

---

---

MATHEMATICAL PHYSICS DEPARTMENT  
MASTER OF COMPUTATIONAL SCIENCE DEGREE 2006-2007  
NUMERICAL ALGORITHMS

*Dr Derek O'Connor*

---

---

**Lab Exercise No. 1 : *Calculating Constants.***

OUT: Wed 13 Sep 2006

IN : Wed 20 Sep 2006

---

## 1 PURPOSE

The purpose of this exercise is to

- Gain some experience with the MATLAB system.
- Write simple but non-trivial MATLAB functions that use loops.
- Examine experimentally the convergence of sequences.

## 2 BACKGROUND

Euler's famous equation

$$e^{i\pi} + 1 = 0$$

contains five of the most important constants in mathematics. Three of them need no calculation. The other two require calculation and the calculation of one of them, today, consumes vast amounts of super-computer power.

### 2.1 Archimedes's Constant $\pi$ .

The ratio of the girth (circumference) to the breadth (diameter) of a circle is a constant and is denoted by  $\pi$ . Lumberjacks (or timbermen in Wicklow) know this implicitly — 'it's about trey t' wan, even at the top where it goes narra'.<sup>1</sup>

The calculation of the transcendental number  $\pi$  has a history that stretches back over 2000 years and is still a 'hot' computational task today.

Our latest record which was announced already at press release time of 6-th of December, 2002 was as the followings;

Declared record:

1,030,700,000,000 hexadecimal digits 1,241,100,000,000 decimal digits

Two independent hexadecimal calculation based on two different algorithms generated more than 1,030,775,430,000 hexadecimal digits of pi and comparison of two generated sequences matched completely. Computed hexadecimal digits of pi were radix converted into base 10, generating more than 1,241,177,300,000 decimal

---

<sup>1</sup>Timbermen (esp. in Wicklow) have long been discussing the philosophical question 'is  $\pi$  a logical or a physical constant?'. What do you think?

digits of pi and generated decimal digits of pi were radix converted again into base 16. Radix converted hexadecimal digits of pi were compared with original hexadecimal digits of pi. There were no difference up to 1,241,100,000,000 decimal digits. Then we are declaring 1,030,700,000,000 hexadecimal digits and 1,241,100,000,000 decimal digits as the new world records. Details of computed results are available on the following URLs.

[http://www.super-computing.org/pi-hexa\\_current.html](http://www.super-computing.org/pi-hexa_current.html) (hexadecimal)

[http://www.super-computing.org/pi-decimal\\_current.html](http://www.super-computing.org/pi-decimal_current.html) (decimal)

Summary of computation:

(1) 64 nodes of HITACHI SR8000/MPP (144 nodes, 14.4GFlops/node, 16GB/node, node to node data transmission speed: 1.6GB/sec one-way and 3.2GB/sec both- way) at Information Technology Center, University of Tokyo were used.

## 2.2 Euler's Number $e$

The exponential function  $e^x$  has the curious property that  $\frac{d e^x}{d x} = e^x$ . Using this property we can derive the power series expansion

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} \dots, \tag{1}$$

from which we get

$$e^1 = 1 + \frac{1}{1!} + \frac{1^2}{2!} + \frac{1^3}{3!} \dots = 2 + \frac{1}{2!} + \frac{1}{3!} \dots \tag{2}$$

Alternatively, applying the Binomial theorem to the definition  $e = \lim_{n \rightarrow \infty} (1 + \frac{1}{n})^n$  we get

$$\begin{aligned} e = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n &= \lim_{n \rightarrow \infty} \left(1 + n \left(\frac{1}{n}\right) + \frac{n(n-1)}{2!} \left(\frac{1}{n}\right)^2 + \frac{n(n-1)(n-2)}{3!} \left(\frac{1}{n}\right)^3 + \dots + \left(\frac{1}{n}\right)^n\right) \\ &= \lim_{n \rightarrow \infty} \left(1 + 1 + \frac{(1-\frac{1}{n})}{2!} + \frac{(1-\frac{1}{n})(1-\frac{2}{n})}{3!} + \dots + \frac{1}{n^n}\right) \\ &= 2 + \frac{1}{2!} + \frac{1}{3!} \dots \end{aligned}$$

## 2.3 Euler's Constant $\gamma$

The Harmonic Series or Number  $H_n$  is defined as

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} = \sum_{k=1}^n \frac{1}{k}, \text{ and it diverges to } +\infty. \tag{3}$$

This was first proved by Nicole d'Oresme, Bishop of Lisieux, (c. 1323–1382).

