Arrays and Matrices

1. Sieve of Eratosthenes
   - counting primes
   - arrays in Python
2. Matrices
   - matrices as lists of arrays
   - finding the minimum
   - finding saddle points

MCS 275 Lecture 3
Programming Tools and File Management
Jan Verschelde, 13 January 2017

Arrays and Matrices

1. Sieve of Eratosthenes
   - counting primes
   - arrays in Python

2. Matrices
   - matrices as lists of arrays
   - finding the minimum
   - finding saddle points

the Sieve of Eratosthenes

Input/Output specification:

input	:	a natural number n
output	:	all prime numbers $\leq n$

Running factorization in primes for all numbers $\leq n$ is too expensive.

Sieve of Eratosthenes:

1. Boolean table `isprime[i]` records prime numbers: if `i` is prime, then `isprime[i] == True`, otherwise `isprime[i] == False`.
2. All multiples of prime numbers are not prime: for all `isprime[i]`: `isprime[i*k] = False`.

all primes less than 10

T = True, F = False

i	2	3	4	5	6	7	8	9	10
0	T	T	T	T	T	T	T	T	T
2	T	T	**F**	T	**F**	T	**F**	T	**F**
3	T	T	F	T	F	T	F	**F**	F

Initially, all entries in the table are `True`.

The algorithm uses a double loop:

1. the first loop runs for i from 2 to n;
2. the second loop runs only if i is prime,
   setting to `False` all multiples of i.

Be more efficient, first loop: `while (i < n//i)`.

Arrays and Matrices

1. Sieve of Eratosthenes
   - counting primes
   - arrays in Python
2. Matrices
   - matrices as lists of arrays
   - finding the minimum
   - finding saddle points

Arrays in Python

Arrays are

- sequences of fixed length;
- filled with objects of the same type.

Compared to lists (variable length, heterogeneous), arrays are more memory efficient, faster access.

Available in Python via the module `array`.
The module `array` exports the class `array`.

creating arrays from lists

```
>>> from array import array
>>> L = range(2, 7)
>>> a = array('b', L)
array('b', [2, 3, 4, 5, 6])
```

The `'b'` stands for signed integer, one byte.

Selecting and slicing:

```
>>> a[1]
3
>>> a[1:3]
array('b', [3, 4])
```

The current memory info is obtained as

```
>>> a.buffer_info()
(28773520, 5)
```

types of entries

The types are restricted to numerical types.

Type code	C Type	#bytes
'b'	signed integer	1
'B'	unsigned integer	1
'u'	Unicode character	2
'h'	signed integer	2
'H'	unsigned integer	2
'i'	signed integer	2
'I'	unsigned integer	2
'l'	signed integer	4
'L'	unsigned integer	4
'q'	signed integer	8
'Q'	unsigned integer	8
'f'	floating point	4
'd'	floating point	8

from arrays to lists

```
>>> from array import array
>>> L = range(2,7)
>>> A = array('b', L)
>>> A
array('b', [2, 3, 4, 5, 6])
>>> type(A)
<class 'array.array'>
```

With the method `tolist`,
we convert the array to a list:

```
>>> K = A.tolist()
>>> K
[2, 3, 4, 5, 6]
>>> type(K)
<class 'list'>
```

arrays and strings

Python session continued ...

```
>>> s = A.tostring()
>>> s
b'\x02\x03\x04\x05\x06'
```

In reverse, we can make an array from a string `s`:

```
>>> B = array('b')
>>> B.fromstring(s)
>>> B
array('b', [2, 3, 4, 5, 6])
```

initializing the sieve

T = True, F = False

i	2	3	4	5	6	7	8	9	10
0	T	T	T	T	T	T	T	T	T

Initially, all entries in the table are `True`.

```
def prime_sieve(nbr):
    """
    Returns all primes less than nbr.
    """
    isprime = array('B')
    for _ in range(nbr+1):
        isprime.append(1)
```

the function `prime_sieve` continued ...

The algorithm uses a double loop:

1. the first loop runs for *i* from 2 to *n*;
2. the second loop runs only if *i* is prime, setting to `False` all multiples of *i*.

```
i = 2
while i < nbr//i+1:
    if isprime[i] == 1:
        for j in range(i, nbr//i+1):
            isprime[i*j] = 0
    i = i + 1
```

Why is `nbr//i+1` ***necessary?***

the function `prime_sieve` continued ...

