Mat. Comp., Practical Exercises, Due 1 Esfand 1404

Please do not share this document. Using any functions described in the Maxima documentation is permitted. In particular, you may freely use Maxima's linear-algebra and list-processing commands. But apply the algorithms presented in class. For exercises involving random output, aim for results that are approximately uniform and free of noticeable bias. Please check the cw website regularly for the latest updates to the exercise statements.

Some remarks:

1. To see how to define a function with required arguments and optional arguments, consider the following example:

```
g(x,[L]):=block([i],if L#[] then x^2+sum(L[i]^2,i,1,length(L))
     else x^2);

g(2);
                              4
g(2,3);
                             13
g(2,3,5);
                             38
```

2. For an editor with syntax highlighting, you may find the following options helpful:
   - VS Code with the *Maxima Syntax Highlighting* extension
   - Notepad++
   - TeXmacs
3. Maxima includes built-in debugging tools. In some cases, the error messages shown in the command-line version are also informative. The WxMaxima interface also provides its own debugging features.
4. On macOS, one of the simplest ways to install Maxima and wxMaxima is via Homebrew (assuming Homebrew is already installed):

```
brew install maxima
brew install wxmaxima
```

Exercise 1. (Random matrix generation and matrix types)

1. Write a function `randmat(m,n,range,[optional_arguments])` where $range = [r, s]$ with the following behavior:
   - **Input:**
     - Two positive integers `m,n`.
     - Two integers $r \leqslant s$.
   - **Output:**

- A random $m \times n$ matrix with entries chosen uniformly from the interval $[r, s]$[1]

- **Optional arguments:** (comma-separated keywords specifying the matrix type)
  `triangular_upper, triangular_lower, diagonal, antidiagonal, symmetric, skew_symmetric, positive`[2]`, nonnegative, companion, permutation, elementary,`[3]`unipotent_lower, unipotent_upper,` ~~hermitian~~ `,tridiagonal, bidiagonal_upper, bidiagonal_lower, scalar, diagonal_positive`[4]`,diagonally_dominant, hessenberg_upper, hessenberg_lower, jordan_upper, jordan_lower, integer, real`[5]`, rational`

  Remark: Exactly one of the following options must be included in the input:

  `integer, real, rational.`

  Among the remaining optional arguments, at most one may be specified, with the exception of using one of

  `positive,nonnegative,diagonal_positive,`

  which may be combined with any of the other options.

2. Write a function `mattype(M)` that takes a matrix M as input and returns a list of all the types above that M satisfies.

Exercise 2. (Householder transformations)

1. Write a function `hh_mat(n,w)` that takes a positive integer n and a nonzero column vector $w \in \mathbb{C}^n$, given either as a list $w = [a_1, \ldots, a_n]$ or as a column matrix, and returns the Householder matrix $H_w(x) = x - \frac{2\langle w,x\rangle}{\langle w,w\rangle} w$.
2. Write a function `hhp(A)` that takes a matrix $A \in \mathbb{R}^{n\times n}$ and returns a vector w such that $H_w = A$. If no such vector exists, the function should return `false`.
3. Write a function `randmat_unitary(n,range,[optional_args])` where n is a positive integer, *range* $= [r, s]$ is a list of integers with $r \leqslant s$, and `optional_args` is one of `real,complex_rational,complex_real, rational,integer`. The output should be the matrix $H_{w_1} H_{w_2} \cdots H_{w_k}$, where $1 \leqslant k \leqslant n$ is chosen at random and each w_i is a random column vector generated by $w_i = \mathrm{randmat}(n, 1, [r, s], \mathrm{optional\_args})$. In the cases `complex_real` and `complex_rational`, use the vectors

---

[1] Except when the option 'rational' is specified, in which case the random entries take the form $\frac{r_1}{s_1}$ with $s_1 \neq 0$, where r_1 and s_1 are integers satisfying $|r_1| \leqslant |r|$ and $|s_1| \leqslant |s|$.

