Homework 3, Problem 1 (solutions)

Theory/Introduction

Diffraction problems often involve the Fresnel sine and cosine integrals C(u) and S(u). For this problem the diffraction intensity is given by

$I=I_0/8([2C(u)+1]^2+[2S(u)+1]^2)$

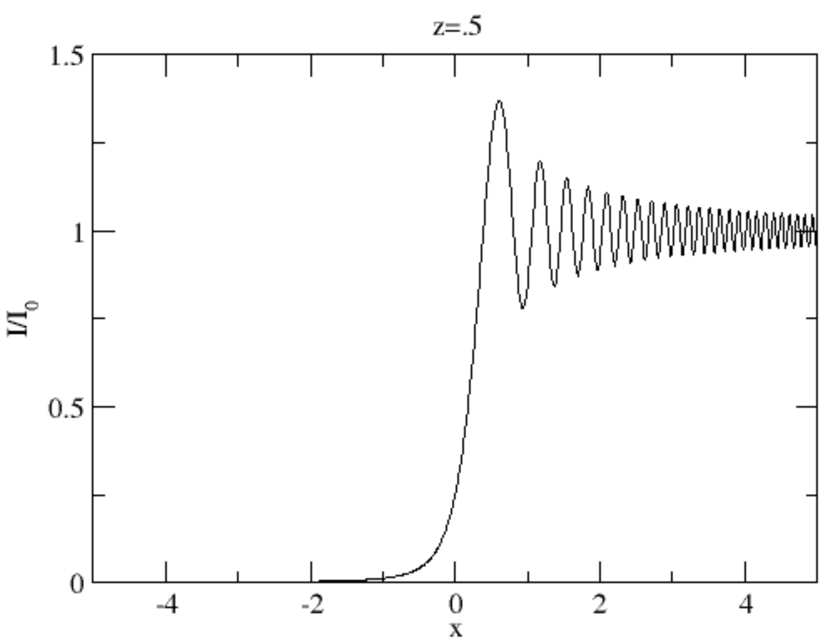
C(u) and S(u) are the integrals of $\cos(1/2 \text{ pi } t^2)$ and $\sin(1/2 \text{ pi } t^2)$, respectively.

