MATLAB is a numerical computing environment and programming language, created by MathWorks, that enables you to easily manipulate matrices, plot functions and data, implement algorithms, create user interfaces, and interface with programs in other languages. Although MATLAB specializes in numerical computing, an optional toolbox interfaces with the Maple symbolic engine, allowing it to be part of a full computer algebra system.

A total of 16 licenses are available to use on either NAS HPC systems or NAS-supported desktop systems. To find out how many licenses are currently in use, run:

```
/u/scicon/tools/bin/check_licenses -m (view general licenses only)

/u/scicon/tools/bin/check_licenses -M (view general licenses and compiler/toolboxes)
```

Then, load a MATLAB module:

```
% module load matlab/2016b
```

See the following websites for more information about MATLAB software:

- MATLAB documentation (MathWorks)
- MATLAB (Wikipedia)

See the following Knowledge Base articles for more information about NAS MATLAB licenses:

- MATLAB License Policy
- Using NAS MATLAB Licenses

NAS currently has 16 MATLAB licenses available. To avoid overwhelming the license server, and to reduce contention for the limited number of licenses, PBS batch jobs running on most NAS compute nodes are not allowed to query the MATLAB license server. Therefore, in order to run a MATLAB script in a batch job on most NAS compute nodes, you must first compile the script. Compiled executables do not access the license server.

Note: It is not necessary to compile your MATLAB scripts for batch jobs running on the LDANs or Endeavour—on these systems, there is less risk of overwhelming the MATLAB license server because fewer hosts query the server. Also, you can still run interactive MATLAB jobs on NAS compute nodes without compiling the scripts (it is easier to debug the scripts if they are not compiled into executables).

Steps for Compiling a MATLAB Script

This method is the simplest way to compile a MATLAB script into a standalone executable. The steps are described using the following sample script, findR.m:

```
function R = findR
% findR - given data T, calculate R
%
T = 10;
R = .5*(-9.8)*T
% end of MATLAB code
```

Complete these steps to compile the sample script:

1. Load the modules:

   ```
   % module load matlab
   % module load gcc/9.3 # or later
   ```

2. If this is the first time you are using MATLAB to create an executable, run:

   ```
   % matlab -nodisplay
   >>mbuild -setup
   >>exit
   ```

3. Compile the script into an executable. At the Linux or MATLAB prompt, run:

   ```
   % mcc -m findR.m -o findR
   ```

   Two files are created:
   - An executable: findR
   - A shell script: run_findR.sh

4. To run the executable, add the following lines to your PBS script or run them at the Linux prompt:

   ```
   % module load matlab/
   % module load gcc/9.3
   ./run_findR.sh $MATLAB
   ```

   The output will be:

   ```
   % ./run_findR.sh $MATLAB
   ---------------------------------------------
   Setting up environment variables
   ---
   LD_LIBRARY_PATH is
   .:/nasa/matlab/2024a/runtime/glnxa64:/nasa/matlab/2024a/bin/glnxa64
   :/nasa/matlab/2024a/sys/os/glnxa64:/nasa/matlab/2024a/sys/opengl/lib/glnxa64
   R =
     -49
   ```

Note: If you have several MATLAB scripts, where master.m calls y.m, and y.m calls z.m, make sure that master.m is the first file listed in the command line when you compile the script:

```
% mcc -m master.m y.m z.m
```

In this case, the resulting files are:

- Executable: master
- Shell script: run_master.sh

In order to effectively utilize the limited number of MATLAB licenses on NAS-supported workstations or supercomputing systems:

- Each user is limited to a single interactive MATLAB license.
- When using the MATLAB GUI, after four hours of being idle and not running a job, the license will be revoked.
- Start a second MATLAB GUI on the same host to compile or use toolboxes. Exit the second GUI when you are done with the license to immediately free up the compiler/toolbox license.
- If users are waiting for licenses, a MATLAB process that has been idle for more than ten minutes may be terminated to revoke its license and make it available to another user.

For more information, including alternatives to using MATLAB licenses, see Using NAS MATLAB Licenses.