[2] Positive (nonnegative) means random entries are strictly positive (nonnegative)

[3] By *elementary* we mean an elementary matrix of the first type, i.e., a matrix that differs from the identity only in a single random entry ij with $i \neq j$.

[4] Means the entries on the main diagonal are positive

[5] Real means floating-point random entries.

$$randmat(n,1,[r,s],real) + \%i\, randmat(n,1,[r,s],real)$$

and $randmat(n,1,[r,s],rational) + \%i\, randmat(n,1,[r,s],rational)$ respectively.

4. Write a function `randmat_hh(n,range,[optional_arguments])` where n is a positive integer, $range = [r,s]$ with integers $r \leqslant s$, and `optional_args` is one of `integer,rational,real,complex_real,complex_rational`. The function should return the Householder matrix determined by the column vector $w = \text{randmat}(n,1,[r,s],\textit{optional})$, where `optional` is the optional argument passed to the `randmat_hh` function.
5. Write a function `hh_vec(a,b)` that takes two column vectors `a,b` of the same norm, given either as lists of numbers or as column matrices, and returns a column vector w such that $H_w(a) = b$.
6. We recall that the output of the Maxima function `eigenvalues(M)` or `eivals(M)` is a weighted list of the form $[[\lambda_1,\ldots,\lambda_k],[n_1,\ldots,n_k]]$ where λ_i's are distinct eigenvalues of a square matrix M and n_i is the multiplicity of λ_i. Write a function `randmat_eivals(L)` that takes as input a list $L = [[\lambda_1,\ldots,\lambda_k],[n_1,\ldots,n_k]]$ and returns a random matrix M whose eigenvalues' list is L. Write it in such a way to accept an optional argument `hermitian` to output a hermitian matrix with the mentioned property. Hint. First construct a triangular matrix R whose eigenvalues' list is L, then randomize it again by ARA^{-1} where A is a random invertible matrix. For hermitian case, construct a diagonal matrix D with this eigenvalues' list then randomize it with UDU^* where U is a random unitary matrix.
7. Write a function `hh_qr(A)` that takes as input a matrix $A \in \mathbb{R}^{m\times n}$ and returns a list $L = [H_{w_1},\ldots,H_{w_k},R]$ where each H_{w_i} is a Householder matrix in $\mathbb{R}^{m\times m}$ and R is an upper triangular matrix in $\mathbb{R}^{m\times n}$ such that $A = QR$ with $Q = H_{w_1}H_{w_2}\cdots H_{w_k}$ and $k \leqslant n$.
8. Write a function `hh_schur(M,L)` where M is a matrix with eigenvalues' list L and returns a list $[H_{w_1},H_{w_2},\ldots,H_{w_k},R]$ such that R is an upper triangular matrix and the H_{w_i} are Householder matrices satisfying
$$H_{w_k}^*\cdots H_{w_1}^* M H_{w_1}\cdots H_{w_k} = R$$
with $k \leqslant n-1$.
9. Write a function `hh_hessenberg(M)` that takes as input a square matrix M and returns a list $[H_{w_1},H_{w_2},\ldots,H_{w_k},H]$ where H is an upper Hessenberg matrix and the H_{w_i} are Householder matrices satisfying
$$H_{w_k}^*\cdots H_{w_1}^* M H_{w_1}\cdots H_{w_k} = H$$
with $k \leqslant n-2$.
10. Write a function `hh_tridiag(M,L)` where M is a hermitian matrix with eigenvalues' list L and returns a list $[H_{w_1},H_{w_2},\ldots,H_{w_k},T]$ such that T

is a tridiagonal matrix and the H_{w_i} are Householder matrices satisfying

$$H_{w_k}^* \cdots H_{w_1}^* M H_{w_1} \cdots H_{w_k} = T.$$

Exercise 3. (Cholesky decomposition and QR algorithm)

