Common uses and performance

Common uses

As seen in the introduction, Matlab® is commonly used for

- Algorithm development
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- ...

Some functions and operators have then been designed specifically to ease the resolution of problems frequently arising in those fields

Data analysis

When dealing with large sets of data, computing the mean-value, standard deviation and variance can be done with

`mean`, `stdvar`, `cov`

If the data is grouped, the function `grpstats` computes the summary statistics of each group in a matrix, dataset array or table

```
>> data = rand(50,1);
>> groups = randi(3,50,1);
>> m = grpstats(data, groups, 'mean')
>> ugrps = unique(groups)
```

creates a 50*1 vector with elements between 1 and 3

does the mean for the elements from the same group

returns the same data as in groups but with no repetitions

Linear system of equations

A recurrent problem in scientific computing is

> Given A an invertible $n \times n$ matrix, b an $n \times 1$ vector, solve the system $Ax = b$

Matlab® offers two ways to solve it

```
>> x=inv(A)*b     % Depreciated, can fail
>> x=A\b          % Optimised, handles overdetermination
```

In practice, there is no exact solution for overdetermined systems of equation (A is a $m \times n$ matrix, $m > n$, b is a $m \times 1$ vector)

→ Not computing an exact solution, the operator `\` minimizes the error `norm(A*x-b)`

Complexity

The *complexity* of an algorithm is a measure that estimates the number of single operations (+, -, *, /) required to end its execution

Let $f, g : \mathbb{N} \to \mathbb{R}$ be functions on natural numbers. Then it is said $f = \mathcal{O}(g)$ if there exists $c \in \mathbb{R}$, s.t.

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} < c$$

- The number of FLoating point Operations Per Second that a computer can do is given in *flops*
- Given the algorithm's complexity and the size of the initial data, the running time is known *a-priori*

Examples:

Search the biggest element of a list of length n: $\mathcal{O}(n)$
Naive multiplication of two $n \times n$ matrices: $\mathcal{O}(n^3)$

Computational efficiency measurement

It is possible to time the execution of blocks of code to compare the running efficiency of two coding possibilities or identify the slowest part of a code with the keywords `tic` and `toc`

```
>> A=rand(2000);
>> tic;              % Starts the timer
>> svd(A);           % Instructions to time
>> toc               % Stops the timer and
>>                   % returns the elapsed time
```

It is possible to start multiple timers

```
>> T1=tic;
>> % wait
>> T2=tic;
```

```
>> elapsed1=toc(T1)
>> elapsed2=toc(T2)
```

Coding habits and performance killing

Try in the prompt the following instructions, that in the end create the same vector

```
>> tic; x=[1:100000]; toc
>> tic; x=zeros(1,100); for
>> k=1:100000, x(k)=k; end, toc
```

What do you observe? What could be the reason(s)?

Coding habits and performance killing

Try in the prompt the following instructions, that in the end create the same vector

```
>> tic; x=[1:100000]; toc
>> tic; x=zeros(1,100); for
>> k=1:100000, x(k)=k; end, toc
```

What do you observe? What could be the reason(s)?

→ Observation: the speed changes by a factor 200!

→ Issue: the variable `x` changes at each iteration its size

→ Issue: Matlab® is an interpreted language: slow loops

Performance

Best practice

- Prefer algorithm having low complexity
- Use vectorisation whenever possible
- Try to avoid loops
- Do not increase the variables' dimension inside loops

Symbolic computations

Definition of symbolic variables

Though **usually not advised**, *symbolic computations* are possible. Symbolic variables should be declared as such before their first use

```
>> syms x y
>> class(x)
```

Computations are then possible with symbols, *i.e.* with no specific value assigned to the *e.g.* `x` ot `y`

```
>> cc=sqrt(x+y)
>> class(cc)
>> f(x) = sqrt(x)
```

Symbolic representation of numbers

A variable's value can be assigned by its symbolic representation

```
>> a=sym('123312312312312.123213123123213123123123')
>> sym('10/3')
>> pp=sym('pi'), sin(pp)
```

The precision of the arithmetic evaluation of a symbolic expression is up to Matlab®'s interpretation unless explicitely specified

```
>> vpa(pp)
>> vpa(pp,500)
```

When representing a number, a sym variable can be turned double

```
>> double(pp)
```

```
>> double(cc)
```

Symbolic representation of matrices

A matrix can also be represented symbolically

```
>> A=sym('A',[2,2])
```

Algebraic quantities are then computed symbolically, returning their expression in function of the symbolic matrix's coefficients

```
>> inv(A)
>> eig(A)
```

Using symbolic expressions (1/2)

When working with symbolic variables acting as symbols and not represented numbers, one can:

- replace or substitute values in an expression
  defining a symbolic variables representing a number

```
>> a=subs(sin(x),x,3*pi/2)    symbolic substitution
```

- simplify expressions

```
>> a=sin(x)^2+cos(x)^2
>> simplify(a)
```

- perform algebraic operations

```
>> expand((1+x)^10)
>> factor(x^10-1)
```

Using symbolic expressions (2/2)

Many analysis operations that can be used on symbolic expressions are already implemented in the toolbox

```
>> limit(x*sin(x),x,0)
>> diff(x^5,x,2)
>> int(x^5,x)
>> int(1/x,x,1,2)
>> symsum(k^2,k,0,10)
>> symprod(1-1/k^2,k,2,Inf)
```

$\log(2)$

Note: Those functions take different kind of arguments, retrieve their details in the help

Solving symbolic equations

Solve algebraic equation by writing naturally their expression after having declared the variable to solve for as a symbolic one

```
>> solve(x^3+2*x==1)
>> [a,b]=solve(x+y==1,...
>>                x-y==-1)
```

```
>> x = sym('1/2')
>> solve(x == 4)
```

Differential equations can be solved in a similar manner

```
>> syms y(t)
>> dsolve(diff(y,2)==-y,y(0)==0)
```

Solving symbolic equations

Solve algebraic equation by writing naturally their expression after having declared the variable to solve for as a symbolic one

```
>> solve(x^3+2*x==1)
>> [a,b]=solve(x+y==1,...
>>                x-y==-1)
```


```
>> x = sym('1/2')
>> solve(x == 4)
```

→ Error: x had a value

Differential equations can be solved in a similar manner

```
>> syms y(t)
>> dsolve(diff(y,2)==-y,y(0)==0)
```

Symbolic visualisation

Plotting with the symbolic library is similar to the classical Matlab® framework

```
>> ezplot(sin(x), [0,3*pi])
>> syms y
>> ezsurf(sin(x)*cos(y),[0,3*pi,0,2*pi])
```

It is particularly useful for anonymous functions

```
>> ezplot(@(s) sin(s),[0,2*pi])
>> ezplot(@cos,[0,2*pi])
```

Exercises

Exercises: Performance

Exercise

1. Write a program which measures (approximatively) the complexity of a matrix-multiplication

   Hint: $f(n) \approx Cn^k \Rightarrow \frac{f(n_1)}{f(n_2)} \approx \left(\frac{n_1}{n_2}\right)^k$

2. Write a program which measures the complexity of the operation `svd`.

Exercises: Symbolic Math Toolbox

Exercise

3. Compute the 57-th number after the comma of the Euler number
4. Compute the integrals of
   a) $sin(3x)\ cos(5x)$
   b) $x\ ln(x)$ on $[2, 5]$
5. Compute the limit for $x \to 0$ of $sin(x)\ (1 - cos(2x))/x^3$
6. Compute the limit for the series $\sum_{i=1}^{\infty} \frac{(-1)^{n+1}}{n}$

Thank you and enjoy programming!