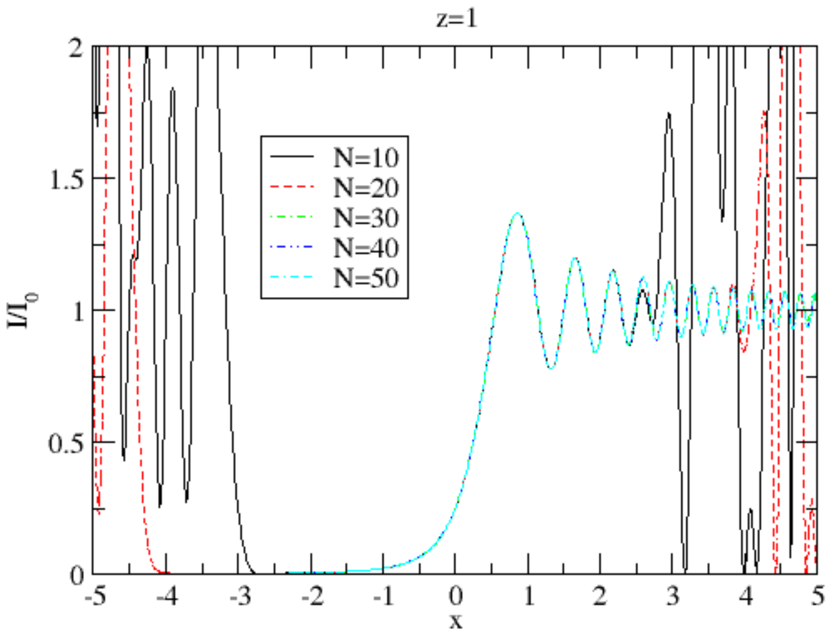
Code

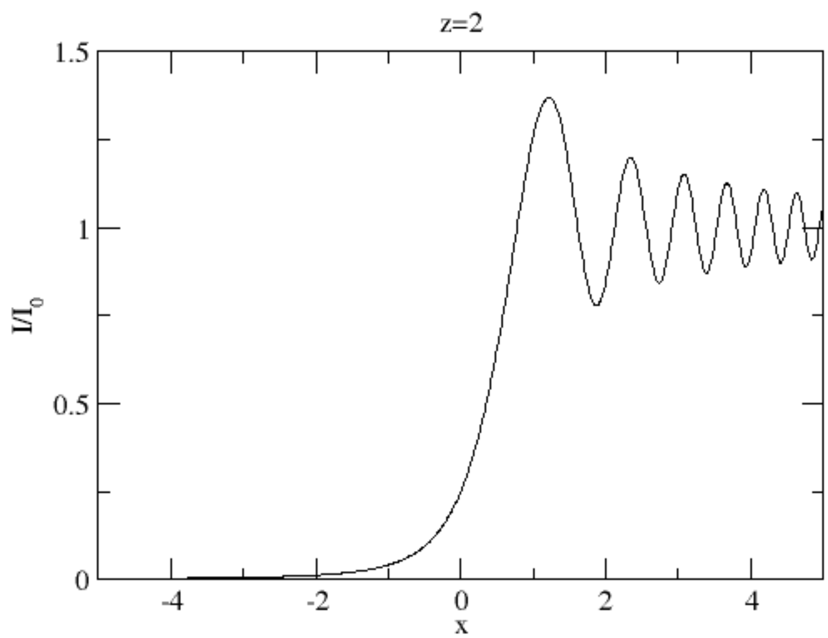
The integrals are done using the Gauss-Legendre quadrature from the textbook

