NaN`s — not a number.

(d) There is no solution. `a\b` returns a least squares approximate solution ($x = 2.3$ in the example). Note that A is full column rank.

(e) There is no solution. The matrix A is not full column rank and `a\b` issues a warning message "Matrix is singular to working precision." and returns a vector of `Inf`s — infinity.

The examples show that Matlab's backslash operator is not sufficient to characterize the solution in general. In cases when the solution is not unique, it will issue a particular solution and in cases when the solution does not exist it will issue an approximate solution without warning you. In addition, when the matrix A is not full rank,

the backslash operator may fail to find a solution even though a solution exists. The finite precision arithmetic that computers use makes the borderline between "unique" and "nonunique" and "exists" and "does not exist" fuzzy. This makes numerical computations nontrivial, especially for large data matrices. □

3. *Polynomial approximation* In this exercise, you will *predict* the computation time for matrix-matrix multiplication. Consider a sequence of problem size n_i, $i = 1,\ldots,N$ and a corresponding sequence of computation times t_i, $i = 1,\ldots,N$. There is a causal relation between t and n, *i.e.*, there is a function f, such that $t = f(n)$. (We discussed that for small values of n the corresponding computation time may vary due to other processes running on the computer, which makes the relation between n and t nondeterministic. For large values of n, however, these small variations are insignificant.) Since the function f is unknown to us, we will discover it from the available data. More precisely, we will approximate f by a polynomial function of degree at most r

$$\widehat{f}(n) = x_0 + x_1 n + \cdots + x_r n^r. \tag{1}$$

Here $x_0, x_1, \ldots, x_r$ are coefficients that parameterize $\widehat{f}$. We will find these coefficients by solving the system of equations

$$\widehat{f}(n_i) = t_i, \qquad \text{for } i = 1,\ldots,N, \tag{2}$$

which postulates that $\widehat{f}$ should do correct predictions for the instances for which we have data. Using the fact that $\widehat{f}$ is a polynomial (1), the system of equations (2) can be written in a standard form of a linear system of equations

$$\underbrace{\begin{bmatrix} 1 & n_1^1 & \cdots & n_1^r \\ 1 & n_2^1 & \cdots & n_2^r \\ \vdots & \vdots & & \vdots \\ 1 & n_N^1 & \cdots & n_N^r \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_r \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}}_{b}. \tag{3}$$

Matrices with structure of the type A has are called Vandermonde matrices.

(a) Using the data (n_i, t_i) from Exercise 1d construct the matrix A, for $r = 3$, and the vector b.

(b) Solve the systems of equations (3) for $r = 0$, 1, 2, and 3, using Matlab's backslash operator. Is the solution exact or approximate, unique or nonunique?

(c) Plot the approximations $\widehat{f}$ of degrees 0, 1, 2, and 3 and the data. Give an interpretation of the 0th order polynomial approximation.

(d) Plot the approximation error $e_r := \|Ax - b\|$ as a function of r

(e) Predict the computation time, using the 3rd order approximation, for the following problem sizes $n = 2500$, 3000, 4000.

(f) Verify experimentally the predicted computation times.

Solution:

(c) Matlab code for parts (a)–(d) is:

```
n = n(:); % make n a column vector
b = t(:); % define b
plot(n,t,'k-','linewidth',2), hold on % plot the data

% 0th order approximation
a0   = ones(N,1);
x0   = a0\b;
th0  = a0*x0;         % approximation
e(1) = norm(b - th0); % approximation error
plot(n,th0,'r:','linewidth',2)

% 1st order approximation
```

```
a1  = [a0 n];
x1  = a1\b;
th1 = a1*x1; e(2) = norm(b - th1);
plot(n,th1,'r-.','linewidth',2)

% 2nd order approximation
a2  = [a1 n.*n];
x2  = a2\b;
th2 = a2*x2; e(3) = norm(b - th2);
plot(n,th2,'r--','linewidth',2)

% 3rd order approximation
a3  = [a2 a2(:,end).*n];
x3  = a3\b;
th3 = a3*x3; e(4) = norm(b - th3);
plot(n,th3,'b--','linewidth',2)

% Plot the approximation error as a function of r
figure, plot(0:3,e,'-',0:3,e,'o','linewidth',2)
```

The 0th order polynomial is a constant equal to the mean of t.

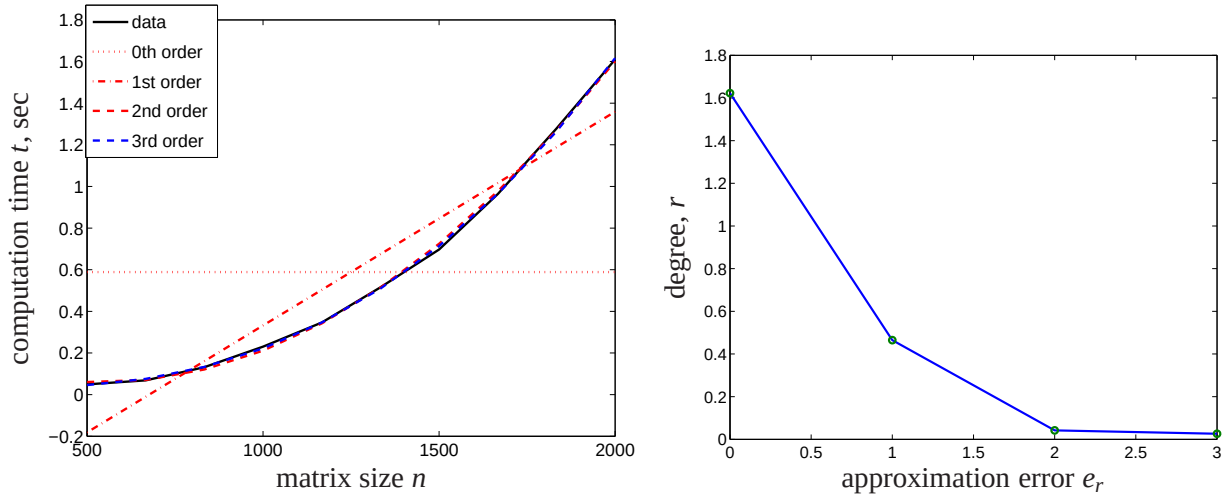

Figure 2: Left: Computation time data and 0th, 1st, 2nd, and 3rd degree polynomial approximations, Right: Approximation error e_r as a function of the as a function of the polynomial degree r.

(b) Matlab code for parts (e) and (f) is:

```
for n = [2000 2500 3000]
   % predicted
   tp = [1 n n*n n*n*n] * x3;
   % experimental
   a = randint(n,n,[0,9]);
   b = randint(n,n,[0,9]);
   tic, c = a * b; te = toc;
   % print the results
   fprintf('n = %4d: predicted %fsec, experiemntal %fsec\n',n,te,tp)
end
```

and the obtained result is

```
n = 2000: predicted 1.627396sec, experiemntal 1.616024sec
n = 2500: predicted 3.131055sec, experiemntal 3.024313sec
n = 3000: predicted 5.372153sec, experiemntal 5.034020sec
```

□