A limited number of MATLAB licenses are available on NAS-supported workstations and supercomputing systems. To check current MATLAB license usage, run the check_licenses command as follows:

```
/u/scicon/tools/bin/check_licenses -m
(view general licenses only)

/u/scicon/tools/bin/check_licenses -M
(view general licenses and compiler/toolboxes)
```

Important Usage Information

Please note the following information when you use a NAS MATLAB license:

- When the MATLAB compiler/toolbox is run from the GUI, the license is held by the user until the GUI is closed.
- When the MATLAB command-line tools (such as the compiler) are used, the license is held by the user for 30 minutes after the last use. Using the tool again restarts the timer.
- If multiple MATLAB licenses are checked out by a single user, the newest MATLAB process will be terminated.
- The single-license limit only applies to the primary MATLAB license. Licenses such as the compiler or toolboxes do not count toward the limit.

See the MATLAB License Policy for more information.

Alternatives to Using NAS MATLAB Licenses

Alternatives to using MATLAB licenses include:

- Compile non-interactive MATLAB jobs so they do not require a license. See Compiling MATLAB Scripts into Executables to Reduce the Use of Licenses.
- If you have access to a local MATLAB license that you are entitled to use remotely, you can utilize it byrunning applications on NAS systems with your local license.
- Use GNU Octave, an open source alternative to MATLAB. GNU Octave is available on NAS systems. To access it, run the following commands:

  ```
  module avail octave

  ---------- /nasa/modulefiles/pkgsrc/toss4 ----------
  octave/6.4   octave/8.3

  module load octave/8.3
  octave
  ```

UPDATE IN PROGRESS: Starting with version 2.17, **SGI MPT** is officially known as **HPE MPT**. Use the command module load mpi-hpe/mpt to get the recommended version of MPT library on NAS systems. This article is being updated to reflect this change.

MATLAB parametric studies and neural network applications can typically be run in parallel for higher productivity. However, because NAS does not have a license for MATLAB's Parallel Computing Toolbox and only 16 MATLAB licenses are available, NAS users often run these applications serially.

There are two ways to run multiple MATLAB instances in parallel without using multiple licenses:

- Invoking multiple MATLAB instances in PBS interactive mode
- Running MATLAB in parallel with MPI and Fortran

Each license is good for one node and one user, so you can run MATLAB on multiple cores on the same node using one license. To check the availability of MATLAB licenses, run the check_licenses tool as follows:

```
pfe% /u/scicon/tools/bin/check_licenses -m
```

Invoking Multiple MATLAB Instances in PBS Interactive Mode

Use this method if you want to execute a MATLAB function multiple times, and the function's input parameter can be created with simple arithmetic. In this case, you can easily invoke MATLAB in a for loop, as described below. (If the input parameter does not follow a simple arithmetic pattern, see Running MATLAB in Parallel with MPI and Fortran.)

To reduce contention for the limited number of licenses, PBS batch jobs are not allowed to query the MATLAB license server. However, for a simple parametric study like the following example, you can use PBS interactive mode.

Example

In this example, a MATLAB function called sampleFn.m is calculated 24 times with input parameters 2, 4, 6, ..., 48. The sampleFn.m function runs all 24 instances in parallel on the same Haswell node.

First, submit a request for one Haswell node:

```
%> qsub -I -l select=1:mpiprocs=24:model=has -q devel -l walltime=2:00:00
qsub: waiting for job 8022657.pbspl1.nas.nasa.gov to start
Job 8022657.pbspl1.nas.nasa.gov started on Mon Feb 03 15:55:10 PST 2020
.....
PBS r509i0n17 51>
```

From the compute node (r509i0n17), run the following commands to start 24 instances in parallel:

```
PBS r509i0n17 51> source /usr/local/lib/init/global.cshrc
PBS r509i0n17 51> module purge
PBS r509i0n17 51> module load matlab/2017b
PBS r509i0n17 51> foreach i (`seq 1 24`)
foreach?  @ j = $i * 2
foreach? matlab -nodesktop -nodisplay -nosplash -r "sampleFn(${?j?});exit" > out__$i < /dev/null &
foreach? end
PBS r509i0n17 51> wait
```

All processes should return at the same time. There will be 24 files with the nameout__*.

