Introduction to MATLAB for Numerical Computing

Hongli Zhao

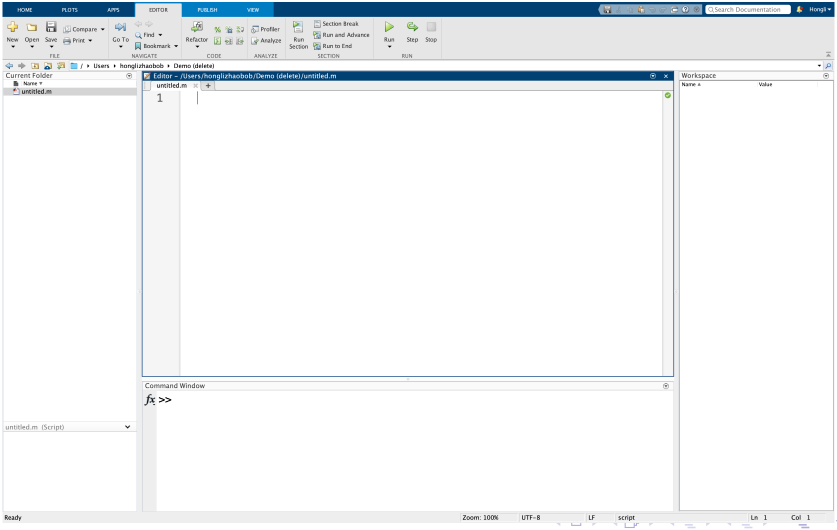
Discussion 1, 01/18/2023

Getting MATLAB

- The University of Chicago provides free MATLAB license for all registered students or staff. You can sign in and download MATLAB by logging onto the MATLAB website with your @uchicago.edu email.

MATLAB Layout (demo)

There are 4 elements of the MATLAB window.



- **Current folder**: Shows a list of files and folders in the *present working directory*. To access files in other folders, one needs to either add its path or switch present folders.
- **Workspace**: Shows a list of objects/variables currently defined in your session.
- **Command Window**: The window where you can interact with objects at any point of the program (e.g. define new variables, run code, perform calculations). The *command history* can be accesses using the up arrow.
- **Editor**: Edits/previews code files, especially `.m`, `.mat` files.

1. `.m` files contain scripts/functions. `.mat` files contain MATLAB-formatted data objects, and can be saved or read-in using common languages such as MATLAB/Python/Julia.
2. The layout can be reset to default by clicking Home > Layout > Default
3. Commands are printed by default, and the result saved in a default variable called `ans`. To suppress this, add semicolon at the end (e.g. `a = 2;`)
4. Use the `clear` command to release variables to garbage collection.

Math operations in MATLAB (demo)

Useful Operations

- Basic arithmetics: `+`, `-`, `*`, `/`, `sqrt()`, `log()`, `exp()`
- Booleans: `true`, `false`, `&`, `|`[a]
- Numerical comparison: `<`, `>`, `<=`, `>=`, `==`, `~=` (note the inequality)
- The dot operator (very useful): For example, `.*`, `.+`, `.-`, which implements broadcasting on arrays.
- Built-in variables: `pi`, `inf`, `nan` ($\pi, +\infty, 0/0$)

[a] There are also `&&`, `||`, which implement short-circuiting.

Where to get help?

1. Type `help func_name` to access the documentation.
2. Ask questions on the MathWorks community

Common Data Types (demo)

This slide will introduce some common data types/data structures and examples for interacting with them.

- Arrays/Multidimensional arrays:
  - Initialize a matrix: `arr = zeros(10);`. Same as `zeros(10, 10)`
  - Assignment: `arr(4, 2) = 2;`, assigns row 4, column 2 position a numerical value of 2.
  - Initialize a vector: `arr = zeros(10, 1);`
  - Accessing the end of the array: `arr(end) = -1;`. Second to last: `arr(end-1) = -2;`
  - General *d*-dimensional arrays: `arr = zeros(n, n, n);`

Use commas to separate elements on the same row; use semicolons to skip rows.

```
a = [1,2; 3,4; 0,6]; % creates a 3 x 2 matrix
b = randn(3, 2); % creates a 3 x 2 N(0,1) sample
c = a + b; % matrix addition
```