Collecting the primes from the sieve:

```
primes = array('l')
for i in range(2, nbr+1):
    if isprime[i] == 1:
        primes.append(i)
return primes
```

the complete function

```
def prime_sieve(nbr):
    """
    Returns all primes less than nbr.
    """
    isprime = array('B')
    for _ in range(nbr+1):
        isprime.append(1)
    i = 2
    while i < nbr//i+1:
        if isprime[i] == 1:
            for j in range(i, nbr//i+1):
                isprime[i*j] = 0
        i = i + 1
    primes = array('l')
    for i in range(2, nbr+1):
        if isprime[i] == 1:
            primes.append(i)
    return primes
```

sieve.py as a script and module

```
def main():
    """
    Prompts the user for a natural number n and
    prints all primes less than or equal to n.
    """
    nbr = int(input('give a positive number : '))
    primes = prime_sieve(nbr)
    count = primes.buffer_info()[1]
    print('#primes = ', count)
    print(' primes = ', primes)

if __name__ == "__main__":
    main()
```

Why are the last two lines useful?

```
>>> from sieve import prime_sieve
>>> p = prime_sieve(100)
```

Arrays and Matrices

1. Sieve of Eratosthenes
   - counting primes
   - arrays in Python
2. Matrices
   - matrices as lists of arrays
   - finding the minimum
   - finding saddle points

matrices as lists of arrays

Python does not have a two dimensional array type.
Instead we can store a matrix as a list of rows,
where the data on each row is stored in an array.

Consider a 5-by-5 matrix of random two digit numbers:

```
>>> from array import array
>>> from random import randint
>>> A = [array('b',[randint(10, 99)
... for _ in range(5)])
... for _ in range(5)]
>>> for row in A: print(row)
...
array('b', [23, 62, 85, 82, 38])
array('b', [68, 54, 18, 16, 37])
array('b', [91, 70, 88, 42, 56])
array('b', [42, 61, 90, 91, 41])
array('b', [40, 13, 19, 66, 54])
```

selecting entries

```
>>> for row in A: print(row)
...
array('b', [23, 62, 85, 82, 38])
array('b', [68, 54, 18, 16, 37])
array('b', [91, 70, 88, 42, 56])
array('b', [42, 61, 90, 91, 41])
array('b', [40, 13, 19, 66, 54])
```

To select the row with index 2:

```
>>> A[2]
array('b', [91, 70, 88, 42, 56])
```

To select the element in column 3 or row 2:

```
>>> A[2][3]
42
```

matrices from functions

A 5-by-5 matrix whose (i,j)-th element is $i+j$:

```
>>> from array import array
>>> f = lambda x,y: x+y
```

The function `f` defines the (i,j)-th entry.
We use `f` in a doubly nested list comprehension to define a list of rows:

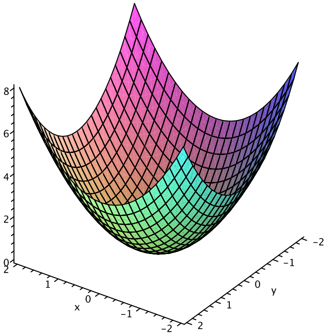
```
>>> L = [array('b',[f(i,j) for i in range(5)])
... for j in range(5)]
>>> for row in L: print(row)
...
array('b', [0, 1, 2, 3, 4])
array('b', [1, 2, 3, 4, 5])
array('b', [2, 3, 4, 5, 6])
array('b', [3, 4, 5, 6, 7])
array('b', [4, 5, 6, 7, 8])
```

paraboloids, for a geometric test

The simplest mathematical description of a paraboloid

$$z = x^2 + y^2$$

$(0, 0, 0)$ is the minimum; plotted (with Maple) as



a sampled paraboloid

In the setup, we want a

- a 10-by-10 matrix of integer valued points;
- where the minimum occurs at row 5, column 5.

Therefore, we sample $(x-5)^2 + (y-5)^2$,
for x and y both ranging from 0 till 9.

```
def test():
    """
    Tests the findmin on the values on a paraboloid.
    """
    from array import array
    paraboloid = lambda x, y: (x-5)**2+(y-5)**2
    data = [array('b', [paraboloid(i, j) \
        for i in range(10)]) \
        for j in range(10)]
    print('looking for a minimum in ')
    for row in data:
        print(row)
```

what `test()` prints

```
looking for a minimum in
array('b', [50, 41, 34, 29, 26, 25, 26, 29, 34, 41])
array('b', [41, 32, 25, 20, 17, 16, 17, 20, 25, 32])
array('b', [34, 25, 18, 13, 10, 9, 10, 13, 18, 25])
array('b', [29, 20, 13, 8, 5, 4, 5, 8, 13, 20])
array('b', [26, 17, 10, 5, 2, 1, 2, 5, 10, 17])
array('b', [25, 16, 9, 4, 1, 0, 1, 4, 9, 16])
array('b', [26, 17, 10, 5, 2, 1, 2, 5, 10, 17])
array('b', [29, 20, 13, 8, 5, 4, 5, 8, 13, 20])
array('b', [34, 25, 18, 13, 10, 9, 10, 13, 18, 25])
array('b', [41, 32, 25, 20, 17, 16, 17, 20, 25, 32])
```

