

11.9 Project

Numerical Computation of Trigonometric Functions

This project in the textbook addresses the dilemma you face if you're stranded for life on a deserted island with only a very basic calculator that does not calculate transcendental functions. Obviously your first priority is to get modern science going on this miserable island. Realizing their central importance for scientific computation, you decide to use power series to calculate tables of values of trigonometric functions. Here we'll allow ourselves to employ modern technology itself, but we observe the spirit of the situation described by using only the four arithmetical operations $+$, $-$, \times , \div as we carry out the necessary approximate summation of series.

To calculate (approximate) values of the basic trigonometric functions we need only sum the Taylor series

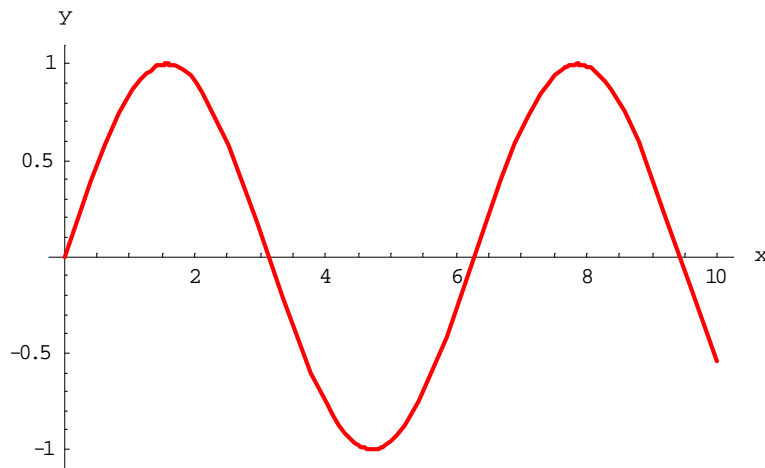
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots \quad (1)$$

and

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots, \quad (2)$$

which converge fairly rapidly unless x is large. And, because of the periodicity of these functions, we need only calculate their values for x between 0 and π .

But suppose that the famous number π itself is unknown on our forlorn island. Then, once we know how to calculate values of the sine function accurately and efficiently, we might proceed to plot the graph $y = \sin x$:



We observe that the initial positive zero of $\sin x$ appears to be a bit larger than 3. Consequently we **define** π to be this smallest positive zero of the sine function.

The remaining question is how to find the numerical value of π accurate to a reasonable number of decimal places. Why not start with the initial guess $x_0 = 3$ and repeat **Newton's iteration** with $f(x) = \sin x$ and $f'(x) = \cos x$,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{\sin x_n}{\cos x_n}, \quad (3)$$

until the resulting iterates stabilize. Isolated on our island, we naturally must use our series-based *approximate* sine and cosine functions in (3) to calculate successive approximations to the number π . In the specific technology sections for this project we do this, hoping to wind up with the approximation $\pi \approx 3.14159\,26536$ accurate to 10 decimal places. Then our island's development of trigonometry will be well on its way!

Using a Graphing Calculator

Suppose we have a TI graphing calculator on our deserted island, but its built-in transcendental functions — including $\cos(x)$ and $\sin(x)$ — are for some reason disabled. We therefore want to reconstruct these functions using only the four arithmetical operations $+$, $-$, \times , \div .

The key to summing the cosine series in (1) is to note the transition from the typical term

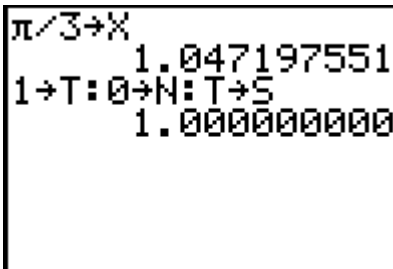
$$T = \pm \frac{x^N}{N!}$$

to the next term

$$T_{\text{next}} = \mp \frac{x^{N+2}}{(N+2)!} = -\frac{\pm x^N}{N!} \cdot \frac{x^2}{(N+1)(N+2)} = -T \cdot \frac{x^2}{(N+1)(N+2)}$$

with N even. The situation with the sine series in (2) is precisely the same, except that N is odd.

