Horner's Method and Trigonometric Functions

Paul Fischer
Department of Physics
California State University Long Beach

February 3, 2020

Abstract

Horner's method is used to increase the efficiency associated with computing finite polynomials. We apply this scheme in a `Fortran90` code that computes a finite number of terms from the Taylor series for sine and cosine. We plot these approximations using `Plot2` in the domain `x` $\in$ [-2π, 2π] against `Fortran90`'s built-in `sin` and `cos` functions for different values of `imax`, the highest degree of the Taylor series approximation. The errors associated with these computations are discussed qualitatively.

1 Introduction

Numerically approximating a function by calculating and adding each term of its Taylor series can be computationally intensive when trying to achieve a desired precision. In 1819, British Mathematician William George Horner published a scheme for evaluating polynomials of degree `n` with only `n` additions and `n` multiplications [1]. This method is shown in Fig. 1. Section 2 will discuss our `Fortran90` code that approximates the sine and cosine functions using Horner's method and will qualitatively evaluate the precision of the results of this method when plotted against the built-in `Fortran90 sin` and `cos` functions in the domain `x` $\in$ [-2π, 2π] for different values of `imax`, the highest degree of the Taylor series approximation.

$$
\begin{aligned}
p(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n \\
&= a_0 + x\Big(a_1 + x\Big(a_2 + x\big(a_3 + \cdots + x(a_{n-1} + x\, a_n)\cdots\big)\Big)\Big)
\end{aligned}
$$

Figure 1: Horner's method for finite polynomials [1].

2 Applying Horner's Method to the Sine and Cosine Taylor Series

We write a `Fortran90` code that uses Horner's method to approximate the sine and cosine functions from their Taylor series to a given degree `imax`. The Makefile used for the code is shown in in Listing 1.

Listing 1: `Makefile`

```

objs1 = numtype.o htrig.o

prog1 = htrig

f90 = gfortran

f90flags = -O3

libs = -framework Accelerate

ldflags = $(libs)

all: $(prog1)

$(prog1): $(objs1)
	$(f90) $(ldflags) -o $@ $(objs1)

clean:
	rm -f $(prog1) *.{o,mod} fort.*

.suffixes: $(suffixes) .f90

```

```
%.o: %.f90
	$(f90) $(f90flags) -c $<
```

The code is shown for sine and cosine in Listing 3 for `imax` = 100. The precision is defined in the Module `numtype` shown in Listing 2. For each trigonometric function, the code uses a `do` loop to create an array `coeff` that contains the coefficients of the desired Taylor series up to the degree `imax`. Another `do` loop writes to the files `fort.1` (for sine) and `fort.3` (for cosine) Horner's method approximations (contained in a nested `do` loop) of the trigonometric functions against their corresponding `x`-values in the domain `x` $\in$ [-2π, 2π] separated by units of $\pi/30$. This `do` loop also writes to the files `fort.2` and `fort.4` the same `x`-values with their new corresponding `y`-values given by the built-in `sin` and `cos` functions respectively.

Listing 2: Module `numtype`

```

module numtype
    save
    integer, parameter :: dp = selected_real_kind(15,307)
    real(dp), parameter :: pi = 4*atan(1._dp)

end module numtype
```

Listing 3: Program `htrig.f90`

```

program htrig

    use numtype
    implicit none

    real(dp) :: coeff(0:100)
    real(dp) :: x, y, dx
    integer :: i, imax

    ! Horner scheme for sine function

    imax = 100

    coeff(0) = 0
```

```fortran
    coeff(1) = 1
    coeff(2:imax:2) = 0

    do i = 3, imax-1, 2
        coeff(i) = -coeff(i-2) / (i*(i-1))
    end do

    x = -2*pi
    dx = pi/30

    do while (x .le. 2*pi)
        y = coeff(imax)
        do i = imax-1, 0, -1
            y = coeff(i) + x*y
        end do
        write(1,*) x, y
        y = sin(x)
        write(2,*) x, y
        x = x+dx
    end do

    ! --------------------------------

    ! Horner scheme for cosine function

    coeff(0) = 1
    coeff(1:imax:2) = 0

    do i = 2, imax, 2
        coeff(i) = -coeff(i-2) / (i*(i-1))
    end do

    x = -2*pi
    dx = pi/30

    do while (x .le. 2*pi)
        y = coeff(imax)
        do i = imax-1, 0, -1
        y = coeff(i) + x*y
        end do
```

```
            write(3,*) x, y
            y = cos(x)
            write(4,*) x, y
            x = x+dx
        end do

end program htrig
```

We can run the code by typing `./htrig`. The files `fort.1` and `fort.2` are plotted using `Plot2` in Fig. 2 with `imax` = 10. This plot shows the large difference between the Horner's method approximation for sine and the built-in `sin` function for such a small `imax` value. The values for `imax` were chosen in an effort to visually see the limit where the Horner's method plots start to coincide with the built-in function plots. An `imax` value of 16 was used in Fig. 3. This value shows that the Horner approximation is rapidly converging to the built-in function within the interval but is still visually off at the ends. It is hard to see any difference between the Horner's method approximation and the built-in function when `imax` = 100 was used for Fig. 4. The plot of the Horner approximation of the cosine function against the built-in `cos` function for `imax` = 100 is shown in Fig. 5.