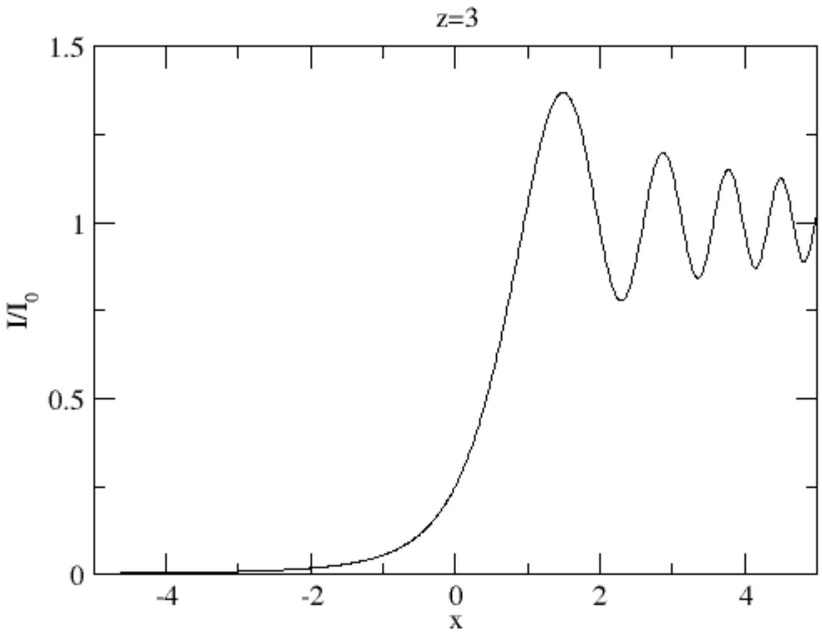
diffract.f90

Results




z=1
N=10
N=20
N=30
N=40
N=50
I/I0
x
2
1.5
1
0.5
0
-5
-4
-3
-2
-1
0
1
2
3
4
5


z=2
1.5
1
0.5
0
I/I0
-4
-2
0
2
4
x



Discussion/Summary

For the z=1 case I plotted the results for different N, the number of quadrature points. The most difficult points to calculate are when the upper limit of the integrals is large. When the upper limit is large, there are regions where the integrands oscillate rapidly, which requires more integration points to calculate the integrals accurately. Fpr z=1, a N of about 40 was needed for 6 digits of accuracy in the worst case (large x region). Small z is harder because it also makes the upper limit u larger.

```fortran
! HW 3 problem 1: Fresnel diffraction
!
!
program diffract
  implicit none

  integer,parameter :: n = 50   ! order of Gaussian quadrature
  real(kind=8),parameter :: pi=3.14159265358979323846264338328_8

  real(kind=8) :: x,z,dx,intens,u

  z=2.0d0
  x=-5.0d0
  dx=0.01d0
  do while (x<(5.0d0+dx))
     u=x*sqrt(2.0d0/z)
     intens=1/8.0d0*( (2.0d0*fcos(u)+1.0d0)**2 + (2*fsin(u)+1.0d0)**2)
     print *,x,intens
     x=x+dx
  enddo

contains

  function f(x)
    real(kind=8) :: f, x

    f=exp(-x)

  end function f

! Fresenel cosine integral
  function fcos(u)
    real(kind=8) :: u,fcos

    real(kind=8),dimension(n) :: w,x
    real(kind=8) :: quad
    integer :: i

    quad=0.0_8
    call gauss(n, 0, 0.0d0, u, x, w)
    do  i=1, n
       quad=quad+cos(x(i)*x(i)*pi*0.5d0)*w(i)
    end do
    fcos = quad
  end function fcos

! Fresenel sine integral
  function fsin(u)
    real(kind=8) :: u,fsin

    real(kind=8),dimension(n) :: w,x
    real(kind=8) ::  quad
    integer :: i

    quad=0.0_8
    call gauss(n, 0, 0.0d0, u, x, w)
    do  i=1, n
       quad=quad+sin(x(i)*x(i)*pi*0.5d0)*w(i)
    end do
    fsin = quad
  end function fsin


!
! gauss.f90: Points and weights for Gaussian quadrature
!       rescale rescales the gauss-legendre grid points and weights
!
!       npts             number of points
!       job =   0        rescaling uniformly between (a,b)
!               1        for integral (0,b) with 50% points inside (0, ab/(a+b))
!               2        for integral (a,inf) with 50% inside (a,b+2a)
!               x, w             output grid points and weights.
!
  subroutine gauss(npts,job,a,b,x,w)
    integer,intent(in) ::npts,job
    real(kind=8),intent(in) :: a,b
    real(kind=8),intent(out) :: x(npts),w(npts)

    real(kind=8) :: xi,t,t1,pp,p1,p2,p3,aj
    real(kind=8),parameter :: pi=3.14159265358979323846264338328_8
    real(kind=8),parameter :: eps=3.0e-16_8
    real(kind=8),parameter :: zero=0.0_8, one=1.0_8, two=2.0_8
    real(kind=8),parameter :: half=0.5_8, quarter=0.25_8
    integer :: m,i,j

    m=(npts+1)/2
    do  i=1,m
       t=cos(pi*(i-quarter)/(npts+half))
       do
          p1=one
          p2=zero
          aj=zero
          do j=1,npts
             p3=p2
             p2=p1
             aj=aj+one
             p1=((two*aj-one)*t*p2-(aj-one)*p3)/aj
          end do
          pp=npts*(t*p1-p2)/(t*t-one)
          t1=t
          t=t1-p1/pp
          if (abs(t-t1)<eps) exit
       enddo
       x(i)=-t
       x(npts+1-i)=t
       w(i)=two/((one-t*t)*pp*pp)
       w(npts+1-i)=w(i)
    end do

    !
    ! rescale the grid points
    !
    select case(job)
    case (0)
       ! scale to (a,b) uniformly
       do i=1,npts
          x(i)=x(i)*(b-a)/two+(b+a)/two
          w(i)=w(i)*(b-a)/two
       end do

    case(1)
       ! scale to (0,b) with 50% points inside (0,ab/(a+b))
       do i=1,npts
          xi=x(i)
          x(i)=a*b*(one+xi)/(b+a-(b-a)*xi)
          w(i)=w(i)*two*a*b*b/((b+a-(b-a)*xi)*(b+a-(b-a)*xi))
       end do

    case(2)
       ! scale to (a,inf) with 50% points inside (a,b+2a)
       do i=1,npts
          xi=x(i)
          x(i)=(b*xi+b+a+a)/(one-xi)
          w(i)=w(i)*two*(a+b)/((one-xi)*(one-xi))
       end do
    end select

  end subroutine gauss


end program diffract
```

Homework 3, Problem 2 (solutions)

