Intro to AMPL Scripting

ER220
Fall 2006

Richard Plevin
plevin@berkeley.edu
If you plan to use AMPL beyond homework-size problems, you’ll need this book.

For more information:

• Chapter 13: Command Scripts
• Appendix A: Reference Manual
Road Map

1. Fundaments of scripting
2. Defining and setting variables
3. Conditional expressions
4. Looping
5. Displaying data and results
6. Extended example
7. Advice from the trenches
Fundaments of Scripting

• Even a trivial script helps avoid typing:

```ampl
model steel.mod;
data steel.dat;
solve;
display Make;
```

• Always create a “run” script, if not several. All non-trivial models will require one.

• Run it by passing filename to ampl program:

```
$ ampl steel.run
```
Common uses of scripts

- Parametric analysis
  - Run the model using a series of parameter values, saving the results for later analysis.

- Time-stepping
  - Run a myopic model over a series of time-steps, accumulating results between runs.

- Monte Carlo simulation
  - Use built-in random number functions to simulate variance of uncertain parameters.
Declaring Parameters

# Basic declaration; no value assigned
param hours;

# Declaration with default value
param daysPerMonth{1..12} default 31;

# Declaration with assignment
param months := 12;

The first two declarations permit updating the parameter with “let”. The third does not.
Assigning values with “let”

# Assignment to a scalar
let hours := 40;

# Assignment to an array element
let daysPerMonth[2] := 28;
let avail_variance[‘reheat’] := 3;

# Assignment to an array using iteration
let {month in {4,6,9,11}}
    daysPerMonth[month] := 30;
Conditional Expressions

• **If-then-else** can control program flow:

```javascript
if month == 2 then
  let daysPerMonth[month] := 28;
else
  let daysPerMonth[month] := 31;
```

• **Blocks of code must be enclosed with braces:**

```javascript
if month == 2 then {
  let daysPerMonth[month] := 2;
  let name[month] := “Feb”;
}
```
Conditional Expressions

- **If-then-else** can also compute a value:

  ```
  let daysPerMonth[month] :=
      if month == 2 then 28 else 31;
  ```

- **If-then-else** expressions can be nested:

  ```
  let {m in 1..12} daysPerMonth[m] :=
      if m in {4,6,9,11} then 30
      else (if m == 2 then 28 else 31);
  ```
Looping

- **for loop** iterates across sets

  ```
  for {i in 1..N} {
      solve;
      let profit[i] := Total_Profit;
  }
  ```

- **Iterate over multiple sets in one statement:**

  ```
  for {f in FUELS, r in REGIONS} {
      let Upstream_CO2 := Upstream_CO2 +
          upstream_co2[f] * production[f, r];
  }
  ```
Escaping a loop

- **Exit a for loop early using break**
  ```plaintext
  for {i in 1..N} {
    solve;
    if Total_Profit >= minimum then
      break;
  }
  ```

- **Skip part the rest of a loop with continue**
  ```plaintext
  for {i in 1..N} {
    if factor[i] == 0 then continue;
    let x := y/factor[i];
  }
  ```
Displaying data and results

• `display` formats data automatically
• Several options exist to control formatting
• These affect all subsequent `display` commands

• `printf` uses the format specified per call
• Both `display` and `printf` can write to files using “output redirection”. 
The printf Command

• `printf format, arguments...`

  ```c
  printf "The month %d has %d days\n",
    m, daysPerMonth[m];
  ```

• The format string uses placeholders that begin with the percent sign, e.g. `%d, %f, %e, etcetera.

• Each corresponds to an argument provided after the format string

• Values are substituted for placeholders to compose the text to display

• Use `\n` for newline; `\t` for tab; `%%` for a literal percent sign.
printf formatting examples

(param x default 6123212312.4239;

printf "x: %f\n", x;       # default format
printf "x: %.2f\n", x;      # 2 digits after dec. point
printf "x: %.2e\n", x;      # exponential, 2 sig figs
printf "x: %.9.2e\n", x;    # 2 sig figs, 9 chars wide
printf "x: %.4e\n", x;      # exponential, 4 sig figs

--------------------------------------------

x: 6123212312.423900
x: 6123212312.42
x: 6.1e+09
x: 6.1e+09
x: 6.123e+09)
printf formatting examples

```
param y default 14323;

printf "y: %d\n", y;     # default format
printf "y: %-10d.\n", y;  # left justify, 10 wide
printf "y: %10d\n", y;    # right justify, 10 wide
printf "y: %010d\n", y;   # zero-padded to 10 wide

```

--------------------------------------------

```
y: 14323
y: 14323    .
y:      14323
y: 0000014323
```

Writing to files

• The “>” symbol causes the printf and display commands to be redirected to the file indicated.

    display Total_Profit > myfile.txt;

• The first time a file is used, it is opened

• Subsequent “>” commands append to the file

• To write to a filename stored in a variable it declare “symbolic” and use parenthesis:

    param filename symbolic;
    let filename := ‘Results.txt’;
    display Total_profit > (filename);
Writing tab-delimited files

let file := "globalUpstreamCarbon.dat";

# Print header row
printf "year\ttoil\tsynfuels\n" > (file);

for {y in YEARS} {
    printf "%d\t%f\t%f\n", y,
    sum{r in REGIONS}
        OilProduction[r,y] * 
        ProductionEmissions['oil'],
    sum{r in REGIONS, s in SYNFUELS}
        AnnualSynfuelProduction[r, s, y] * 
        ProductionEmissions[s]
    > (file);
}

Writing tab-delimited files

<table>
<thead>
<tr>
<th>year</th>
<th>oil</th>
<th>synfuels</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>0.836765</td>
<td>0.000000</td>
</tr>
<tr>
<td>2001</td>
<td>0.797215</td>
<td>0.108034</td>
</tr>
<tr>
<td>2002</td>
<td>0.808856</td>
<td>0.140848</td>
</tr>
<tr>
<td>2003</td>
<td>0.849277</td>
<td>0.110047</td>
</tr>
<tr>
<td>2004</td>
<td>0.892283</td>
<td>0.077152</td>
</tr>
<tr>
<td>2005</td>
<td>0.937616</td>
<td>0.053260</td>
</tr>
<tr>
<td>2006</td>
<td>0.983835</td>
<td>0.027231</td>
</tr>
<tr>
<td>2007</td>
<td>1.028771</td>
<td>0.001824</td>
</tr>
<tr>
<td>2008</td>
<td>1.060364</td>
<td>0.000720</td>
</tr>
<tr>
<td>2009</td>
<td>1.073055</td>
<td>0.036965</td>
</tr>
<tr>
<td>2010</td>
<td>1.090488</td>
<td>0.078196</td>
</tr>
<tr>
<td>2011</td>
<td>1.127128</td>
<td>0.076165</td>
</tr>
<tr>
<td>2012</td>
<td>1.162157</td>
<td>0.073627</td>
</tr>
</tbody>
</table>

...
Extended Example

- See steel model...
Advice from the trenches

Most software development is code maintenance. Try to make this as easy as possible.

- Choose names carefully
- Use space generously
- Comment non-obvious code
- Take baby steps
- Test on small data sets