1. Write a function `randmat_pd(n,[optional_args])` that takes a positive integer n and returns a Hermitian random positive definite $n \times n$ matrix. The optional argument may be one of `integer`, `rational`, `real`, `complex_rational`, or `complex_real`. *Hint.* First generate a random lower triangular matrix L, then return LL^*.
2. Write a function `chol(A)` that takes a positive definite matrix A and returns a lower triangular matrix L with positive diagonal entries such that $A = L^* L$. Use the recursive algorithm presented in class.
3. Write a function `is_pd(A)` that takes a matrix A and returns `true` or `false` depending on whether A is Hermitian and positive definite, using the recursive algorithm described above. Write a second function `is_pd_sylvester (A)` that returns `true` or `false` according to Sylvester's criterion. Compare the execution time of these two functions on a large matrix using `showtime :true`.
4. Write a function `chol_qr_alg(A,[optional_args])` that takes as input a real positive definite matrix A. The optional argument may be either a positive integer n, in which case the function performs n steps of the Cholesky iteration, or a positive number $\epsilon < 1$, in which case the iteration continues until the sum of squares of the off–diagonal entries of the current iterate is less than ϵ.
5. Write a function `qr_alg(A,[optional_arg])` that takes as input a real positive definite matrix A. The optional argument may be either a positive integer n, in which case the function performs n steps of QR algorithm, or a positive number $\epsilon < 1$, in which case the iteration continues until the sum of squares of the off–diagonal entries of the current iterate is below ϵ.

Exercise 4. (Iterative methods: Jacobi, Gauss-Seidel, SOR, Power method)

1. Write a function `iteration_jacobi(A, b, x_0, [optional_arg])` that takes as input a matrix A, a column vector b, and an initial vector x_0, where b and x_0 may be given either as a list of numbers or as a Maxima column matrix. The optional argument may be either
   - a positive integer n, in which case the function performs n steps of the Jacobi iteration method to solve $Ax = b$ and returns x_n, or
   - a positive number $\epsilon < 1$, in which case the iteration continues until the stopping condition $\|x_k - x_{k-1}\|_2 < \epsilon$ is satisfied and returns x_k.
2. Write a function `iteration_GS(A, b, x_0, [optional_arg])` that takes as input a matrix A, a column vector b, and an initial vector x_0, where

b and x_0 may be given either as a list of numbers or as a Maxima column matrix. The optional argument may be either

- a positive integer n, in which case the function performs n steps of the Gauss–Seidel iteration method to solve $Ax = b$ and returns x_n, or
- a positive number $\epsilon < 1$, in which case the iteration continues until the stopping condition $\|x_k - x_{k-1}\|_2 < \epsilon$ is satisfied and returns x_k.

3. Write a function `iteration_sor(A, b, omega, x_0,[optional_arg])` that takes as input a matrix A, a number ω with $0 < \omega < 2$, a column vector b, and an initial vector x_0, where b may be given either as a list of numbers or as a Maxima column matrix. The optional argument may be either

   - a positive integer n, in which case the function performs n steps of the SOR iteration method to solve $Ax = b$ and returns x_n, or
   - a positive number $\epsilon < 1$, in which case the iteration continues until the stopping condition $\|x_k - x_{k-1}\|_2 < \epsilon$ is satisfied and returns x_k.

4. Write a function `iteration_power_method(A,b_0, [optional_arg])` that takes as input a matrix A and an initial vector b_0 that may be given either as a list of numbers or as a Maxima column matrix. The optional argument may be either

   - a positive integer n, in which case the function performs n iterations of the power method and returns the list $[b_n, \lambda_1]$, where b_n is the vector obtained after n steps and λ_1 is the dominant eigenvalue of A, or
   - a positive number $\epsilon < 1$, in which case the iteration continues until the stopping condition $\|b_k - b_{k-1}\|_2 < \epsilon$ is met, and the function returns the list $[b_k, \lambda_1]$.