MAT 305: Mathematical Computing

Lecture 7: Decision-making in Sage

John Perry

University of Southern Mississippi

Fall 2009

Outline

You should be in worksheet mode to repeat the examples.

Outline

1. Decision-making
2. Boolean statements
3. Having said all that…
4. Summary

Decision making?

A function may have to act in different ways, depending on the arguments.

Decision making?

A function may have to act in different ways, depending on the arguments.

Example

Piecewise functions:

$$f(x) = \begin{cases} f_1(x), & x \in (a_0, a_1) \\ f_2(x), & x \in [a_1, a_2) \end{cases}.$$

Decision making?

A function may have to act in different ways, depending on the arguments.

Example

Characterizing concavity:

$$\begin{gathered}\text{If } f''(a) > 0, \text{ then } f \text{ is concave up at } x = a;\\ \text{if } f''(a) < 0, \text{ then } f \text{ is concave down at } x = a;\\ \text{if } f''(a) = 0, \text{ then } a \text{ is an inflection point of } f.\end{gathered}$$

`if` statements

`if` (*condition*) `:`
    if-statement1
    if-statement2
    …
non-if statement1

where

- *condition*: expression that evaluates to `True` or `False`
- *condition* `True`? *statement1*, *statement2*, etc. performed
  - control passes *finally* to *non-if statement1*
- *condition* `False`? *statement1*, *statement2*, … skipped
  - control passes *immediately* to *non-if statement1*

Example

```
sage:  f = cos(x)
sage:  ddf = diff(f,2)
sage:  if (ddf(3*pi/4) > 0):
         print 'concave up at', 3*pi/4
concave up at 3/4*pi
```

`if-else` statements

`if` (*condition*)`:`
    if-statement1
    `...`
`else:`
    else-statement1
    `...`
non-if statement1

where

- *condition* `True`? *statement1*, … performed
    - *else-statement1*, … skipped
- condition `False`? *else-statement1*, … performed
    - *statement1*, … skipped
- control passes finally to *non-if statement1*

`if-elif-else` statements

```
if (condition1):
    if-statement1
    ...
elif (condition2):
    elif1-statement1
    ...
elif (condition3):
    elif2-statement1
    ...
...
else:
    else-statement1
    ...
non-if statement1
```

where

- statement block selected by condition

Pseudocode for `if-elif-else`

if *condition1*
    if-statement1
    …
else if *condition2*
    elseif1-statement1
    …
else if *condition3*
    elseif2-statement1
    …
…
else *condition2*
    else-statement1
    …

Notice:

- indentation
- no parentheses, colons
- **else if**, not **elif**

Example: concavity

Write a Sage function that tests whether a function f is concave up or down at a given point. Have it return the string 'concave up', 'concave down', or 'neither'.

Example: concavity

Write a Sage function that tests whether a function f is concave up or down at a given point. Have it return the string 'concave up', 'concave down', or 'neither'.

Different choices $\Longrightarrow$ need to make a decision! $\Longrightarrow$ **if**

Example: concavity

Write a Sage function that tests whether a function f is concave up or down at a given point. Have it return the string 'concave up', 'concave down', or 'neither'.

Different choices $\Longrightarrow$ need to make a decision! $\Longrightarrow$ **if**

Start with pseudocode.

- What inputs are needed?
- What output is expected?
- What has to be done?
    - step by step
    - *Divide et impera!* **Divide and conquer!**

Pseudocode for Example

algorithm *check_concavity*
inputs

Pseudocode for Example

algorithm *check_concavity*
inputs
  $a \in \mathbb{R}$
  $f(x)$, a twice-differentiable function at $x = a$
outputs

Pseudocode for Example

algorithm *check_concavity*
inputs
  $a \in \mathbb{R}$
  $f(x)$, a twice-differentiable function at $x = a$
outputs
  'concave up' if f is concave up at $x = a$
  'concave down' if f is concave down at $x = a$
  'neither' otherwise
do

Pseudocode for Example

algorithm *check_concavity*
inputs
  $a \in \mathbb{R}$
  $f(x)$, a twice-differentiable function at $x = a$
outputs
  'concave up' if f is concave up at $x = a$
  'concave down' if f is concave down at $x = a$
  'neither' otherwise
do
  if $f''(a) > 0$
    return 'concave up'
  else if $f''(a) < 0$
    return 'concave down'
  else
    return 'neither'

Try it!

```
sage:  def check_concavity(a, f, x):
         ddf = diff(f, x, 2)
         if (ddf(x=a) > 0):
           return 'concave up'
         elif (ddf(x=a) < 0):
           return 'concave down'
         else:
           return 'neither'
```