Theory/Introduction

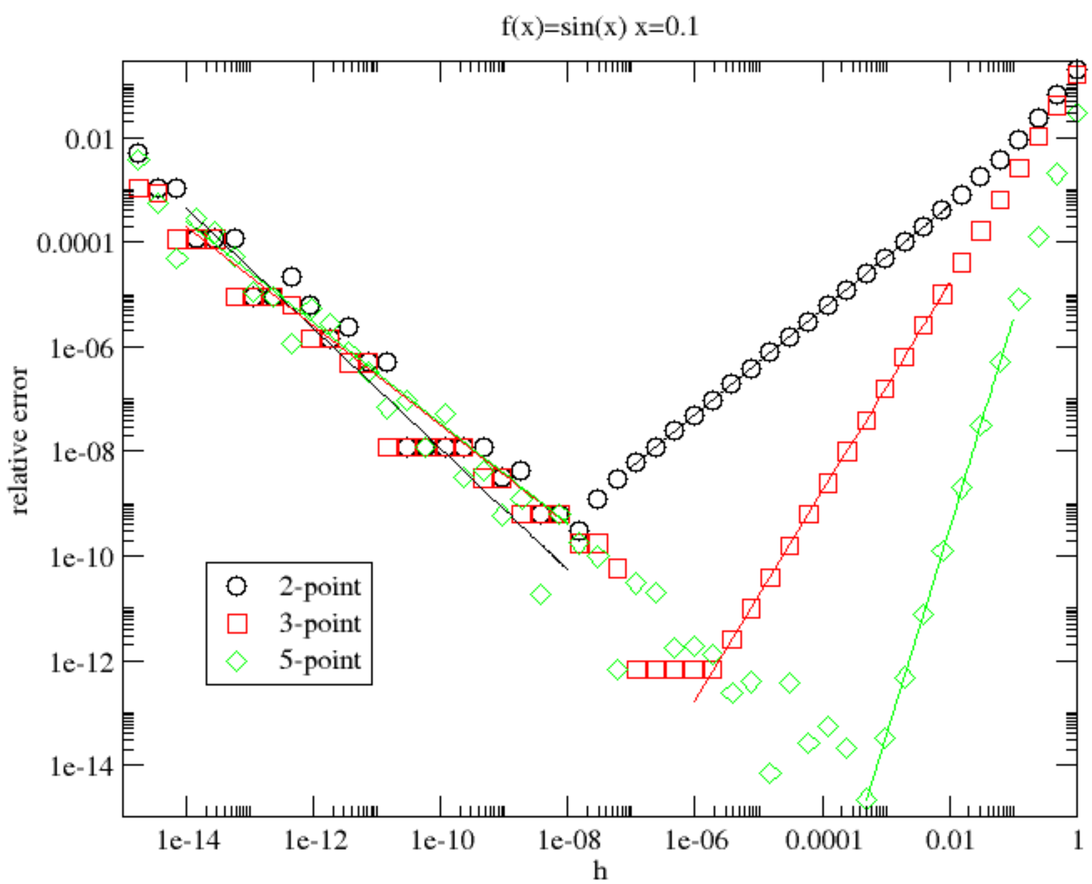
This problem compares the error behavior of three different formulas for the first derivative, the 2-point, 3-point, and 5-point formulas. The derivative formulas depend on a point spacing h. As h decreases, the approximation error decreases. However, for small enough h, subtraction in the formulas leads to roundoff error dominating over the approximation error.

Code

[2pt.f90]