It is easy to show that

$$\ln n + \frac{1}{n} \leq H_n \leq \ln n + 1.$$

We can see that the difference  $(H_n - \ln n)$  lies in the interval  $[1/n, 1]$ , which goes to  $[0, 1]$  in the limit.

Euler assumed the sequence  $H_n - \ln n$  converged and defined the limit

$$\gamma = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} - \ln n\right) = \lim_{n \rightarrow \infty} (H_n - \ln n) = 0.5772156649015328606\dots \tag{4}$$

This shows that  $H_n$  lies a little over half-way between  $\ln n$  and  $\ln n + 1$ . The constant  $\gamma$  is called *Euler's Constant*.

This constant  $\gamma$  is something of a mystery because it is not known whether it is rational, irrational, or transcendental. Indeed, if it is rational, i.e., if there are integers  $p$  and  $q$  such that  $\gamma = p/q$ , then it has been shown that, of necessity,  $q > 10^{242,080}$ , i.e., an integer with at least 242,080 digits.

### 3 EXERCISE

Write and test four small MATLAB functions to

1. Calculate  $\pi$  using  $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$
2. Calculate  $\pi$  using  $\frac{\pi}{2} = \frac{2 \cdot 2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdot 8 \cdot 8 \dots}{1 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdot 7 \cdot 9 \dots}$
3. Calculate Euler's number using  $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} \dots$
4. Calculate Euler's constant using  $\gamma = (1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} - \ln n)$

Write the MATLAB functions with the header function value=Pi1(n), [Pi2(n), EulerN(n), EulerC(n)], where  $n$  is the number of terms in the series. Test each function for  $n = 10^1, 10^3, 10^6, 10^9$ . Compare your results with MATLAB's pi and exp(1), and PariGP's Euler = 0.5772156649015329. These values are correct to 15 or 16 decimal digits.

### 4 SOLUTION

#### 4.1 The MATLAB Functions for $\pi$ , $e$ , and $\gamma$

Here are the two functions for calculating  $\pi$ .

function y = Pi1(n);	function y = Pi2(n);
sum = 0; sgn = 1;	prod = 1;
for k = 1:2:n;	for k = 2:2:floor(n/2)
sum = sum + sgn/k;	T = k^2/(k^2-1);
sgn = -sgn;	prod = prod*T;
end;	end;
y = 4*sum;	y = 2*prod;

The function Pi1(n) is easy to implement. The **for** -loop gives  $k$  the values 1,3,5,...,  $n-1$  or  $n$ , and the statement  $sgn = -sgn$  gives the alternating sign.

The function Pi2(n) is not so easy to implement. It is important to realize that the terms in the numerator and the denominator must be grouped in pairs, i.e.,

$$\frac{\pi}{2} = \left(\frac{2 \cdot 2}{1 \cdot 3}\right) \left(\frac{4 \cdot 4}{3 \cdot 5}\right) \left(\frac{6 \cdot 6}{5 \cdot 7}\right) \dots \left(\frac{k \cdot k}{(k-1) \cdot (k+1)}\right) \dots = T_1 \cdot T_2 \cdot T_3 \dots T_k \dots \tag{5}$$

The general term is  $T_k = \frac{k \cdot k}{(k-1) \cdot (k+1)} = \frac{k^2}{k^2-1}$  and the **for** -loop forms the product of these for  $k = 2, 4, 6, \dots, \lfloor n/2 \rfloor$ .

Naïve programmers might be tempted to implement this function by calculating the numerator and the denominator separately :

$$N_n = \prod_{k=2}^n k^2 \quad \text{and} \quad D_n = \prod_{k=2}^n (k^2 - 1).$$

This will cause overflow because both  $N_n$  and  $D_n$  are huge for even modest values of  $n$ . Notice that  $\lim_{k \rightarrow \infty} T_k = 1$ , and so the product (5) converges, and no overflow occurs.

Here are the two functions for calculating  $e$  and  $\gamma$ .

function y = EulerN(n);		function y = EulerC(n);
sum = 1; T = 1;		sum = 0;
for k = 1:n		for k = 1:n
T = T/k;		sum = sum + 1/k;
sum = sum + T;		end;
end;		y = sum - log(n);
y = sum;		

These two functions are very easy to implement but again, there is a trap waiting for the naïve programmer: if EulerN is implemented as  $\text{sum} = \text{sum} + 1/\text{factorial}(k)$  then  $\text{factorial}(k)$  will overflow for  $k = 171$  because  $171! \approx 1.241018070 \times 10^{309}$ .

