Curtin University
DEPARTMENT OF MATHEMATICS & STATISTICS

ACCELERATED MATHEMATICS UNITS MATH1017 AND MATH1021

Lab Sheet 6

This week we want you to code a function that will calculate $\sin(x)$ for general x, using what you coded for sin and cos, last week. The idea is to use symmetries of sin (and cos) to write $\sin(x)$ as either $\sin(x_1)$ or $\cos(x_1)$ where $x_1 \in [0, \pi/4]$. This ensures very quick convergence and accuracy and hence a good coding solution for $\sin(x)$ for all $x \in \mathbb{R}$.

Background

The idea this week is to use the following properties of sin:

$$\begin{aligned}
\sin(-x) &= -\sin(x), && \sin \text{ is odd} \\
\sin(x) &= \sin(x + 2\pi), && \sin \text{ is perodic with period } 2\pi \\
\sin(x) &= \sin(\pi - x), && \sin \text{ is symmetric about } x = \pi/2 \\
\sin(x) &= \cos(\pi/2 - x), && \text{complementary angle property}
\end{aligned}$$

the easiest way to convince yourself of the correctness of these is to sketch a sinusoidal graph representing $y = \sin(x)$.

Ingredients: `proc`, `if`, and your two `proc`s from last week.

The approach we'll take is to create a recursive function `sinx` that calls itself until the argument `x` is in $[0, \pi/2]$. Once we have that right, we'll organise that your new `sinx` calls the functions from last week, with one of them renamed.

Here is a suggested approach.

1. First you need a function `mod2pi` that gives the remainder on division by 2π. The following will do it:

```
mod2pi := x -> x - floor(x/(2*Pi)) * 2*Pi;
```

   This syntax emulates the mathematics notation of $x \mapsto f(x)$, and is equivalent to the following `proc`,

```
mod2pi := proc(x)
               return x - floor(x/(2*Pi)) * 2*Pi
             end;
```

2. Now code the following function,

```
sinx := proc(x)
          if is(x < 0) then
            printf("sin(%g) = -sin(%g)\n", x, -x);
            return -sinx(-x)
          elif is(x > 2 * Pi) then
            printf("sin(%g) = sin(%g)\n", x, mod2pi(x));
            return sinx(mod2pi(x))
          else
            printf("sin(%g) = sin(%g)\n", x, x);
            return Sinx(x)
          fi
        end;
```

and run it with a few values, e.g. try `-10`. The `printf` statements do *formatted printing*. The conditional tests require `is(...)`. Perhaps you should find out what that does, by reading the documentation and omitting it to see what happens. You should notice that eventually the argument is in $[0, 2\pi]$.

3. Now extend your function to:

```
    sinx := proc(x)
              if is(x < 0) then
                printf("sin(%g) = -sin(%g)\n", x, -x);
                return -sinx(-x)
              elif is(x > 2 * Pi) then
                printf("sin(%g) = sin(%g)\n", x, mod2pi(x));
                return sinx(mod2pi(x))
              elif is(x > Pi) then
                printf("sin(%g) = sin(%g)\n", x, x - 2*Pi);
                return sinx(x - 2*Pi)
              elif is(x > Pi/2) then
                printf("sin(%g) = sin(%g)\n", x, Pi - x);
                return sinx(Pi - x)
              elif is(x > Pi/4) then
                printf("sin(%g) = cos(%g)\n", x, Pi/2 - x);
                return cosx(evalf(Pi/2 - x))
              else
                printf("sin(%g) = sin(%g)\n", x, x);
                return Sinx(evalf(x))
              fi
            end;
```

and again run it with a few values, e.g. try `-10`. You should notice that eventually the argument is in $[0, \pi/4]$.

4. Now we use your code from last week. All you need to do is `read` them in, so that they are defined for Maple. You can use `cosx` as it is, but `sinx` needs to be renamed, since we have already have a `sinx`. If you look in the code above, I've assumed you will rename last week's function `Sinx`. Also, if you defined `cosx` and (now) `Sinx` with a second argument, you will need to change the calls to `cosx` and `Sinx` to have a second argument, e.g. you will need something like

```
                return cosx(evalf(Pi/2 - x), 1e-12)
```

and similar for `Sinx`.

If you didn't finish `cosx` last week, but did finish `sinx` your code will still work with the lines

```
              elif is(x > Pi/2) then
                printf("sin(%g) = sin(%g)\n", x, Pi - x);
                return cosx(evalf(Pi/2 - x), 1e-12)
```

removed or commented out.

5. Run your `sinx` function (i.e. `proc`) with argument `x` set to -70., and compare it with Maple's `sin(-70.)`.