Results

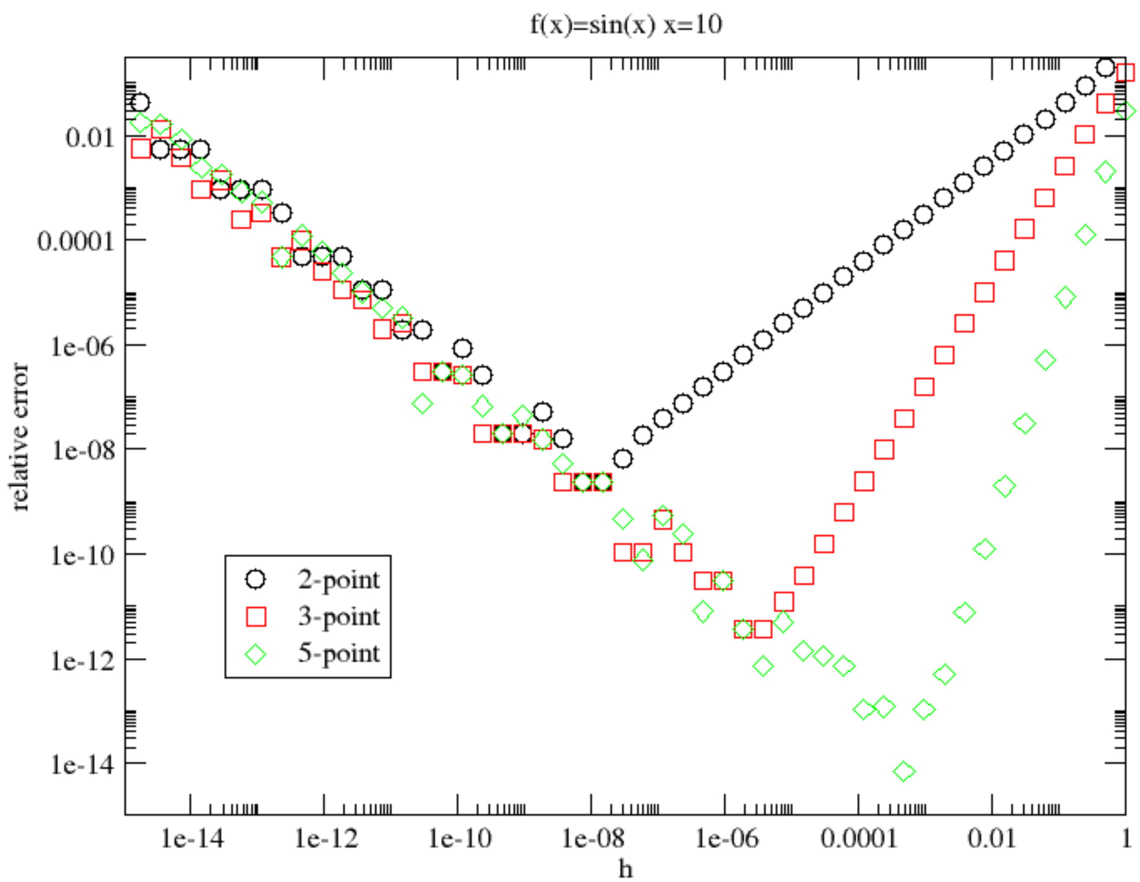
f(x)=sin(x), x=0.1



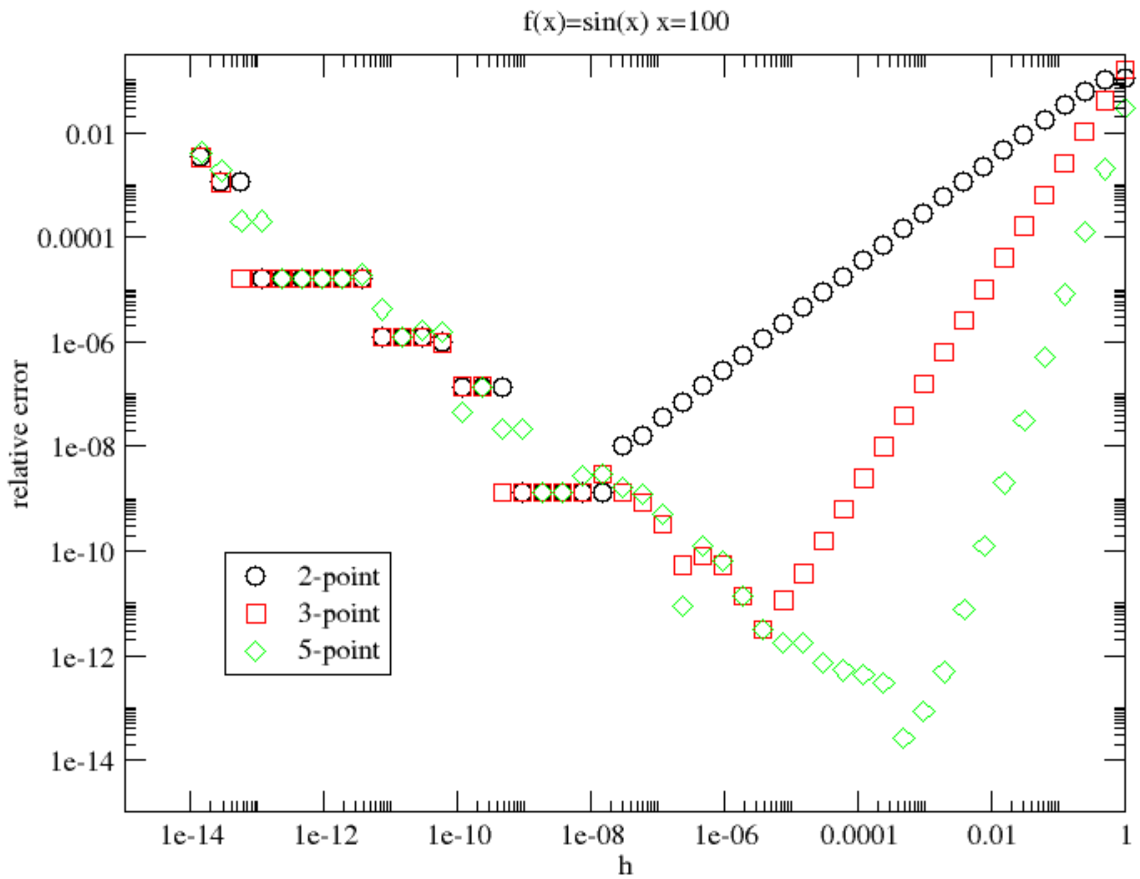
Fitting:

- 2-point, approximation error; slope = 1.0005, roundoff slope = -1.1(2); minimum error of about 4.0e-10 at h of about 1.0e-08
- 3-point, approximation error; slope = 1.9999, roundoff slope = -0.9(3); minimum error of about 8.0e-13 at h of about 2.0e-06
- 5-point, approximation error; slope = 3.9992, roundoff slope = -1.0(4); minimum error of about 3.0e-15 at h of about 0.0005