Try it!

```
sage:  def check_concavity(a, f, x):
         ddf = diff(f, x, 2)
         if (ddf(x=a) > 0):
           return 'concave up'
         elif (ddf(x=a) < 0):
           return 'concave down'
         else:
           return 'neither'
sage:  check_concavity(3*pi/4, cos(x), x)
'concave up'
sage:  check_concavity(pi/4, cos(x), x)
'concave down'
```

Example: piecewise function

Write a function whose input is any $x \in \mathbb{R}$ and whose output is

$$f(x) = \begin{cases} 1 - x^2, & x < 0 \\ 0, & x = 0 \\ x^2 - 1, & x > 0. \end{cases}$$

Example: piecewise function

Write a function whose input is any $x \in \mathbb{R}$ and whose output is

$$f(x) = \begin{cases} 1 - x^2, & x < 0 \\ 0, & x = 0 \\ x^2 - 1, & x > 0. \end{cases}$$

Three different choices $\Longrightarrow$ need to make a decision! $\Longrightarrow$ **if**

Pseudocode for example

algorithm *piecewise_f*
inputs

Pseudocode for example

algorithm *piecewise_f*
inputs
    $a \in \mathbb{R}$
outputs

Pseudocode for example

algorithm *piecewise_f*
inputs
    $a \in \mathbb{R}$
outputs
    $f(a)$, where f is defined as above
do

Pseudocode for example

algorithm *piecewise_f*
inputs
    $a \in \mathbb{R}$
outputs
    $f(a)$, where f is defined as above
do
    if $a < 0$
        return $1 - a^2$

Pseudocode for example

algorithm *piecewise_f*
inputs
  $a \in \mathbb{R}$
outputs
  $f(a)$, where f is defined as above
do
  if $a < 0$
    return $1 - a^2$
  else if $a = 0$
    return 0

Pseudocode for example

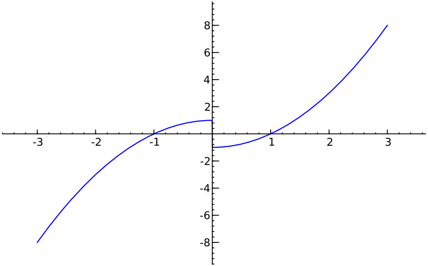
algorithm *piecewise_f*
inputs
   $a \in \mathbb{R}$
outputs
   $f(a)$, where f is defined as above
do
   if $a < 0$
      return $1 - a^2$
   else if $a = 0$
      return 0
   else
      return $a^2 - 1$

Python code

```
sage: def piecewise_f(a):
        if (a < 0):
          return 1 - a**2
        elif (a == 0):
          return 0
        else:
          return a**2 - 1
sage: piecewise_f(3)
8
```

It gets better

```
sage:  plot(piecewise_f, xmin=-3, xmax=3)
```



It gets worse, too

How do we handle a piecewise function defined over more complicated intervals?

Example

Suppose

$$g(x) = \begin{cases} 3x, & x \in [0,2) \\ -\frac{x}{3} + \frac{20}{3}, & x \in [2,20) \\ 0, & x \geq 20. \end{cases}$$

How do we define this in Sage?

Pseudocode deceptively easy

algorithm *piecewise_g*
inputs
  $a \in [0, \infty)$
outputs
  $g(a)$, where g is defined as above
do
  if $a \in [0, 2)$
    return $3a$
  else if $a \in [2, 20)$
    return $-\frac{a}{3} + \frac{20}{3}$
  else
    return 0

Pseudocode deceptively easy

algorithm *piecewise_g*
inputs
  $a \in [0, \infty)$
outputs
  $g(a)$, where g is defined as above
do
  if $a \in [0, 2)$
    return $3a$
  else if $a \in [2, 20)$
    return $-\frac{a}{3} + \frac{20}{3}$
  else
    return 0

…but how does does Sage decide $a \in [x_1, x_2)$?!?

Sage code: careful use of `if-elif`?

```
def piecewise_g(a):
  if a ≥ 0
    if a ≥ 2
      if a ≥ 20
        return 0
      else
        return -a/3 + 20/3
    else
      return 3a
```

How do we even check this?

Sage code: careful use of `if-elif`?

```
def piecewise_g(a):
  if a ≥ 0
    if a ≥ 2
      if a ≥ 20
        return 0
      else
        return -a/3 + 20/3
    else
      return 3a
```

How do we even check this?

Works, but not pleasant to work out (or easy to read).

Outline

Boolean algebra

Boolean algebra operates on only two values: {`True`, `False`}.

…or {1, `0`} if you prefer

…or {Yes, No} if you prefer

Boolean algebra

Boolean algebra operates on only two values: {`True`, `False`}.