Given x , the commands



```

π/3→X
1.047197551
1→T:0→N:T→S
1.000000000
  
```

store x as well as

- the initial term $T = 1$ in the cosine series,
- then initial index (or exponent) $N = 0$, and
- the initial sum $S = T$.

With each press of the **Enter** key, the multi-line command

```
-T*X^2/((N+1)(N+2)
) ) → T : N+2 → N : S+T → S
.45169
.50180
.49996
.50000
```

- uses the rule of transition above to calculate the next term in the series,
- increments the index by 2 to prepare for the next step, and
- adds the new term to the old sum to obtain the new sum.

We see that after adding 4 or 5 terms we have calculated $\cos(\pi/3) \approx 0.50000$ accurate to 5 decimal places.

In order to calculate sines instead of cosines, we need only change the initialization appropriately:

```
π/3 → X
1.04720
X → T : 1 → N : T → S
1.04720
```

Then the same multi-line iterative step still works:

```

-T*X²/((N+1)(N+2
))→T:N+2→N:S+T→S
.85580
.86630
.86602
.86603

```

The following TI-83 programs **COSINE** and **SINE** automate this procedure, continuing to add each series term that exceeds 10^{-10} in absolute value (so as to calculate the sine and cosine of x accurate to 9 decimal places). (Recall that the calculator's own trigonometric functions are **COS** and **SIN** rather than **COSINE** and **SINE**.)

```

PROGRAM: COSINE
:1→T:0→N:T→S
:While abs(T)>1E
-10
: -T*X²/((N+1)(N+
2))→T:N+2→N:S+T→
S:End
:S

```

```

PROGRAM: SINE
:X→T:1→N:T→S
:While abs(T)>1E
-10
: -T*X²/((N+1)(N+
2))→T:N+2→N:S+T→
S:End
:S

```

Note that only the initialization lines differ. Checking them out:

```

π/4→X:PrgmSINE
.707106781
π/2→X:PrgmSINE
1.000000000
π→X:PrgmCOSINE
-1.000000000

```

We leave it to the reader to write **COSINES** and **SINES** programs to calculate lists of trigonometric values. You need only replace the single line **:prgmNLOG** in the **NLOGS** program above by either **:prgmCOSINE** or **:prgmSINE**, as desired.

The Value of the Number Pi

Given the sine and cosine functions, we can calculate π as the zero of the equation $\sin x = 0$ that is near the initial guess $x = 3$. The following TI-83 screen encapsulates Newton's method in the **Y=** menu.

```

Plot1 Plot2 Plot3
\Y1=sin(X)
\Y2=cos(X)
\Y3=X-Y1/Y2
\Y4=
\Y5=
\Y6=
\Y7=

```

We have defined the function $f(x) = \sin x$ as **Y1** and its derivative $f'(x) = \cos x$ as **Y2**. Then **Y3** implements Newton's iteration itself. Having entered our initial guess, we need only Enter **Y3** \rightarrow **X** repeatedly until the iterates "stabilize".

```

3→X
3.000000000
Y3→X
3.142546543
3.141592653
3.141592654
3.141592654

```

This is the true value of π accurate to the 9 decimal places shown!

Remark: We have used here the calculator's built-in sine and cosine functions, and leave it to the reader to think about how to use our "desert island" trig functions instead.

Using Maple

Suppose we have a computer and a copy of Maple on our deserted island, but its built-in transcendental functions — including $\cos(x)$ and $\sin(x)$ — are for some reason disabled. We therefore want to reconstruct these functions using only the four arithmetical operations $+$, $-$, \times , \div .