f(x)=sin(x), x=10.0

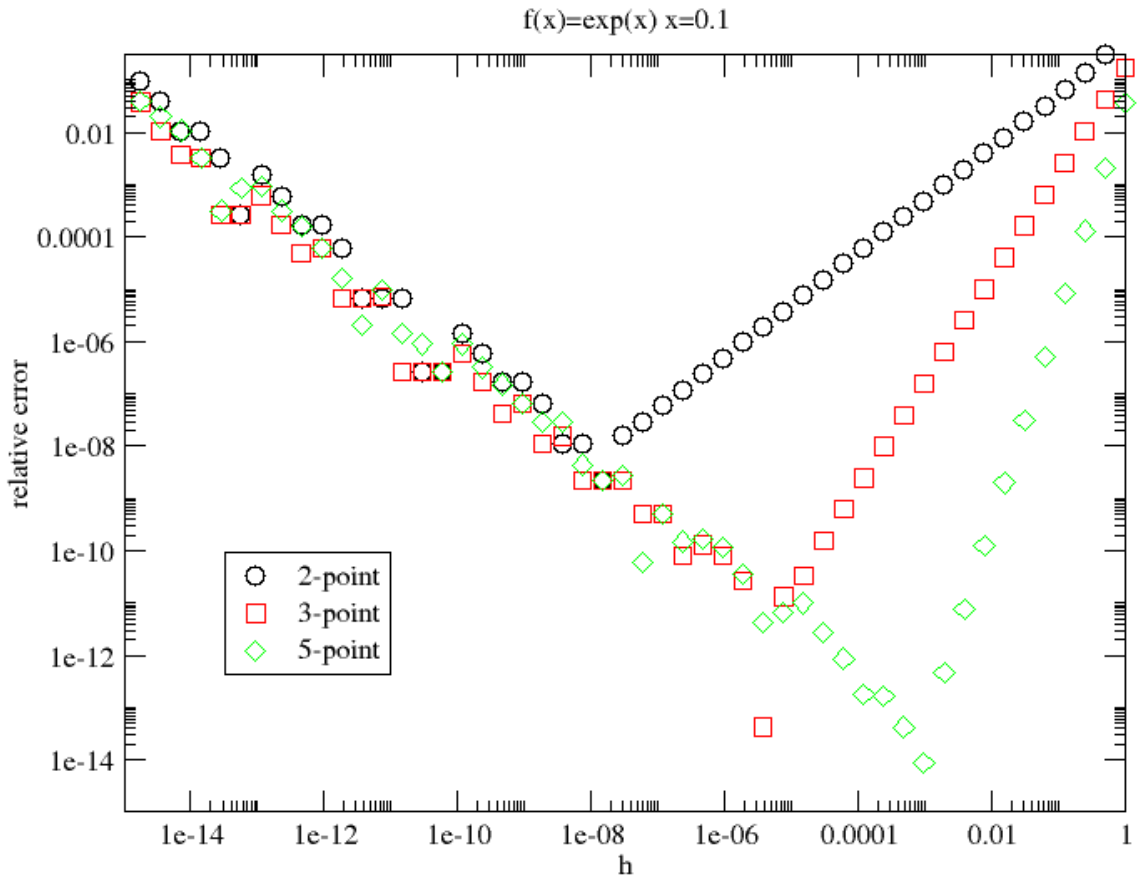


f(x)=sin(x), x=100.0

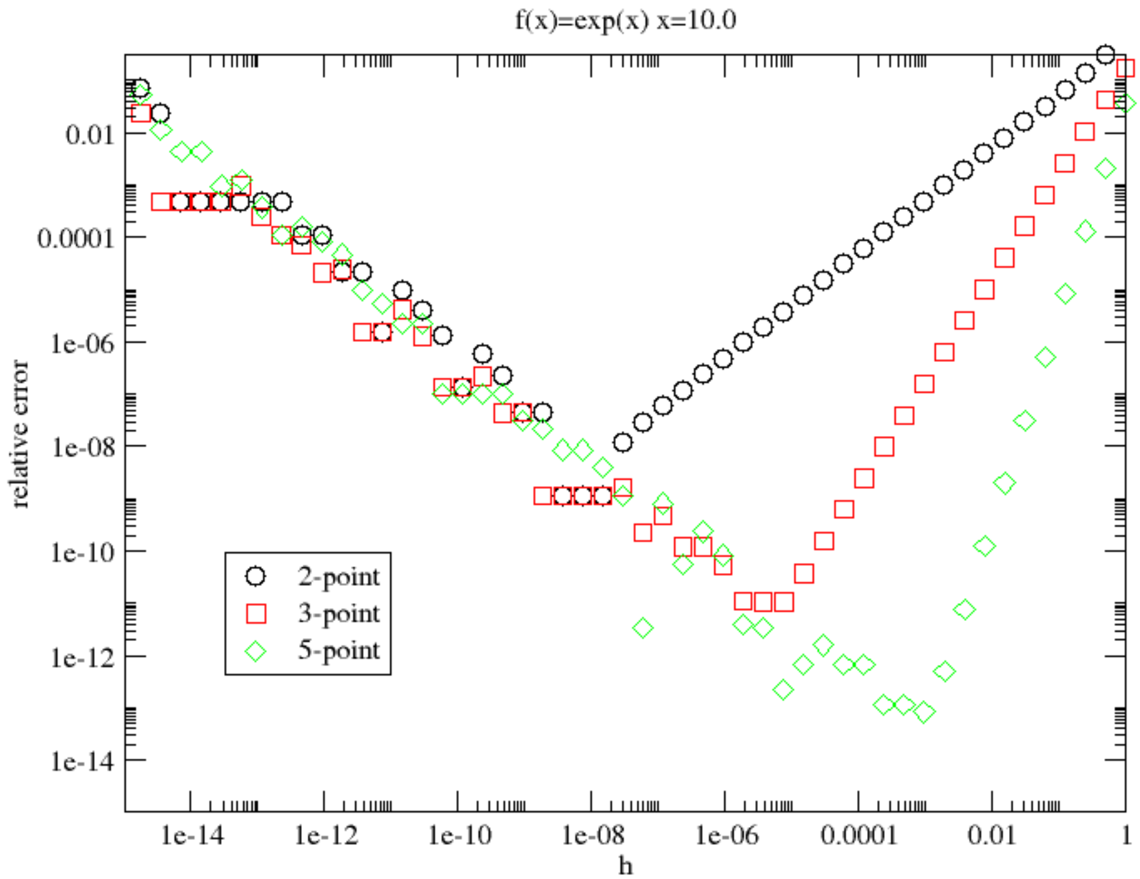


f(x)=sin(x) x=100
relative error
h
0.01
0.0001
1e-06
1e-08
1e-10
1e-12
1e-14
1e-14
1e-12
1e-10
1e-08
1e-06
0.0001
0.01
1
2-point
3-point
5-point

