Homework 3, Computational Physics

Seann Smallwood

October 24, 2020

Abstract

This goal of this work was to analyze systems for which curve fittings and approximations were essential. The first problem we fit a curve to a specified function, the second, we used a summation relationship and its partial sums to approximate values for which the sum diverges.

1 Introduction

1.1 Harmonic Oscillator Fit

We used a potential function $V(x)$

$$V(x) = -2 + 2(1 - \frac{1}{2}exp(-\frac{x}{2} + 2))^2$$

The objective for this problem was to use the potential function for a harmonic oscillator to fit a curve to the absolute minimum of the given function $V(x)$.

1.2 Sum Approximation

Given the following sum $\zeta(s)$

$$\zeta(s) = \sum_{n=1}^{\infty} n^{-s}$$

The objective of this task was to calculate a few values of ζ and use these to predict the value of ζ when $s = 1$.

1.3 Setup and General Methods

Fortran was used to analyze each system. To implement common mathematical terms and define the precision to which values were calculated a file was created. This file, named numtype, is shown below.

```
module numtype

integer,parameter :: dp = selected_real_kind(15,307)
integer,parameter :: qp = selected_real_kind(33,4931)
real(dp), parameter :: pi = 4*atan(1._dp)
complex(dp), parameter :: iic = (0._dp,1._dp)

end module numtype
```

A Makefile, figure 3, was used to compile the fortran code and create and executable file.

```
OBJS1 = numtype.o thielecf.o prob2.o

PROG1 = approx

F90 = gfortran

F90FLAGS = -O3 -funroll-loops  -fexternal-blas

LIBS = -framework Accelerate

LDFLAGS = $(LIBS)

all: $(PROG1)

$(PROG1): $(OBJS1)
$(F90) $(LDFLAGS) -o $@ $(OBJS1)

clean:
rm -f $(PROG1) *.{o,mod} fort.*

.SUFFIXES: $(SUFFIXES) .f90

.f90.o:
$(F90) $(F90FLAGS) -c $<
```

2 Solutions

2.1 Harmonic Oscillator Fit

First, the potential function $V(x)$ was plotted to find the minimum.

$$V(x) = -2 + 2(1 - \frac{1}{2}exp(-\frac{x}{2} + 2))^2$$

The initial plot of the function did not make the minima immediately obvious, so we tooke the derivative of $V(x)$ giving

$$dV(x)/dx = 7.389e^{-x}(e^{x/2} - 3.69453)$$

Finding the zero's of dV/dx narrowed our parameters to find the minimum of $V(x)$ The following is the plot of the minimum of $V(x)$

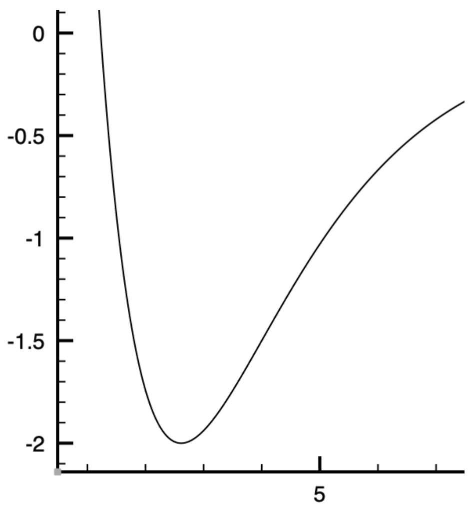

Figure 1: Minimum of $V(x)$

The goal was then to fit a standard harmonic oscillator potential function $U(x)$ to the minimum of $V(x)$.

$$U(x) = \frac{1}{2}a(x - b)^2 + c$$

a, b, c are parameters that define and shape the harmonic potential function. An intial guess was made for each, a, b, and c based on the plot of $V(x)$. These paramters were then used to iterate through a fitting algorithim to fit the harmonic oscillator potential, $U(x)$, to the given potential function, $V(x)$.

