Numerical Methods (NUM101) — Coursework 3

This coursework consists of one part. It is worth 10% of the credits for this unit. The maximum number of marks for this coursework is 10.

Deadline	hand-in or upload?
• 27 Mar (Tue)	23:30 upload to Victory Assignment `CW3`
• to be announced (near end of term)	hand-in of printouts at CAM office

Instructions and rules

- Material to be uploaded to Victory: Matlab function file `Vigenere.m`, and, if you need them, function files of other functions that you call inside `Vigenere.m`.
- The uploaded code will be tested by me and receive a mark that is provisional until I mark the printout at the end of term.
- Scripts and functions that generate Matlab errors receive provisionally 0 marks.
- Scripts and functions that do not perform correctly for all valid arguments due to minor errors receive a partial credit of at most 30%. Affected students will have the opportunity to upload a corrected submission at a later deadline to improve their partial credit (see list below).
- Credit (see list of possible reasons for additional point deductions below):

  10 Code performs computation correctly and efficiently for all valid input arguments;

  6–8 Code performs computation mostly correctly in first submission but has problems[1], *and* all errors are corrected in second submission;

  4–5 Code does not perform computations correctly on first submission but could be made to work with minor corrections, *and* all errors are corrected in second submission;

  3 Code does not perform computations correctly in first submission, and second submission still contains errors or is missing.

  ≤ 2 First submission contained errors that require substantial changes to fix. In this case *no second submission is possible*!

- Additionally, points will be deducted for:
  - missing semicolons (the function echoes its calculations);
  - poorly structured code, for example, if some statemens are unreachable, or variables are introduced and assigned but never used;
  - misspelled main file name: only the name `Vigenere.m` is accepted (note the capitals and the suffix).
- Students who collaborate on their coursework **must declare their collaboration by email to** `jan.sieber@port.ac.uk` **before submission deadline**. The mark achieved by the collaborative work will be **split evenly** between the students (rounded down if necessary). All submissions will be cross-checked against each other for signs of plagiarism (including collusion). Evidence for plagiarism, or collaboration without prior declaration, will result provisionally in 0 marks and will be subjected to further investigation.
- For questions, clarifications and further help contact:

  Jan Sieber (`jan.sieber@port.ac.uk`, office LG.146).

[1] for example, the function works correctly most of the time but fails for some valid arguments

Question 1: `Vigenere` — encrypt with the Vigenére cipher **10 marks**

The Vigenére cipher is a simple ancient cipher (invented by Bellaso in 1553 according to `http://en.wikipedia.org/wiki/Vigenere_cipher`). Its key is a word or sentence, say, for example, `'ROME'`. It takes a message, say, `'AVECAESAR'`, and shifts every letter of the message to the right in the alphabet. How far, is determined by the position of the letter in the key. In the example, if we use the letters `['A':'Z']` as our alphabet, the letters of the key have positions `[17,14,12,4]` in the alphabet (we start counting from 0). This means that the Vigenére cipher will work as shown in Table 1

1	message	A	V	E	C	A	E	S	A	R
2	position in alphabet (start count at 0)	0	21	4	2	0	4	18	0	17
3	key (repeated if needed)	R	O	M	E	R	O	M	E	R
4	position of key letters (start count at 0)	17	14	12	4	17	14	12	4	17
5	shift (add row 2 and 4)	17	35	16	6	17	18	30	4	34
6	wrap (take modulo alphabet length)	17	9	16	6	17	18	4	4	8
7	encrypted message	R	J	Q	G	R	S	E	E	I

Table 1: Tabular procedure for Vigenére cipher. Decryption does the same but shifts to the left (subtracting row 4 from row 2 in row 5).

Your function is a minor extension of the principle in Table 1: it has to work for arbitrary alphabets. Step-by-step instructions:

1. Create a new `m` file and save it as `Vigenere.m`. This file will contain the function `Vigenere`.
2. The first line of the file `Vigenere.m` has to look like this:

```
function code=Vigenere(message,key,alphabet)
```

Inputs

- `message` Text to be encrypted. This is a string (a row vector of characters, see hints for examples).
- `key` Key for encryption and decryption. This is a non-empty string for encryption, or the negative of a non-empty string for decryption (see hints for examples).
- `alphabet` Alphabet of valid characters. This is a string of distinct characters. Each character in `message` and in `key` is from `alphabet`. How much each letter from `key` shifts, is decided by its position in `alphabet` (first letter of `alphabet` shifts by 0).

Output

- `code` A string: encrypted or decrypted message depending on the sign of `key` (see hints for examples). It has the same length as the input `message`.

Input `key`	Output `code`
`key` is contained in `alphabet`	encrypted `message`
`-key` is contained in `alphabet`	decrypted `message`

Add your code and test your function on some examples (see hints below).

3. Upload the file `Vigenere.m` and, if necessary, other files that contain functions which you call inside `Vigenere` as attachments to Victory. The Victory assignment is called `CW3`.

Question 1 continues overleaf

Hints and further instructions

- If the first letter of `alphabet` is in `key` it has to shift by zero. When `key` is shorter than `message`, `key` is used repeatedly (see Table 1: `'ROME'` is shorter than `'AVECAESAR'`, thus, one uses `'ROMEROMER'` for encryption. With these two conditions all correct implementations should give the *same* outputs for given inputs.
- Below are examples of how your function should behave on the command-line. The example from Table 1 would work out as follows on the command line:

```
>> encrypted=Vigenere('AVECAESAR','ROME','A':'Z')
encrypted =     'RJQGRSEEI'
>> decrypted=Vigenere(encrypted,-'ROME','A':'Z')
decrypted =     'AVECAESAR'
```

  Observe the minus sign in front of the key for decryption. The next example permits whitespace and commas as part of the alphabet.

```
>> encrypted=Vigenere('AVE CAESAR, MORITURI TE SALUTANT',...
                                 'ROME',['A':'Z',', '])
encrypted = RHQDTOQWRDKDBABMIGBMQFQDHOXYIOZX
>> decrypted=Vigenere(encrypted,-'ROME',['A':'Z',', '])
decrypted = AVE CAESAR, MORITURI TE SALUTANT
```

  Note that, if we re-arrange the alphabet, we get a different encrypted message. In the following example comma and whitespace are at the start of the alphabet:

```
>> encrypted=Vigenere('AVE CAESAR, MORITURI TE SALUTANT',...
                       'ROME',[', ','A':'Z'])
encrypted = TJSFVQSYTFMFDCDOKIDOSHSFJQZ,KQ Z
>> decrypted=Vigenere(encrypted,-'ROME',[', ','A':'Z'])
decrypted = AVE CAESAR, MORITURI TE SALUTANT
```

- You can assume that
  - `message` and `key` (or `-key`) are contained in `alphabet`;
  - all characters in `alphabet` are distinct;
  - `alphabet` and `key` are not empty.

  Thus, you do not have to check the validity of your arguments.
- An empty `message` is valid. In this case, the output `code` should also be empty.
- Keep your file `Vigenere.m` anonymous because it may be passed on to other students for testing, evaluation or use. That is, do not put your name or student ID into the comments.