Arrays and Matrices

1. Sieve of Eratosthenes
   - counting primes
   - arrays in Python

2. Matrices
   - matrices as lists of arrays
   - finding the minimum
   - finding saddle points

finding the minimum in a matrix

input : a matrix `A`
output : `(i, j)`: `A[i][j]` is the minimum

Algorithm:

1. initialize current minimum with `A[0][0]`, and initialize its position to `(0, 0)`
2. go over all rows `i` and columns `j` of `A`
3. if `A[i][j]` is less than current minimum, assign `A[i][j]` to the current minimum, assign `(i, j)` to its position

the function findmin

```
def findmin(matrix):
    """
    Returns the row and the column indices
    of the minimum element in the matrix.
    """
    (row, col) = (0, 0)
    val = matrix[row][col]
    for i in range(0, len(matrix)):
        nbcols = matrix[i].buffer_info()[1]
        for j in range(0, nbcols):
            if matrix[i][j] < val:
                (row, col) = (i, j)
            val = matrix[row][col]
    return (row, col)
```

calling findmin()

```
def test():

    ...

    (i, j) = findmin(data)
    print('minimum value %d occurs at (%d, %d)' \
        % (data[i][j], i, j))

if __name__ == "__main__":
    test()
```

Running the Test

At the command prompt `$`:

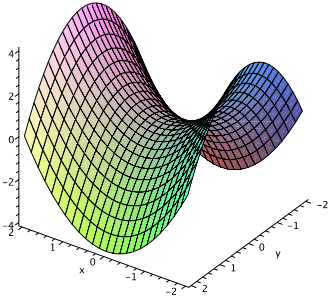
```
$ python findmin.py
looking for a minimum in
array('b', [50, 41, 34, 29, 26, 25, 26, 29, 34, 41])
array('b', [41, 32, 25, 20, 17, 16, 17, 20, 25, 32])
array('b', [34, 25, 18, 13, 10, 9, 10, 13, 18, 25])
array('b', [29, 20, 13, 8, 5, 4, 5, 8, 13, 20])
array('b', [26, 17, 10, 5, 2, 1, 2, 5, 10, 17])
array('b', [25, 16, 9, 4, 1, 0, 1, 4, 9, 16])
array('b', [26, 17, 10, 5, 2, 1, 2, 5, 10, 17])
array('b', [29, 20, 13, 8, 5, 4, 5, 8, 13, 20])
array('b', [34, 25, 18, 13, 10, 9, 10, 13, 18, 25])
array('b', [41, 32, 25, 20, 17, 16, 17, 20, 25, 32])
minimum value 0 occurs at (5, 5)
```

Saddles

The simplest mathematical description of a saddle is

$$z = x^2 - y^2$$

$(0, 0, 0)$ is a saddle point; plotted (with Maple) as



saddle points in a sampled surface

The test function is similar.

```
def test():
    """
    Testing the location of saddle points.
    """
    from array import array
    surface = lambda x, y: - (x-5)**2 + (y-5)**2
    data = [array('b', [surface(i, j) \
        for i in range(10)]) \
        for j in range(10)]
    print('looking for a saddle in ')
    for row in data:
        print(row)
```

Observe the minus sign in the definition of `surface`.

the test matrix

Edited output (to display the columns better):

```
looking for a saddle in
array('b', [  0,   9, 16, 21, 24, 25, 24, 21, 16,   9])
array('b', [ -9,   0,  7, 12, 15, 16, 15, 12,  7,   0])
array('b', [-16,  -7,  0,  5,  8,  9,  8,  5,  0,  -7])
array('b', [-21, -12, -5,  0,  3,  4,  3,  0, -5, -12])
array('b', [-24, -15, -8, -3,  0,  1,  0, -3, -8, -15])
array('b', [-25, -16, -9, -4, -1,  0, -1, -4, -9, -16])
array('b', [-24, -15, -8, -3,  0,  1,  0, -3, -8, -15])
array('b', [-21, -12, -5,  0,  3,  4,  3,  0, -5, -12])
array('b', [-16,  -7,  0,  5,  8,  9,  8,  5,  0,  -7])
array('b', [ -9,   0,  7, 12, 15, 16, 15, 12,  7,   0])
```

Observe the occurrence of the zero entries.