Similarly, the key to summing the cosine series in (1) is to note the transition from the typical term

$$T = \pm \frac{x^N}{N!}$$

to the next term

$$T_{\text{next}} = \mp \frac{x^{N+2}}{(N+2)!} = -\frac{\pm x^N}{N!} \cdot \frac{x^2}{(N+1)(N+2)} = -T \cdot \frac{x^2}{(N+1)(N+2)}$$

with N even. The situation with the sine series in (2) is precisely the same, except that N is odd.

The Maple procedures below use this rule of transition to sum the cosine and sine series in (1) and (2). Each starts the summation with

- the initial term ($T = 1$ for the cosine series, $T = x$ for the sine series),
- the initial sum $S = T$, and
- then initial index ($N = 0$ for the cosine series, $N = 1$ for the sine series).

Each step in the "while loop" then calculates and adds (to the preceding partial sum S) each successive new term T so long as it exceeds 10^{-16} in absolute value, and increments the index N by 2 to prepare for the next step.

```
cosine := proc(x)
local T, S, N;
T := 1;
S := T;
N := 0;
while abs(T) > 10^(-16) do
T := -((T*x^2)/((N + 1)*(N + 2)));
S := S + T;
N := N + 2;
od;
S;
end;
```

We need only change the initial term and index to convert our cosine function to a sine function.

```
sine := proc(x)
local T, S, N;
T := x;
S := T;
N := 1;
while abs(T) > 10^(-16) do
T := -((T*x^2)/((N + 1)*(N + 2)));
S := S + T;
```

```

N := N + 2;
od;
S;
end:

```

First we check a few familiar values see that all is well.

```

npi := evalf(Pi,20):
sine(npi/4);
cosine(npi/3);
sine(npi/2);
cosine(npi);

```

.7071067813
.5000000001
.9999999993
-1.000000000

(Recall that Maple's own trigonometric functions are **cos** and **sin** rather than **cosine** and **sine**.) Indeed, we can use Maple's **map** function to apply our new trigonometric functions to a whole column of x -values simultaneously, and thereby calculate a brief table of cosines and sines (with 15° increments, for instance) in one fell swoop:

```

x := [seq(i*Pi/12, i=0..12)];

```

x:= [0, Pi/12, Pi/6, Pi/4, Pi/3, 5 Pi/12, Pi/2,
7 Pi/12, 2 Pi/3, 3 Pi/4, 5 Pi/6, 11 Pi/12, Pi]

```

vc := map(cosine,evalf(x,20)):
vs := map(sine,evalf(x,20)):
xdeg := [seq(15*i*deg,i=0..12)]:
with(linalg):transpose([xdeg,vc,vs]);

```

[0, 1, 0]
[15 deg, .9659258262, .2588190451]
[30 deg, .8660254038, .5000000001]
[45 deg, .7071067812, .7071067812]
[60 deg, .5000000001, .8660254037]
[75 deg, .2588190452, .9659258263]
[90 deg, .4805793342e-10, 1.000000000]
[105 deg, -.2588190456, .9659258261]
[120 deg, -.5000000004, .8660254036]
[135 deg, -.7071067813, .7071067820]
[150 deg, -.8660254042, .5000000008]
[165 deg, -.9659258272, .2588190457]
[180 deg, -1.000000000, .1339342607e-8]

The Value of the Number Pi

Here we've used Maple's built-in value of Pi but we can use our own sine and cosine functions to approximate π as the smallest positive zero of the equation $\sin x = 0$. The

following Maple function carries out a single iteration of Newton's method for this equation.

```
newt := x -> x - sine[x]/cosine[x]:
```

We need only start with the initial guess $x = 3$ and iterate **newt** until the iterates "stabilize".

```
Digits := 15:x := 3.0:  
x := newt(x);           x := 3.14254654307428  
x := newt(x);           x := 3.14159265330047  
x := newt(x);           x := 3.14159265358979  
x := newt(x);           x := 3.14159265358979
```

This is the true value of pi accurate to the 14 decimal places shown!