## 4.2 Testing the Functions

This exercise requires testing the four functions with four values of  $n$  and comparing the values obtained with the 'correct' values given by MATLAB and the given value for Euler's constant. This means that for each value of  $n$  we must calculate each function and the error in each function. Thus there will be 8 values for each  $n$ , or 32 values in total.

The simplest way to do this is to type in the command window something such as  $y = \text{Pi1}(n)$  and  $\text{errPi1} = y - \text{pi}$  for each function and each  $n$ . This is a tedious and error-prone task. Once you are satisfied that the functions are working properly then the systematic testing should be done by a function written to do all tests and summarize the results. In other words, get the computer to do all the mundane repetitive work. Here is such a function.

```

%=====
function result = TestPiEtcLE1;
% -----
% Tests the functions Pi1,Pi2,EulerN, and EulerC
% Derek O'Connor, UCD, Sept 2006
%
disp('Press RETURN to start tests ...');
pause;
%
eulerG = 0.577215664901533;
n = [10^1; 10^3; 10^6; 10^9];
nvals = length(n);
p1 = zeros(nvals,1); errp1 = zeros(nvals,1);
p2 = zeros(nvals,1); errp2 = zeros(nvals,1);
en = zeros(nvals,1); erren = zeros(nvals,1);
ec = zeros(nvals,1); errec = zeros(nvals,1);
%
for k = 1:nvals
    p1(k) = Pi1(n(k));    errp1(k) = abs(p1(k) - pi);
    p2(k) = Pi2(n(k));    errp2(k) = abs(p2(k) - pi);
    en(k) = EulerN(n(k)); erren(k) = abs(en(k) - exp(1));
    ec(k) = EulerC(n(k)); errec(k) = abs(ec(k) - eulerG);
end;
result = [n p1 errp1 p2 errp2 en erren ec errec];
%
%----- End Function TestPiEtcLE1 -----

```

Table 1: Results of Tests on Functions for Calculating  $\pi$ ,  $e$ , and  $\gamma$ .

$n$	Pi1	ErrPi1	Pi2	ErrPi2	En	ErrEn	Ec	ErrEc
$10^1$	3.3396825	0.1980899	2.8444444	0.2971482	2.7182818	0.0000000	0.6263832	0.0491675
$10^3$	3.1395927	0.0020000	3.1384589	0.0031338	2.7182818	0.0000000	0.5777156	0.0004999
$10^6$	3.1415907	0.0000020	3.1415895	0.0000031	2.7182818	0.0000000	0.5772162	0.0000005
$10^9$	3.1415927	0.0000000	3.1415926	0.0000000	2.7182818	0.0000000	0.5772157	0.0000000

### 4.3 Analysis

We have very little to say analytically at the moment because we have not covered convergence of sequences and the order of convergence. Examining Table 1 we can say that the convergence of Pi1, Pi2, and Ec is very slow while En is much faster. Given that full precision in IEEE double-precision floating point arithmetic is about 16 decimal digits and that Pi1, Pi2, and Ec are accurate to about 8 or 9 digits for  $n = 10^9$ , we would need phenomenally high values of  $n$  to attain full precision. Clearly, these formulas are not adequate.

Figure 1 illustrates the behaviour of the error for each sequence but obscures the slow convergence of these sequences.