Arrays and Matrices

1. Sieve of Eratosthenes
   - counting primes
   - arrays in Python

2. Matrices
   - matrices as lists of arrays
   - finding the minimum
   - finding saddle points

finding saddle points

A saddle point in a matrix A is

1. maximal in its row; and
2. minimal in its column.

Problem (input/output specification):

input	:	a matrix `A`
output	:	list of `(i,j)` of saddles `A[i][j]`

Algorithm:

1. for all rows `i`, let `A[i][maxcol]` be maximal
2. check in column `maxcol` whether
   `A[i][maxcol] <= A[k][maxcol]`, for all `k != i`

tracing the execution of the algorithm

```
looking for a saddle in
array('b', [  0,   9, 16, 21, 24, 25, 24, 21, 16,   9])
array('b', [ -9,   0,  7, 12, 15, 16, 15, 12,  7,   0])
array('b', [-16,  -7,  0,  5,  8,  9,  8,  5,  0,  -7])
array('b', [-21, -12, -5,  0,  3,  4,  3,  0, -5, -12])
array('b', [-24, -15, -8, -3,  0,  1,  0, -3, -8, -15])
array('b', [-25, -16, -9, -4, -1,  0, -1, -4, -9, -16])
array('b', [-24, -15, -8, -3,  0,  1,  0, -3, -8, -15])
array('b', [-21, -12, -5,  0,  3,  4,  3,  0, -5, -12])
array('b', [-16,  -7,  0,  5,  8,  9,  8,  5,  0,  -7])
array('b', [ -9,   0,  7, 12, 15, 16, 15, 12,  7,   0])
max 25.0 in row 0, column 5 smaller value in row 1
max 16.0 in row 1, column 5 smaller value in row 2
max 9.0 in row 2, column 5 smaller value in row 3
max 4.0 in row 3, column 5 smaller value in row 4
max 1.0 in row 4, column 5 smaller value in row 5
max 0.0 in row 5, column 5 saddle at (5,5)
max 1.0 in row 6, column 5 smaller value in row 5
max 4.0 in row 7, column 5 smaller value in row 4
max 9.0 in row 8, column 5 smaller value in row 3
max 16.0 in row 9, column 5 smaller value in row 2
```

the function saddle()

We go over all rows, looking for the largest element.

```
def saddle(matrix):
    """
    Returns the coordinates of saddles:
    maximum in rows, minimum in columns.
    """
    result = []
    for i in range(0, len(matrix)):
        (maxval, maxcol) = (matrix[i][0], 0)
        nbcols = matrix[i].buffer_info()[1]
        for j in range(1, nbcols):
            if matrix[i][j] > maxval:
                (maxval, maxcol) = (matrix[i][j], j)
        prt = 'max %.1f in row %d, column %d' \
            % (maxval, i, maxcol)
```

checking whether minimal in its column

We have a candidate saddle point in `matrix[i][maxcol]`.
Now we check whether it is minimal in its column, in the `i`-loop:

```
    is_saddle = True
    for k in range(0, len(matrix)):
        if(k != i):
            if matrix[k][maxcol] < maxval:
                prt = prt + ' smaller value in row %d' % k
                is_saddle = False
                break
    if is_saddle:
        prt = prt + ' saddle at (%d,%d)' % (i, maxcol)
        result.append((i, maxcol))
    print prt
return result
```

Summary

For working with numerical data, arrays are more memory efficient.

Searching in two dimensional tables is a common problem:

- finding extremal values;
- locating saddle points.

Such searching algorithms apply the same double loop:
for all rows, for all columns, do something.

Later we will scan all words from a page of text on file:
for all lines on the page, read all words on the line.

Exercises

1. Give a Python function `swap` to reverse the order of the elements in an array. After calling `swap` on `A = [2 9 8 3]`, we have `A = [3 8 9 2]`. Do not create a new array inside `swap`.
2. Write a Python function `Duplicates` whose input is an array `A` and output is a list of elements that occur at least twice in `A`.
3. A plateau in an array is the longest sequence of the same elements that occur in the array. Write a Python function that returns the start and begin index of a plateau in a given array.

More exercises

4. Extend the function `saddle` with a parameter `verbose` which takes the default value `True`. If `verbose`, then the strings `prt` are shown, otherwise the function `saddle` does not print anything.
5. For a two dimensional matrix of integer values, write a formatted print function: every entry is printed with the formatting string `'%3d'`.
6. Extend the `findmin` to find the smallest value in a 3-dimensional matrix. Think of a elements in the matrix as temperature measurements.