Using Mathematica

Suppose we have a computer and a copy of Mathematica on our deserted island, but its built-in transcendental functions — including $\cos(x)$ and $\sin(x)$ — are for some reason disabled. We therefore want to reconstruct these functions using only the four arithmetical operations $+$, $-$, \times , \div .

The key to summing the cosine series in (1) is to note the transition from the typical term

$$T = \pm \frac{x^N}{N!}$$

to the next term

$$T_{\text{next}} = \mp \frac{x^{N+2}}{(N+2)!} = -\frac{\pm x^N}{N!} \cdot \frac{x^2}{(N+1)(N+2)} = -T \cdot \frac{x^2}{(N+1)(N+2)}$$

with N even. The situation with the sine series in (2) is precisely the same, except than N is odd. The following Mathematica functions use this rule of transition to sum the cosine and sine series in (1) and (2). Each starts the summation with

- the initial term ($T = 1$ for the cosine series, $T = x$ for the sine series),
- the initial sum $S = T$, and
- then initial index ($N = 0$ for the cosine series, $N = 1$ for the sine series).

Each step in the "while loop" then calculates and adds (to the preceding partial sum S) each successive new term T so long as it exceeds 10^{-16} in absolute value, and increments the index N by 2 to prepare for the next step.

```
cosine[x_] :=
Module[{T, S, N},
T = 1;
S = T;
N = 0;
While[Abs[T] > 10^(-16),
T = -((T*x^2)/((N + 1)*(N + 2)));
S = S + T;
N = N + 2];
S]
```

We need only change the initial term and index to convert our cosine function to a sine function.

```
sine[x_] :=
Module[{T, S, N},
T = x;
S = T;
N = 1;
While[Abs[T] > 10^(-16),
T = -((T*x^2)/((N + 1)*(N + 2)));
S = S + T;
N = N + 2];
S]
```

First we check a few familiar values see that all is well.

```
sine[Pi/4] // N
0.707107
```

```
cosine[Pi/3] // N
0.5
```

```
sine[Pi/2] // N
1.
```

```
cosine[Pi] // N
-1.
```

(Recall that Mathematica's own trigonometric functions are **Cos** and **Sin** rather than **cosine** and **sine**.) Indeed, we can use Mathematica's **Map** function to apply our new trigonometric functions to a whole column of x -values simultaneously, and thereby calculate a brief table of cosines and sines (with 15° increments, for instance) in one fell swoop:

```
x = Table[x, {x, 0, Pi, Pi/12}]
{0, Pi/12, Pi/6, Pi/4, Pi/3, 5 Pi/12, Pi/2, 7 Pi/12, 2
Pi/3, 3 Pi/4, 5 Pi/6, 11 Pi/12, Pi}
```

```
Transpose[
{180*X*deg/Pi, Map[cosine,X], Map[sine,X]} ]
// N // TableForm
0                1.                0
15. deg         0.965926          0.258819
30. deg         0.866025          0.5
45. deg         0.707107          0.707107
60. deg         0.5               0.866025
75. deg         0.258819          0.965926
90. deg         4.25663 10^(-17) 1.
105. deg        -0.258819         0.965926
120. deg        -0.5              0.866025
135. deg        -0.707107         0.707107
150. deg        -0.866025         0.5
165. deg        -0.965926         0.258819
180. deg        -1.               3.59188 10^(-16)
```

The Value of the Number Pi

Here we've used Mathematica's built-in value of Pi but we can use our own sine and cosine functions to approximate π as the smallest positive zero of the equation $\sin x = 0$. The following Mathematica function carries out a single iteration of Newton's method for this equation.

```
newt[x_] := x - sine[x]/cosine[x]
```

The Mathematica function **FixedPoint[f, x]** applies the function **f** repeatedly to the value **x** until the result no longer changes. Thus

```
FixedPoint[newt, 3.]
3.14159
```

```
N[FixedPoint[newt, 3.], 15]
3.14159265358979
```

This is the true value of π accurate to the 14 decimal places shown!