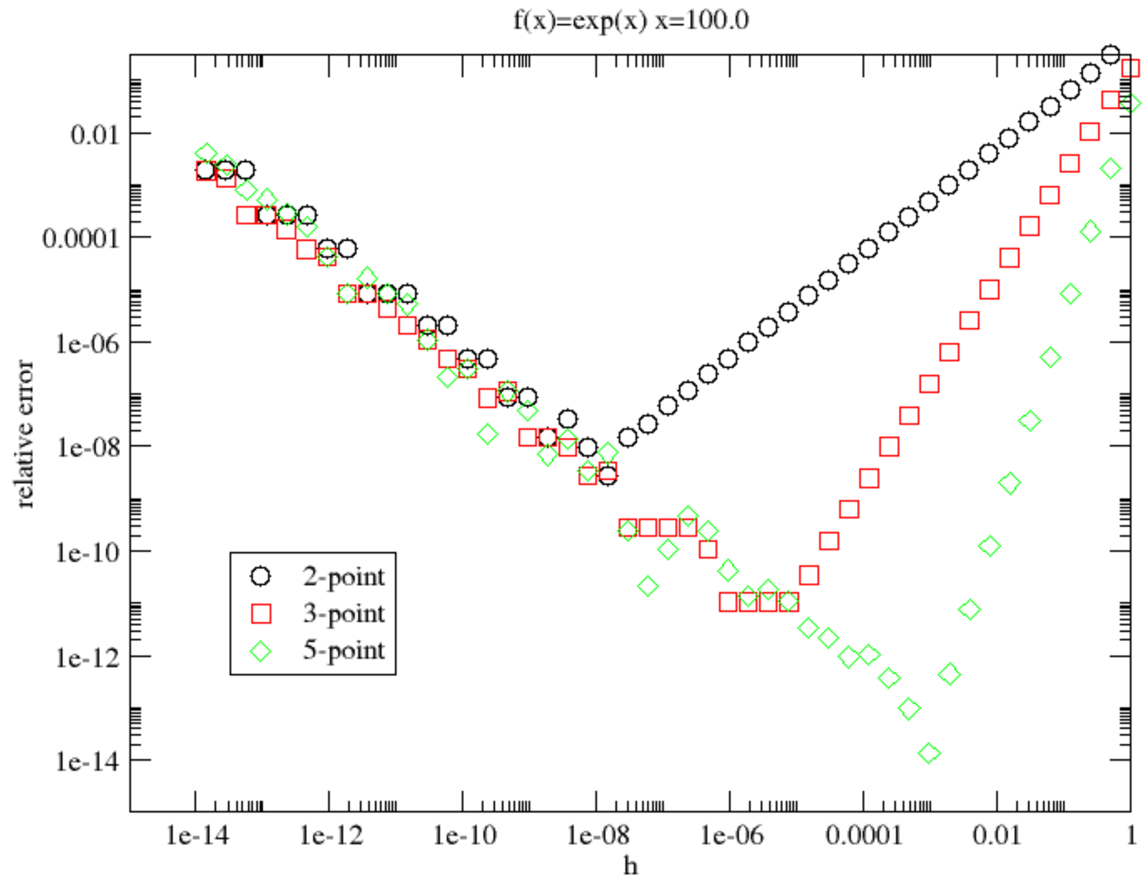
f(x)=exp(x), x=0.1


f(x)=exp(x) x=0.1
relative error
h
2-point
3-point
5-point
0.01
0.0001
1e-06
1e-08
1e-10
1e-12
1e-14
1e-12
1e-10
1e-08
1e-06
0.0001
0.01
1

f(x)=exp(x), x=10.0



f(x)=exp(x), x=100.0



Discussion

The actual approximation error dependence on h was very close to the prediction (slopes of 1, 2, and 4 for the 2, 3, and 5-point formulas). For all formulas the roundoff error slope was close to -1, with roundoff error beginning to dominate at larger h for the more accurate formulas. Because all formulas exhibit such similar roundoff error h dependence, this suggests that the roundoff error is coming from a similar cause- in this case probably subtraction.

The predicted minimum errors for the 2-point and 3-point formulas were 3e-08 and 3e-11, assuming double precision and that the next higher derivative is of order 1 (text equation 7.15). The h for these lowest errors was predicted to be 4e-08 and 3e-05, respectively. The actual minimum errir was somewhat less that this, while the h where the minimum error occured was consistent with this estimate. For the 5-point formula we did not derive a formula for the minimum error, but the results seem consistent, with the optimum h increasing by around two orders of magnitude.

Changing the value of x changes the relative error very little in these examples, because the derivative is of similar magnitude to the function itself. The absolute error for the derivative of exp(x) will of course be larger for large x.

```fortran
! prints error in 2-point, 3-point, and 5-point derivatice
! formulas as a function of h
!
! RT Clay 09/2015
!
program twopoint
  implicit none

  real(kind=8) :: h,x

  h=1.0d0
  x=100.0d0
  do while (h>1.0d-16)
     print *,h,abs((diff2pt(x,h)-exact(x))/exact(x)), &
          abs((diff3pt(x,h)-exact(x))/exact(x)),  &
          abs((diff5pt(x,h)-exact(x))/exact(x))