Below is the code used to genarte the plot, initialize and call the fitting algortihim, and asses the strength of fit using a chi squared calculation.

```
    module setup

    use numtype
    implicit none

    integer, parameter :: npmax = 400, npar = 3
    integer, parameter :: nspmin = 1, nspmax = 250
    real(dp) :: xx(1:npmax), yy(1:npmax)
    integer ::  nsp, ical, iprint

end module setup

program problem_1

    use setup
    implicit none

    integer :: stat, i, itmin, itmax, in
    real(dp), external :: chi2
    real(dp) :: xstart(npar), fstart, stepi, epsf, dx


    xx(1) = 1.0_dp                                                         !-----------------
    dx = .02_dp

    do in = 1, 250                                                         !This loop populat
                                                                           !given function
        yy(in) = -2.0_dp + (2 * ((1 - (0.5*(exp((-xx(in)/2)+2))))**2))

        write(11,*) xx(in), yy(in)                                         !used to graph the

        xx(in+1) = xx(in) + dx
    end do                                                                 !-----------------


    xstart(1:npar) = (/ 2.0_dp , 2.5_dp, -2.0_dp /)
    xstart(1:npar) = (/ 0.5573_dp , 3.2226_dp, -2.0384_dp /)
```

```
    ical = 0
    iprint = 7
    fstart = chi2 (xstart)
    stepi= 0.05_dp
    epsf = 0.001_dp
    itmin = 100
    itmax = 1000

    iprint = 0
    call downhill(npar,chi2,xstart,fstart,stepi,epsf,itmin,itmax)

    iprint = 17
    fstart = chi2 (xstart)
    print *, xstart(1:npar)

end program problem_1

function chi2( par )

    use setup
    implicit none
    real(dp) :: chi2
    real(dp) :: par(npar)
    real(dp) :: K, mid, x, fi, height
    integer :: i

    ical = ical + 1
    K = par(1); mid = par(2); height = par(3)          !using harmonic function V(x) = 1/2

    chi2 = 0        ! chi^2

    do i = nspmin, nspmax

        x = xx(i)
        fi = 0.5_dp * K * (x - mid)**2 + height         !harmonic function V(x) = 1/2 K (x

        chi2 = chi2 +  ( yy(i) - fi )**2  * 1/sqrt(  2._dp + yy(i) )

    end do
    chi2 = chi2 / abs(nspmax-nspmin)
```

```
    print '(i4,2x,3f12.2,3x,f20.4)',ical, par(1:npar), chi2

    ! printing
    if (  iprint /= 0 ) then

        do i = nspmin, nspmax

            x = xx(i)
            fi =  0.5_dp * K * (x-mid)**2 + height
            write( unit=iprint, fmt='(3f15.4)') xx(i), yy(i)
            write( unit=iprint + 1, fmt='(3f15.4)') xx(i),  fi

        end do

    end if

end function chi2
```