When *a*, *b* are matrices, multiplication operator $\star$ defaults to matrix multiplications. For elementwise multiplication, need to use `.*`.

```
d = a * b; % errors
d = a .* b % correct, now d(i,j)==a(i,j)*b(i,j)
```

Transposing:

```
a_transpose = a';
```

Broadcasting operations:

```
a = ones(5, 1); % creates a vector of all 1's
b = 1:5; % creates 1, 2, 3, 4, 5
% note the difference of the following
c1 = b*a; % inner product
c2 = b.*a; % outer product
```

Broadcasting operations (matrix):

```
% block-repeats the vector 1 x 5
A = repmat(([1, 2, 3, 4, 5])', 1, 5);
c = [1; 2; 3; 4; 5];
% subtract the column from each column of A
% (expects 0)
B = A - c;
```

Slicing:

```
A = randn(100);
% get first column
c = A(:, 1);
% get last row
d = A(end, :);
% get sub matrix
E = A(2:10, 5:20); % submatrix of size 9 x 16
% block assignment
A(2:10, 5:20) = randn(9, 16);
% get diagonal
diag(A)
% if diagonal is called on a vector,
% creates a matrix
diag([1,2,3])
```

Solving a linear system:

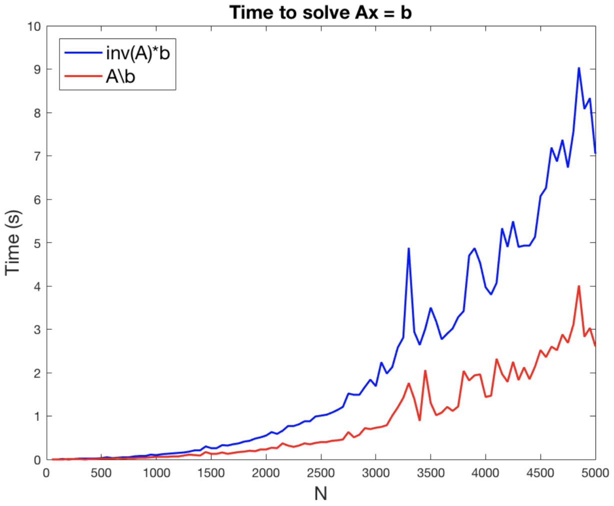
```
A = randn(100);
b = randn(100, 1);
% The preferred way
sol = A\b;
% The bad way (in many senses)
sol = inv(A)*b;
```

Depending on the structure of A, $A^{-1}b$ can be computed more efficiently with different algorithms. The backslash operator implements case checking. See the documentation here, whereas `inv(A)` computes the matrix inverse explicitly, this can be bad in two ways:

1. Does not take advantage of special structure of A
2. Scales error if A is ill-conditioned:

$$\frac{\|\Delta x\|}{\|x\|} \leq \kappa(A) \cdot \left(\frac{\|\Delta A\|}{\|A\|} + \frac{\Delta b}{\|b\|} \right) \tag{1}$$

Performance comparison



- Strings: typically used for printing results. Useful commands include `strcat`, `fprintf`, `sprintf`, `num2str`, `str2num`
- Cell/Cell arrays: A cell is a generic, versatile data type that represents a container.

```
a = cell(4, 1); % creates a cell array
% putting things into the cell array
a{1}= "hello world";
a{2}= randn(10,10);
a{3}= cell(5, 1);
% slicing
a{1:2}
% clearing
a{3}= [];
```

  Cell arrays are quite flexible and can be operated on like an array, in particular, it can contain other cell arrays. This can be very useful for designing recursive algorithms.

- A **structure array**, or struct can be considered as a container with field names. Fields are accessible through the dot notation, and can be nested.

```
fdm.name = "Finite difference";
fdm.step_size = 0.01;
% this is a lambda function
fdm.func = @(x) 3*x;
fdm.inner_fdm = struct;
```

See more data types in the documentation.

Plotting

Effective visualization is another crucial component (other than correctness) to numerical computing. It helps us identify inaccuracies through intuition, and present results succinctly.

- We will introduce the plotting features used to reproduce the following three plots. See `example_plots.m`

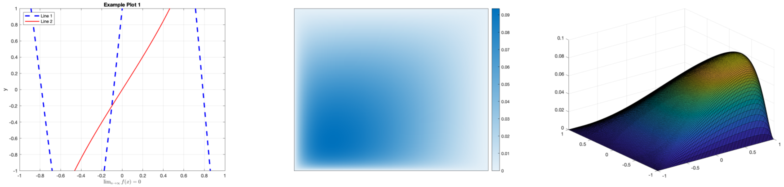


Figure: Line plot, heatmap, and surface plots.