Using MATLAB

Suppose we have a computer and a copy of MATLAB on our deserted island, but its built-in transcendental functions — including $\cos(x)$ and $\sin(x)$ — are for some reason disabled. We therefore want to reconstruct these functions using only the four arithmetical operations $+$, $-$, \times , \div .

The key to summing the cosine series in (1) is to note the transition from the typical term

$$T = \pm \frac{x^N}{N!}$$

to the next term

$$T_{\text{next}} = \mp \frac{x^{N+2}}{(N+2)!} = -\frac{\pm x^N}{N!} \cdot \frac{x^2}{(N+1)(N+2)} = -T \cdot \frac{x^2}{(N+1)(N+2)}$$

with N even. The situation with the sine series in (2) is precisely the same, except that N is odd. Consequently, the following MATLAB functions sum the cosine and sine series. Each starts with

- the initial term ($T = 1$ for the cosine series, $T = x$ for the sine series),
- the initial sum $S = T$, and
- then initial index ($N = 0$ for the cosine series, $N = 1$ for the sine series).

Each step in the "while loop" then calculates and adds (to the preceding partial sum S) each successive new term T so long as it exceeds 10^{-16} in absolute value, and increments the index N by 2 to prepare for the next step.

```
function y = cosine(x)
% our own cosine function
T = 1; % initial term
S = T; % initial sum
N = 0; % initial index
while max(abs(T)) > 1e-16
T = -T.*x.*x./((N+1)*(N+2)); % new sum
S = S + T; % new term
N = N + 2; % new index
end
y = S; % approx value of cos(x)
```

We need only change the initial term and index to convert our cosine function to a sine function.

```
function y = sine(x)
% our own sine function
T = x; % initial term
S = T; % initial sum
N = 1; % initial index
while max(abs(T)) > 1e-16
T = -T.*x.*x./((N+1)*(N+2)); % new term
S = S + T; % new sum
N = N + 2; % new index
end
y = S; % approx value of sin(x)
```

First we check a few familiar values see that all is well.

```
sine(pi/4)
ans = 0.7071
```

```
cos(pi/3)
ans = 0.5000
```

```
sine(pi/2)
ans = 1.0000
```

```
cosine(pi)
ans = -1.0000
```

(Recall that MATLAB's own trigonometric functions are **cos** and **sin** rather than **cosine** and **sine**.) Because our **cosine** and **sine** functions are "vectorized", we can apply them to a whole column of x -values simultaneously, and thereby calculate a brief table of cosines and sines (with 15° increments, for instance) in one fell swoop:

```
x = (0 : 15 : 180)*pi/180;
[180*x/pi; cosine(x); sine(x)]'
ans =
0          1.0000    0
15.0000    0.9659    0.2588
30.0000    0.8660    0.5000
45.0000    0.7071    0.7071
60.0000    0.5000    0.8660
75.0000    0.2588    0.9659
90.0000    0.0000    1.0000
105.0000   -0.2588    0.9659
120.0000   -0.5000    0.8660
135.0000   -0.7071    0.7071
150.0000   -0.8660    0.5000
165.0000   -0.9659    0.2588
180.0000   -1.0000    0.0000
```

The Value of the Number Pi

Here we've used MATLAB's built-in value of pi, but we can use our own sine and cosine functions to approximate π as the smallest positive zero of the equation $\sin x = 0$. The MATLAB function

```
function y = newt(x)
y = x - f(x)./df(x);
```

carries out a single iteration in Newton's method, once we've defined the function f and its derivative df :

```
function y = f(x)
y = sine(x);

function y = df(x)
% derivative of f(x)
y = cosine(x);
```

Then we get

```
format long;
x = 3;
x = newt(x)
x = 3.14254654307428

x = newt(x)
x = 3.14159265330048

x = newt(x)
x = 3.14159265358979

x = newt(x)
x = 3.14159265358979
```

This is the true value of π accurate to the 14 decimal places shown!