…or {1, `0`} if you prefer

…or {Yes, No} if you prefer

Basic operations:

- **not** x
  - `True` iff x is `False`
- x **and** y
  - `True` iff both x and y are `True`
- x **or** y
  - `True` iff
    - x is `True`; or
    - y is `True`; or
    - both x and y are `True`

Example: and, or

```
sage: 5 > 4
True
```
obvious enough

```
sage: 5 < 4
False
sage: (5 > 4) or (5 < 4)
True
```
because at least one is `True` $(5 > 4)$

```
sage: (5 > 4) and (5 < 4)
False
```
because neither is `True`

Example: not

```
sage: 4 > 4
False
```
obvious enough

```
sage: not (4 > 4)
True
sage: not ((5 > 4) or (4 < 5))
False
```
we have (not True)

```
sage: not (4 == 5)
True
```
we have (not False)

Equality and inequalities

Recall: = and == are not the same

- `x = y` assigns value of y to `x`
- `x == y` compares values of `x`, `y`, reports `True` or `False`

Equality and inequalities

Recall: `=` and `==` are not the same

- `x = y` assigns value of `y` to `x`
- `x == y` compares values of `x`, `y`, reports `True` or `False`

For inequalities,

- `x != y` compares values of `x`, `y`
    - `True` iff `not (x == y)`
- `x > y`, `x < y` have usual meanings

Equality and inequalities

Recall: = and == are not the same

- `x = y` assigns value of y to x
- `x == y` compares values of x, y, reports `True` or `False`

For inequalities,

- `x != y` compares values of x, y
  - `True` iff `not (x == y)`
- `x > y`, `x < y` have usual meanings
- $x \geq y$? use `x >= y`
  - `True` iff `not (x < y)`
- $x \leq y$? use `x <= y`
  - `True` iff `not (x > y)`

Back to the example

Example

Suppose

$$g(x)=\begin{cases}3x, & x\in[0,2)\\ -\frac{x}{3}+\frac{20}{3}, & x\in[2,20)\\ 0, & x\geq 20.\end{cases}$$

How do we define this in Sage? Using Boolean algebra, the pseudocode (and Python code) becomes much simpler.

Pseudocode, again

algorithm *piecewise_g*
inputs
  $a \in [0, \infty)$
outputs
  $g(a)$, where g is defined as above
do
  if $a \in [0, 2)$
    return $3a$
  else if $a \in [2, 20)$
    return $-\frac{a}{3} + \frac{20}{3}$
  else
    return 0

Pseudocode, again

algorithm *piecewise_g*
inputs
  $a \in [0, \infty)$
outputs
  $g(a)$, where g is defined as above
do
  if $a \in [0, 2)$
    return $3a$
  else if $a \in [2, 20)$
    return $-\frac{a}{3} + \frac{20}{3}$
  else
    return 0

…but how does does Sage decide $a \in [x_1, x_2)$?!?

use $a \geq x_1$ **and** $a < x_2$!

Sage code

```
sage:  def piecewise_g(a):
         if ((a >= 0) and (a < 2)):
           return 3*a
         elif ((a >= 2) and (a < 20)):
           return -a/3 + 20/3
         else:
           return 0
```

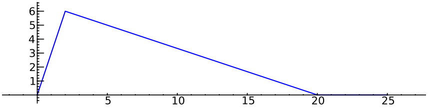
Sage code

```
sage: def piecewise_g(a):
        if ((a >= 0) and (a < 2)):
          return 3*a
        elif ((a >= 2) and (a < 20)):
          return -a/3 + 20/3
        else:
          return 0
```

Much easier to look at.

Voilà!

```
sage: def piecewise_g(a): ...
sage: pgplot = plot(piecewise_g, 0, 25)
sage: show(pgplot, aspect_ratio=1)
```



Outline

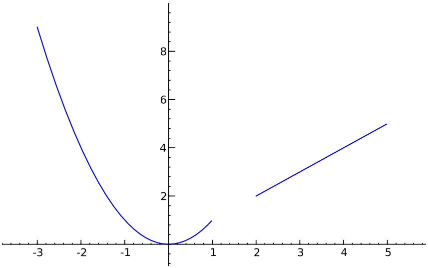
Sage has a `piecewise()` command…

`piecewise([[(`a_1`, `b_1`), `f_1`], [(`a_2`, `b_2`), `f_2`], ...])` where

- $a_i, b_i \in \mathbb{R}$
- f_i describes the behavior of the function on the interval (a_i, b_i)

…so it's actually a little easier

```
sage: piecewise_g = piecewise([[(-3,1), x**2],
                               [(2,5), x]])
sage: plot(piecewise_g, xmin=-3, xmax=3)
```



Outline

Summary

- Decision making accomplished via `if-elif-else`
  - pseudocode: **if**, **else if**, **else**
- Mathematical examples abound!
  - testing properties of functions
  - piecewise functions
- Boolean algebra helps create conditions for `if` and `elif`
  - **and**, **or**, **not**
  - `<=`, `!=`, `>=`