Practice: Advection-Diffusion-Reaction equation in 2d

Let's walk through an example using what we just saw. This example solves the following equation:

$$-\nabla \cdot \kappa \nabla u + v(x) \cdot \nabla u + cu = f$$

with Dirichlet boundary condition:

$$u\big|_{\partial\Omega} = 0$$

here let $\Omega = [-1, 1]$, $v(x, y) = [x + 0.5, y + 0.5]^T$, $\kappa = c = 1$, $f = e^{-\frac{1}{2}(x^2+y^2)}$. The source-term has the following shape.

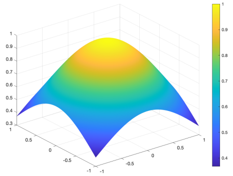


Figure: Plot of $f(x) = \exp\left(-\frac{1}{2}(x^2 + y^2)\right)$

Let's review the finite difference rule:

$$\Delta u = u_{xx} + u_{yy}$$

$$\approx \frac{1}{h^2}\left[(U_{i+1,j} - 2U_{i,j} + U_{i-1,j}) + (U_{i,j+1} - 2U_{i,j} + U_{i,j-1})\right] + O(h^2)$$

$$v(x,y)\cdot\nabla u = v_1 u_x + v_2 u_y$$

$$\approx \frac{v_i^{(1)}}{2h}(U_{i+1,j} - U_{i-1,j}) + \frac{v_j^{(2)}}{2h}(U_{i,j+1} - U_{i,j-1}) + O(h^2)$$

where $v^{(1)}(x) = x + 0.5, v^{(2)}(y) = y + 0.5$. And cu is simply $cU_{i,j}$ when discretized.

Let us collect the matching terms. We arrive at:

$$\Big(-\frac{\kappa}{h^2}+\frac{v_i^{(1)}}{2h}\Big)U_{i+1,j}+\Big(-\frac{\kappa}{h^2}-\frac{v_i^{(1)}}{2h}\Big)U_{i-1,j}+\Big(c+\frac{4\kappa}{h^2}\Big)U_{i,j}\cdots$$

$$+\Big(-\frac{\kappa}{h^2}+\frac{v_j^{(2)}}{2h}\Big)U_{i,j+1}+\Big(-\frac{\kappa}{h^2}-\frac{v_j^{(2)}}{2h}\Big)U_{i,j-1}=f_{i,j}$$

which in (after linearizing) matrix form should be a block tridiagonal matrix:

$$\begin{bmatrix} D_{\mathbf{x},h} & S_{1,h}^{+} & 0 & \cdots & \cdots & 0 \\ S_{2,h}^{-} & D_{\mathbf{x},h} & S_{2,h}^{+} & \cdots & \cdots & 0 \\ 0 & S_{3,h}^{-} & D_{\mathbf{x},h} & S_{3,h}^{+} & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \cdots & \vdots \\ 0 & \cdots & \cdots & \cdots & \ddots & S_{N-1,h}^{+} \\ 0 & \cdots & \cdots & \cdots & S_{N,h}^{-} & D_{\mathbf{x},h} \end{bmatrix}$$

with the following definitions:

$$\begin{cases} S_{j,h}^{\pm} = \text{diag}\big[-\frac{\kappa}{h^2} \pm \frac{v_j^{(2)}}{2h} \big] \\ D_{\mathbf{x},h} = \text{tridiag}\Big[\big(-\frac{\kappa}{h^2} - \frac{v_{\mathbf{x}[1:n-1]}{}^{(1)}}{2h} \big), \big(c + \frac{4\kappa}{h^2} \big), \big(-\frac{\kappa}{h^2} + \frac{v_{\mathbf{x}[2:n]}{}^{(1)}}{2h} \big) \big) \Big] \end{cases}$$

where $v_{\mathbf{x}[1:m]}^{(1)}$ denotes the vector $[v^{(1)}(x_1), \ldots, v^{(1)}(x_m)]^T$.

- The example is implemented in the code file `adr2d.m`. Play with it and try to understand what each part is doing. Please send an email to me or Jeremy for any questions.