PROGRAMMING LANGUAGES
FOR THE NUMERICAL ANALYSIS CLASSROOM

ED BUELER

MATLAB ("matrix laboratory"; see `mathworks.com`) was created in the late 1970s by Cleve Moler for teaching numerical linear algebra without requiring FORTRAN programming. It has since become a powerful programming language and engineering tool. A large fraction of upper-division and graduate students at UAF are already familiar with it, and it is available through UAF. The "Matlab student" version at `www.mathworks.com/academia/ student_version.html`, works fine for the numerical mathematics classes I teach. It works well, looks good, and I like it.

MATLAB is recommended if you have no existing preference, but I prefer free and open source software. Among the alternatives are three which work very well for numerical courses:

1. OCTAVE is a MATLAB clone. Download it at `www.gnu.org/software/octave`. The "`.m`" examples on the next page, and thoughout this course, work in an identical way in MATLAB and in OCTAVE. I will mostly use OCTAVE myself during the course, but I'll also make sure examples work the same way in MATLAB.
2. The general-purpose language PYTHON (`python.org`) works very well if you use the SCIPY (`scipy.org`) and MATPLOTLIB (`matplotlib.org`) libraries. Then it has all of MATLAB functionality and more. Using it with the IPYTHON interactive shell (`ipython.org`) gives the most MATLAB-like experience.
3. The Julia language (`julialang.org`) is a modern redesign of MATLAB, but it is not a compatible clone like OCTAVE. It easy to learn. Equivalent codes run much faster than in MATLAB or OCTAVE.

On the next page are two algorithms each in MATLAB/OCTAVE (left column) and PYTHON (right column); see me for the corresponding Julia examples. To download these examples, go to

`bueler.github.io/M614F21`

and look in the left column. Next are some brief "how-to" comments.

Program `gaussint.m` is a *script*. A script is run by starting MATLAB/OCTAVE, usually in the directory containing the script you want to run. Then type the name of the script at the prompt, without the `.m`:

```
>> gaussint
```

Typing `help gaussint` at the MATLAB/OCTAVE prompt shows the first block of comments.

`bis.m` is a *function* which needs inputs. At the prompt enter, for example,

```
>> f = @(x) cos(x) - x
>> bis(0,1,f)
```

Note we have given `bis.m` three arguments; the last is a function.

For the PYTHON codes: You can do `python gaussint.py` directly from a shell. Alternatively, from the PYTHON or IPYTHON prompt, type `run gaussint`. For the function `bis.py`, first do: `from bis import bis`. Then run the example as shown in the docstring; in IPYTHON you can type `bis?` to print the docstring.

Date: July 30, 2021.

gaussint.m

```
% plot the integrand and approximate
% the integral
%   / 1
%   |    exp(-x^2/pi) dx
%   / 0
% by left-hand, right-hand, and
% trapezoid rules

N = 1000;
dx = (1 - 0) / N;
x = linspace(0,1,N+1);
y = exp(- x.^2 / pi);

plot(x,y)
axis([0 1 0 1]), grid

format long
lhand = dx * sum(y(1:end-1))
rhand = dx * sum(y(2:end))
trap  = (dx/2) * sum(y(1:end-1)+y(2:end))
exact = (pi/2) * erf(1/sqrt(pi))
```

gaussint.py

```
'''plot the integrand and approximate
the integral
    / 1
    |    exp(-x^2/pi) dx
    / 0
by left-hand, right-hand, and
trapezoid rules'''

import numpy as np
import matplotlib.pyplot as plt
from scipy.special import erf

N = 1000
dx = (1.0 - 0.0) / N
x = np.linspace(0.0,1.0,N+1)
y = np.exp(- x**2 / np.pi)

plt.plot(x,y)
plt.axis([0.0,1.0,0.0,1.0])
plt.grid(True)

lhand = dx * sum(y[:-1])
print("lhand = %.15f" % lhand)
rhand = dx * sum(y[1:])
print("rhand = %.15f" % rhand)
trap  = (dx/2) * sum(y[:-1]+y[1:])
print("trap = %.15f" % trap)
exact = (np.pi/2) * erf(1/np.sqrt(np.pi))
print("exact = %.15f" % exact)
plt.show()
```

bis.m

```
function c = bis(a,b,f)
% BIS  Apply the bisection method to solve
%   f(x) = 0
% with initial bracket [a,b].  Example:
%   >> f = @(x) cos(x) - x      % define fcn
%   >> r = bis(0,1,f)           % find root
%   >> f(r)                     % confirm

if (feval(f,a)) * (feval(f,b)) > 0
  error('not a bracket!'), end
for k = 1:100
  c = (a+b)/2;
  r = feval(f,c);
  if abs(r) < 1e-12
    return  % we are done
  elseif feval(f,a) * r >= 0.0
    a = c;
  else
    b = c;
  end
end
error('no convergence')
```

bis.py

```
def bis(a,b,f):
    '''BIS  Apply the bisection method to solve
        f(x) = 0
    with initial bracket [a,b].  Example:

    from bis import bis
    def f(x):
        from math import cos
        return cos(x) - x
    r = bis(0.0,1.0,f)
    print([r,f(r)])'''

    if f(a) * f(b) > 0.0:
        print("not a bracket!")
        return
    for k in range(100):
        c = (a+b)/2
        r = f(c)
        if abs(r) < 1e-12:
            return c # we are done
        elif f(a) * r >= 0.0:
            a = c
        else:
            b = c
    print("no convergence")
    return
```