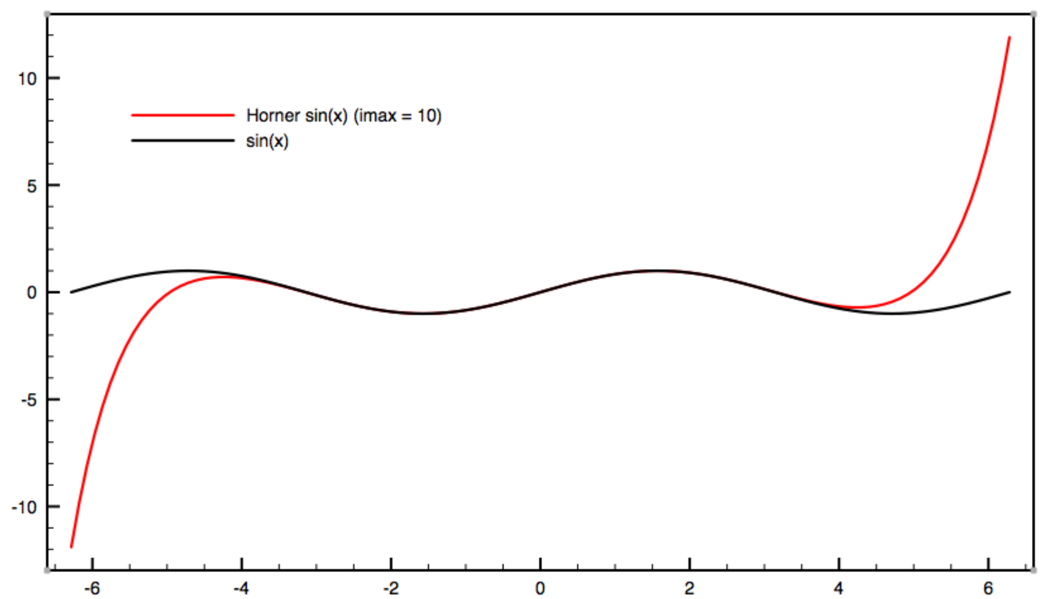

Figure 2: Horner sin vs. built-in `sin` for `x` $\in$ [-2π, 2π], `imax` = 10.

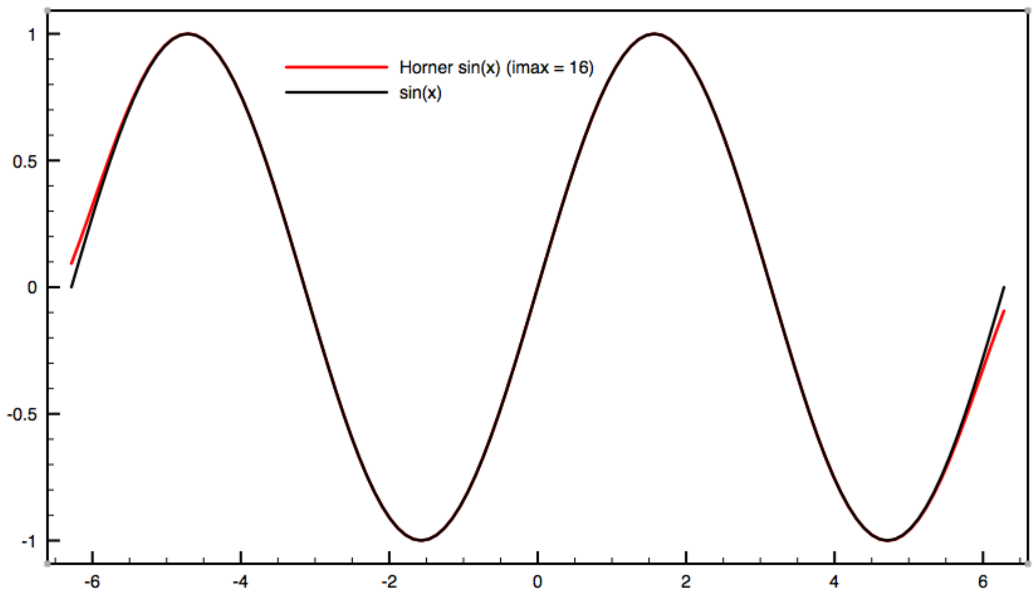

Figure 3: Horner sin vs. built-in `sin` for $\mathtt{x} \in$ [-2π, 2π], `imax` = 16.