Note: Because there are 24 cores on a Haswell node, no more than 24 MATLAB instances should be run at the same time.

Sample MATLAB function: sampleFn.m

The sampleFn function called in the example above could be any MATLAB function. For example, it might look like this:

```
function D = sampleFn(T)
%
% sampleFn - given data T, return D = 3x3 matrix with entries T
% D = is 3x3 matrix with entries T
%
n=3;
D = T*ones(n,n)
```

Running MATLAB in Parallel with MPI and Fortran

Use this method if you want to execute a MATLAB function multiple times and the input parameter does*not* follow a simple arithmetic pattern. (Otherwise, use the PBS script method described above.)

This method uses an MPI/Fortran program to run multiple MATLAB instances in parallel. Each MPI process in the Fortran program activates the MATLAB engine and calls the MATLAB function you intend to run. Due to differences in the ways Fortran and MATLAB pass arguments, the Fortran code also needs to call additional MATLAB functions to properly transfer argument values.

The MPI/Fortran program is then compiled via MATLAB's mex script and linked with MATLAB and MPI libraries. You can run the resulting executable as a typical MPI executable using the mpiexec command through a PBS job.

Sample MPI/Fortran Code

Suppose you want to call the sampleFn(s) routine described in the script method above, but you want to pass in a real number whose value "s" cannot be obtained by simple arithmetic. Instead, the value will be obtained by a Fortran (or C) routine.

Note: The value of "s" can be any data type.

The sample MPI/Fortran code test.F demonstrates how this method works. The key actions in test.F are to activate the MATLAB engine, facilitate transfer of input, call sampleFn(), and transfer output, as shown in the following steps. Each step indicates a specific line in the test.F code.

1. Line 41 - activate the MATLAB engine:
   `ep = engOpen('matlab ')`
2. Line 49 - define the Fortran pointer of a MATLAB scalar:
   `sPtr = mxCreateDoubleMatrix(1, 1, 0) ! 1x1 matrix is a scalar`
   This scalar is the "s" in the MATLAB function.
3. Line 52 - obtain the parameter to feed into the MATLAB code, sampleFn(s):
   `k = IRAND(rank)`
   In this example, a random integer value "k" for each MPI rank is obtained from a random number generator. (This Fortran value "k" will be converted to value "s" in the MATLAB code.)
4. Line 54 - assign the Fortran value "k" to the pointer sPtr:
   `call mxCopyReal8ToPtr(dble(k), mxGetPr(sPtr), 1)`
   Note that each MPI process has its own value for "k".
5. Line 59 - call sampleFn():
   `if (engEvalString(ep, 'D = sampleFn(s);') .ne. 0) then`
   Each MPI process executes the MATLAB function that has output D. Note that the value of D can be any data type and dimension. In this case, D is a real nxn matrix.
6. Line 65 - assign the MATLAB value D to Fortran pointer DPtr:
   `DPtr = engGetVariable(ep, "D")`
7. Line 66 - instruct each MPI process to obtain the value of the nxn matrix D:
   `call mxCopyPtrToReal8(mxGetPr(DPtr), D, n*n)`