The fitting algorithim used the inital parameters and iterates through to optimize these parameters. It is called the downhill method and was used as follows

```
    subroutine downhill(n,func,xstart,fstart,stepi,epsf,itmin,iter)
!
!   n           dimension of the problem
!   func        function
!   xstart      starting values
!   fstart      conrespoding function value
!   stepi       relative stepsize for initial simplex
!   epsf        epsilon for termination
!   itmin       termination is tested if itmin < it
!   iter        maximum number of iterations
!

    use numtype
    implicit none
    integer :: n, iter, itmin
    real(dp), external :: func
    real(dp) :: xstart(1:n), fstart, stepi, epsf
    real(dp), parameter :: alph=1._dp, gamm=2._dp, &
                           rho=0.5_dp, sig=0.5_dp
```

```
real(dp) :: xi(1:n,1:n+1), x(1:n,1:n+1), &
    fi(1:n+1), f(1:n+1),  &
    x0(1:n), xr(1:n), xe(1:n), xc(1:n), &
    fxr, fxe, fxc, deltaf
integer :: i, ii, it

xi(1:n,1) = xstart(1:n);    fi(1) = fstart
do i = 2, n+1
    xi(1:n,i)=xi(1:n,1)
    xi(i-1,i)=xi(i-1,i)*(1+stepi)
    fi(i)=func(xi(1:n,i))
end do

do it = 1, iter

    do i = 1, n+1                           ! ordering
        ii = minloc(fi(1:n+1),dim=1)
        x(1:n,i) = xi(1:n,ii);  f(i) = fi(ii)
        fi(ii) = huge(0._dp)
    end do
    xi(1:n,1:n+1) = x(1:n,1:n+1)
    fi(1:n+1) = f(1:n+1)

    x0(1:n) = sum(x(1:n,1:n),dim=2)/n   ! central

    if ( itmin < it ) then              ! condition for exit
        deltaf = (f(n)-f(1))
        !write(777,*) it,deltaf
        if(deltaf < epsf ) exit
    end if

    xr(1:n) = x0(1:n)+alph*(x0(1:n)-x(1:n,n+1))
    fxr = func(xr)
    if( fxr < f(n) .and. &              ! reflection
            f(1) <= fxr ) then
        xi(1:n,n+1) = xr(1:n);  fi(n+1) = fxr
        cycle

    else if ( fxr < f(1) ) then         ! expansion
        xe(1:n) = x0(1:n)+gamm*(x0(1:n)-x(1:n,n+1))
```

```
            fxe = func(xe)
            if( fxe < fxr ) then
                xi(1:n,n+1) = xe(1:n);  fi(n+1) = fxe
                cycle
            else
                xi(1:n,n+1) = xr(1:n);  fi(n+1) = fxr
                cycle
            end if

        else if ( fxr >= f(n) ) then        ! contraction
            xc(1:n) = x(1:n,n+1)+rho*(x0(1:n)-x(1:n,n+1))
            fxc = func(xc)
            if( fxc <= f(n+1) ) then
                xi(1:n,n+1) = xc(1:n);  fi(n+1) = fxc
                cycle
            else                                 ! reduction
                 do i = 2, n+1
                    xi(1:n,i) = x(1:n,1)+sig*(x(1:n,i)-x(1:n,1))
                    fi(i) = func(xi)
                end do
                cycle
            end if

        end if

    end do

    xstart(1:n)=xi(1:n,1); fstart = fi(1)

end subroutine downhill
```

2.2 Sum Approximation

We analyzed the function

$$\zeta(s) = \sum_{n=1}^{\infty} n^{-s}$$

The sum $\zeta(1)$ diverges, so we used other local convergent s values to make an approximation for $\zeta(1)$ by means of a continuous fraction algorithm.

We used s values from 2 to 4 and calculated each corresponding $\zeta(s)$ sum. The s and $\zeta(s)$ values were input into the continuos fraction algorithm to then evaluate $\zeta(1)$.