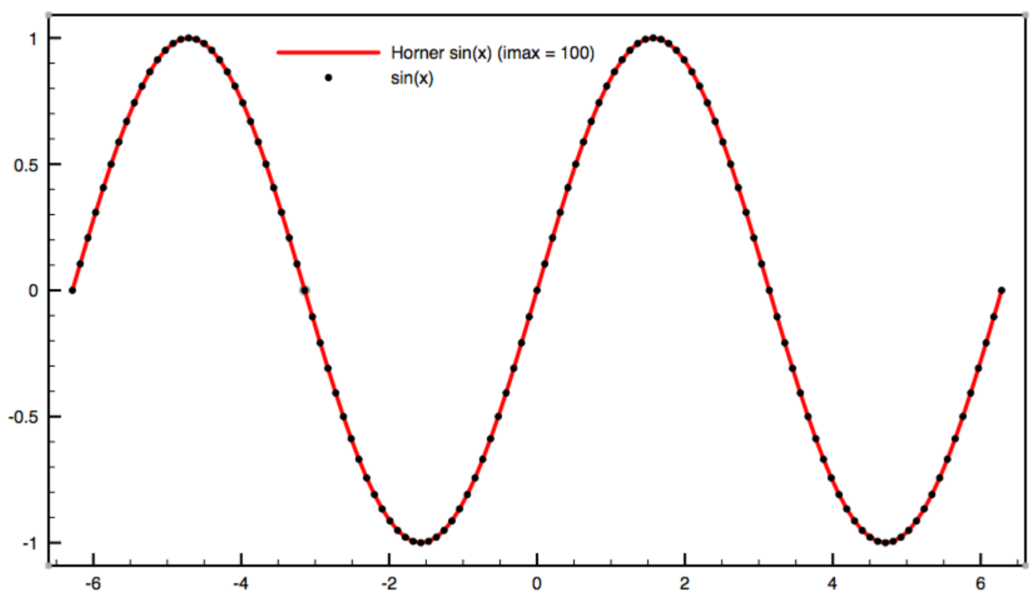

Figure 4: Horner sin vs. built-in `sin` for $\mathtt{x} \in$ [-2π, 2π], `imax` = 100.

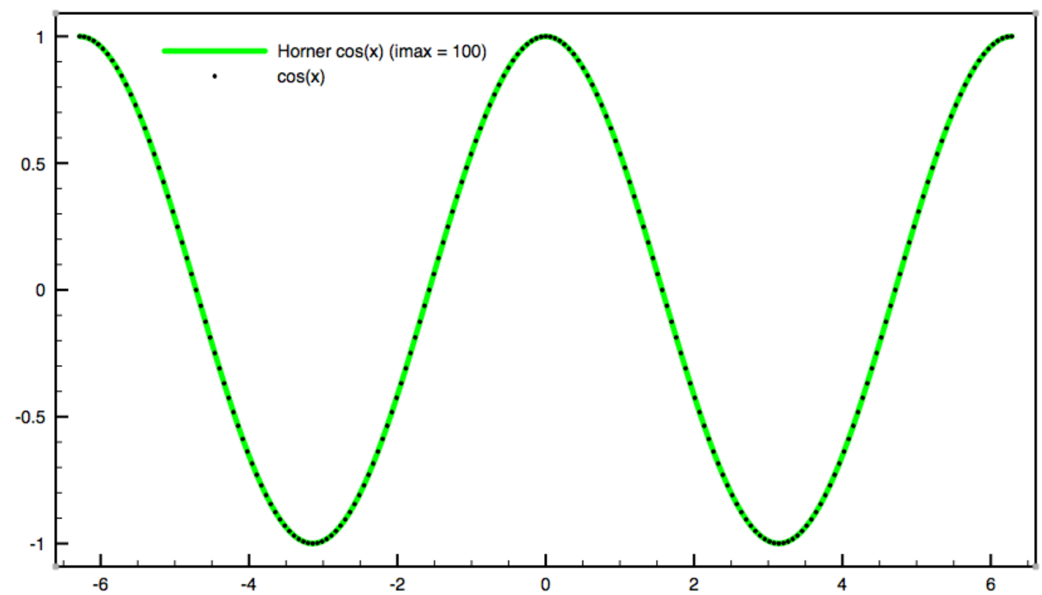

Figure 5: Horner cos vs. built-in `cos` for `x` $\in$ [-2π, 2π], `imax` = 100.

3 Summary and conclusions

We presented our `Fortran90` code for the Horner's method approximation of sine and cosine and qualitatively looked at the degrees `imax` to which the approximation converges to the built-in functions. Horner's method allows for the calculation of these approximations to be done more efficiently than having the program add and multiply each term of the Taylor series separately. The plots indicate that at around 20 degrees the Horner approximation starts to be visually indistinguishable on the graph from the built-in functions.

References

[1] *Horner's Method* (n.d). In *Wikipedia*. Retrieved January 30, 2020, https://en.wikipedia.org/wiki/Horner