Fortran Code: test.F

Scroll inside the code box to see the whole code. The command lines noted in the steps above are highlighted.

```
 1 #include "fintrf.h"
 2 C
 3 #if 0
 4 !
 5 !    test.F
 6 !    .F file need to be preprocessed to generate .for equivalent
 7 !
 8 #endif
 9 !
10 !    test.F
11 !
12 !    This is a simple program modified from a sample code that
13 !    illustrates how to call the MATLAB Engine functions.
14 !
15 ! NASA Ames Research Center
16 !=========================================================================
17 ! 2015/7/7
18
19      program main
20      implicit none
21      include 'mpif.h'
22 !-------------------------------------------------------------------------
23 !    (pointer) Replace integer by integer*8 on 64-bit platforms
24 !
25      mwpointer engOpen, engGetVariable, mxCreateDoubleMatrix
26      mwpointer mxGetPr
27      mwpointer ep, DPtr, sPtr
28 !-------------------------------------------------------------------------
29 !
30 !    Other variable declarations here
31      integer, parameter :: n=3, M=12
32      double precision D(n,n), Darray(n,n,M), s
33      integer engPutVariable, engEvalString, engClose
34      integer i,k, temp, status, size , rank, ierror
35 !
36      ! MPI initilization
37      call MPI_INIT(ierror)
38      call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierror)
```

```
 39       call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)
 40
        ep = engOpen('matlab ')
 42 !
 43       if (ep .eq. 0) then
 44          write(6,*) 'Can"t start MATLAB engine'
 45          stop
 46       endif
 47 C
 48 !    Define the pointer of a Matlab scalar
       sPtr = mxCreateDoubleMatrix(1, 1, 0)    ! 1x1 matrix is scalar
 50
 51 !    Obtain the value of k from a routine (here it is just random generator
       k = IRAND(rank)
 53 !      Copy Fortran value k to Matlab array mxGetPr(sPtr)
        call mxCopyReal8ToPtr(dble(k), mxGetPr(sPtr), 1)
 55       status = engPutVariable(ep, 's', sPtr)
 56       if (status .ne. 0) write(*,*)"FAIL!!! engPutVariable s"
 57 !
 58 !      Compute via matlab function : sampleFn
        if (engEvalString(ep, 'D = sampleFn(s);') .ne. 0) then
 60         write(6,*) 'engEvalString failed'
 61         stop
 62       endif
 63 !
 64 !    Bring pointer back to Fortran
       DPtr = engGetVariable(ep, "D")
       call mxCopyPtrToReal8(mxGetPr(DPtr), D, n*n)
 67
 68 !     The MPI master can gather all D's to Darray
 69 !     via MPI_Send/Recv or MPI_Gather or MPI_Reduce
 70 !     Darray(:,:,k) <= D(:,:)
 71 !
 72     print *, 'Print results created by Matlab :'
 73        do i=1,n
 74        write(*,*) D(:,i)
 75        enddo
 76        write(*,*) " "
 77 !
 78 !
 79     call mxDestroyArray(sPtr)
 80     call mxDestroyArray(DPtr)
 81     status = engClose(ep)
 82 !
 83     if (status .ne. 0) then
 84        write(6,*) 'engClose failed'
 85        stop
 86      endif
 87      call MPI_Finalize(ierror)
 88 !
 89      stop
 90      end program main
```

Compiling the MPI/Fortran Code

This example uses mex, /usr/bin/gfortran, matlab/2016b, and mpi-sgi/mpt.2.12r26. to compile the program. You can choose other versions of compiler, MATLAB, and MPT.

To compile, run:

```
mex -v LDFLAGS="-lmpi" FFLAGS=" -fexceptions -fbackslash \
   -I/nasa/mw/2016b/extern/include -I/nasa/sgi/mpt/2.12r26/include" \
   -f /nasa/mw/2016b/bin/engopts.sh test.F
```

The resulting executable is called test.

If you want to use the Intel Fortran compiler, add FC=ifort and replace -fbackslash with -assume bscc in the mex command, as follows:

```
mex FC=ifort -v LDFLAGS="-lmpi" FFLAGS=" -fexceptions -assume bscc \
   -I/nasa/mw/2016b/extern/include -I/nasa/sgi/mpt/2.12r26/include" \
   -f /nasa/mw/2016b/bin/engopts.sh test.F
```

Running the Executable

Within a PBS script or an interactive PBS session, load the necessary MATLAB, MPI and compiler modules, and set the LD_LIBRARY_PATH environment variable to the correct setting before executing mpiexec as follows:

```
% setenv LD_LIBRARY_PATH  "/nasa/mw/2016b/bin/glnxa64:   \
    /nasa/mw/2016b/sys/os/glnxa64:$LD_LIBRARY_PATH"
```

```
% mpiexec -np 4 ./test
```

In this example, four MATLAB instances are running.

IMPORTANT: As a courtesy to other MATLAB users, please do not use more than two MATLAB licenses (i.e., using more than 2 nodes) per PBS job.