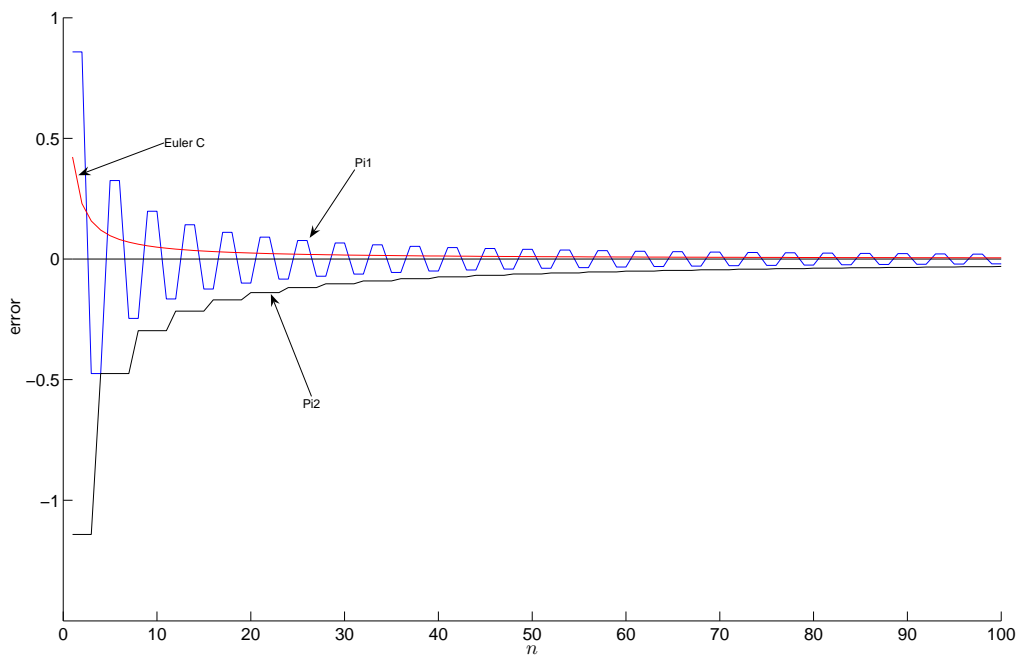


Figure 1: Convergence of Pi1, Pi2, and EulerC

## 4.4 Programming Notes

These notes apply to programming and programming languages in general.

### 4.4.1 Style

1. The function header should be highlighted.
2. Each function should have a brief description of its purpose, followed by the author's name and date, etc.
3. Consistent style and *mnemonic* names<sup>2</sup> for variables.
4. Consistent variable names for loops and sequences : we always use  $k$  as a counter which ranges from a lower bound 1 or 2 to an upper bound  $n/2$  or  $n$ . We reserve  $i$  and  $j$  for indexing vectors and matrices.
5. The body of each **for** -loop is indented 3 or 4 spaces. This can be done automatically by the `MATLAB` editor. **while** -loops and **if-then-else** statements should be indented in a similar manner.

### 4.4.2 Test Functions

Writing test functions is a vitally important part of any software project, no matter how small. The numerical linear algebra package LAPACK has a huge test function which is broken down into many smaller functions. In writing test software it is well to remember Dijkstra's immortal words

*Testing can show only the presence of errors, not their absence.*

Writing test functions can be difficult. The test function above took more time than writing and debugging the functions themselves.

Once a decent test function has been written and debugged it becomes easier to add new tests or alter old ones. For example, it would have been better to calculate the relative error rather than the absolute error for each function. This is easily done by changing `errp1(k) = abs(p1(k) - pi)` to `errp1(k) = abs(p1(k) - pi)/pi`. Try doing that change using the command line or a script file and you will see the benefit.

**Summarizing and Presenting Results** This is one of the most important aspects of test functions. A test program that churns out MBs of numbers or data is useless to humans (but it may be useful as input to other programs). The summary table for this exercise is fairly obvious : the constant value and its error for each value of  $n$ . Thus we have a  $(1 + 4 \times 2) \times 4$  table (or should that be the other way around?). The test function `TestPiEtcLE1` returns the result

```
result = [n p1 errp1 p2 errp2 en erren ec errec];
```

which concatenates 9 column vectors (each  $4 \times 1$ ), so that a  $4 \times 9$  table is displayed on the screen. This can be copied and pasted into your  $\LaTeX$  file, after which you add various commands to typeset this table. Typesetting tables and matrices is one of the tedious and boring jobs that afflicts those who must use `MATLAB` output in their reports. Applying the 'let the computer do the dirty work' principle I have written a `MATLAB` file `Mat2Latex.m` function that typesets matrices. The `MATLAB` command `Mat2Latex(result, 'R', 6)` prints the  $\LaTeX$  version of the results table to the screen.

<sup>2</sup>A name that reflects the purpose of the variable.



## 5 COMMENTS

Most of you did a good job programming in `MATLAB` and the `LaTeX`ed reports were fairly good also. The following are some points you should keep in mind for future reference.

1. Use the function headers given in the exercise.
2. Use **for**–loops where you know in advance the number of iterations in a loop. All the functions in this exercise should be written using **for**–loops.
3. Poor choice of variable names.
4. Poor program layout : use white-space and indenting.
5. Why do students have no faith in the unary minus? In 30 years of grading programs the following keeps recurring : `sign = (-1)*sign` when `sign = -sign` is what is needed.
6. Too many parentheses : `sum = (sum + (1/x))`.
7. Do not use anthropomorphisms : computers do not think, struggle, or get constipated. Humans do.

*Think before you compute. Think after you compute.*