The sums were calculated as follows:

```
program prob_2

use numtype
use thiele_approx
implicit none

integer, parameter :: np = 11
integer :: inmax, in, i
real(dp), dimension(1:np) :: sums, xx, yy
real(dp) :: x


inmax = 10**7
sums = 0



do in = 1,inmax                                  !itereate sums for function chi(s)
    sums(1) = sums(1) + 1/(in**2.0_dp)      !chi(s) = sum from 1 to inf. of n^-s
    sums(2) = sums(2) + 1/(in**2.2_dp)      !at various values of s
    sums(3) = sums(3) + 1/(in**2.4_dp)
    sums(4) = sums(4) + 1/(in**2.6_dp)
    sums(5) = sums(5) + 1/(in**2.8_dp)
    sums(6) = sums(6) + 1/(in**3.0_dp)
    sums(7) = sums(7) + 1/(in**3.2_dp)
    sums(8) = sums(8) + 1/(in**3.4_dp)
    sums(9) = sums(9) + 1/(in**3.6_dp)
    sums(10) = sums(10) + 1/(in**3.8_dp)
    sums(11) = sums(11) + 1/(in**4.0_dp)
```

```
    end do

    xx(1) = 2.0_dp                                   !chosen values for s to analyze sum
    xx(2) = 2.2_dp                                   !to input into thiele cf approximation
    xx(3) = 2.4_dp
    xx(4) = 2.6_dp
    xx(5) = 2.8_dp
    xx(6) = 3.0_dp
    xx(7) = 3.2_dp
    xx(8) = 3.4_dp
    xx(9) = 3.6_dp
    xx(10) = 3.8_dp
    xx(11) = 4.0_dp

    yy(1) = sums(1)                                  !resulting sums of corresponding s values
    yy(2) = sums(2)                                  !yy is a function of xx
    yy(3) = sums(3)
    yy(4) = sums(4)
    yy(5) = sums(5)
    yy(6) = sums(6)
    yy(7) = sums(7)
    yy(8) = sums(8)
    yy(9) = sums(9)
    yy(10) = sums(10)
    yy(11) = sums(11)

    write(10,*) xx, yy                               !make sure s values and partial sums are g

    call thiele_coef( np, xx, yy, an )               !generate continued fraction coefficients

    x = 1.0_dp

    print *, x, thiele_cf (x, np, xx, an)            !use cf coeffs to evalutae the function at



end program
```

The continued fraction method used was the Thiele method and this was the code used:

```
    module thiele_approx

use numtype
implicit none
integer, parameter :: maxpt = 50
    real(dp), dimension(maxpt) :: zn, fn, an

contains

        subroutine thiele_coef( nn, zn, fn, an )
        ! coefficients of Thiele continued fraction

            use numtype
            implicit none
            real(dp), dimension(maxpt) :: zn, fn, an
            real(dp), dimension(maxpt,maxpt) :: gn
            integer :: nn, n, nz

            gn(1,1:nn) = fn(1:nn)
            do n = 2, nn
                do nz = n, nn
                    gn(n,nz) = ( gn(n-1,n-1) - gn(n-1,nz) )/ &
                        ( (zn(nz)-zn(n-1) ) * gn(n-1,nz) )
                end do
            end do
            forall ( n = 1:nn ) an(n) = gn(n,n)

        end subroutine thiele_coef

        function thiele_cf (z, nn, zn, an)  result(cfrac)
        ! evaluate the Thiele continued fraction

            use numtype
            implicit none
            real(dp) :: z
            real(dp), dimension(maxpt) :: zn, an
            integer :: nn, n
            real(dp) :: cf0(2), cf1(2), cf(2), cfrac

            cf0(1) = 0._dp; cf0(2) = 1._dp
```

```
            cf1(1) = an(1); cf1(2) = 1._dp
            do n = 1, nn-1
                cf = cf1 + (z - zn(n)) * an(n+1) * cf0
                cf0 = cf1;  cf1 = cf
            end do
            cfrac = cf(1)/cf(2)

        end function thiele_cf

end module thiele_approx
```

3 Results

3.1 Harmonic Oscillator Fit

With x ranging from 1 to 6 the resulting Chi^2 value was 0.1650 and the plots are shown in figure 2.

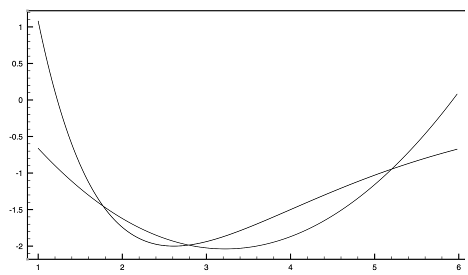

Figure 2: $U(x)$ fit to $V(x)$

To get a better fit, the range of x would need to be shrunk, closer to the absolute minimum of $V(x)$.

3.2 Sum Approximation

Using 10^8 terms in each sum and eleven pairs of s and $\zeta(s)$ values, $\zeta(1)$ was approximated to be

$$\zeta(1) = 542548.10289$$