     h=h/2
  enddo

contains

  ! function to take derivative of
  function f(x)
    real(kind=8) :: f,x
    f=exp(x)
  end function f

  ! exact derivative of f(x)
  function exact(x)
    real(kind=8):: exact,x
    exact=exp(x)
  end  function exact


  ! 2-point derivative
  function diff2pt(x,h)
    real(kind=8) :: diff2pt,x,h

    diff2pt=(f(x+h)-f(x))/h
  end function diff2pt

  ! 3-point derivative
  function diff3pt(x,h)
    real(kind=8) :: diff3pt,x,h

    diff3pt=(f(x+h)-f(x-h))/(2.0d0*h)
  end function diff3pt

  ! 5-point derivative
  function diff5pt(x,h)
    real(kind=8) :: diff5pt,x,h

    diff5pt=(-f(x+2.0d0*h)+8.0d0*f(x+h)-8.0d0*f(x-h)+f(x-2.0d0*h))/(12.0d0*h)
  end function diff5pt


end program twopoint
```

Homework 3, Problem 3 (solutions)

Introduction

In this problem we solve an elliptic integral for the period of a plane pendulum. It is easy in this case to construct a Gaussian quadrature that is very accurate because the form of the integral is that of a Gauss-Chebyshev integral. In this case the weights and abscissas for the Gaussian quadrature are known analytically. In the integral the function f(x) and weight function w(x) are:

$f(x) = 2/\mathrm{sqrt}(1-k^2x^2)$

$w(x) = 1/\mathrm{sqrt}(1-x^2)$

Code

Simple code: fixed n

pendulum.f90

Code with automatic n

To determine the n needed automatically, compare the value of the integral for n and n+1. The relative error can be estimated from these two values.

pendulum2.f90

Results

Simple code, $theta_0$=0.1

The small-amplitude period is 2pi. With a small angle the difference from 2*pi is very small:

```
n   T                       T/(2*pi)
1   6.2831853071795862      1.0000000000000000
3   6.2871145483499413      1.0006253581548621
5   6.2871145493104796      1.0006253583077367
7   6.2871145493104796      1.0006253583077367
9   6.2871145493104805      1.0006253583077367
```

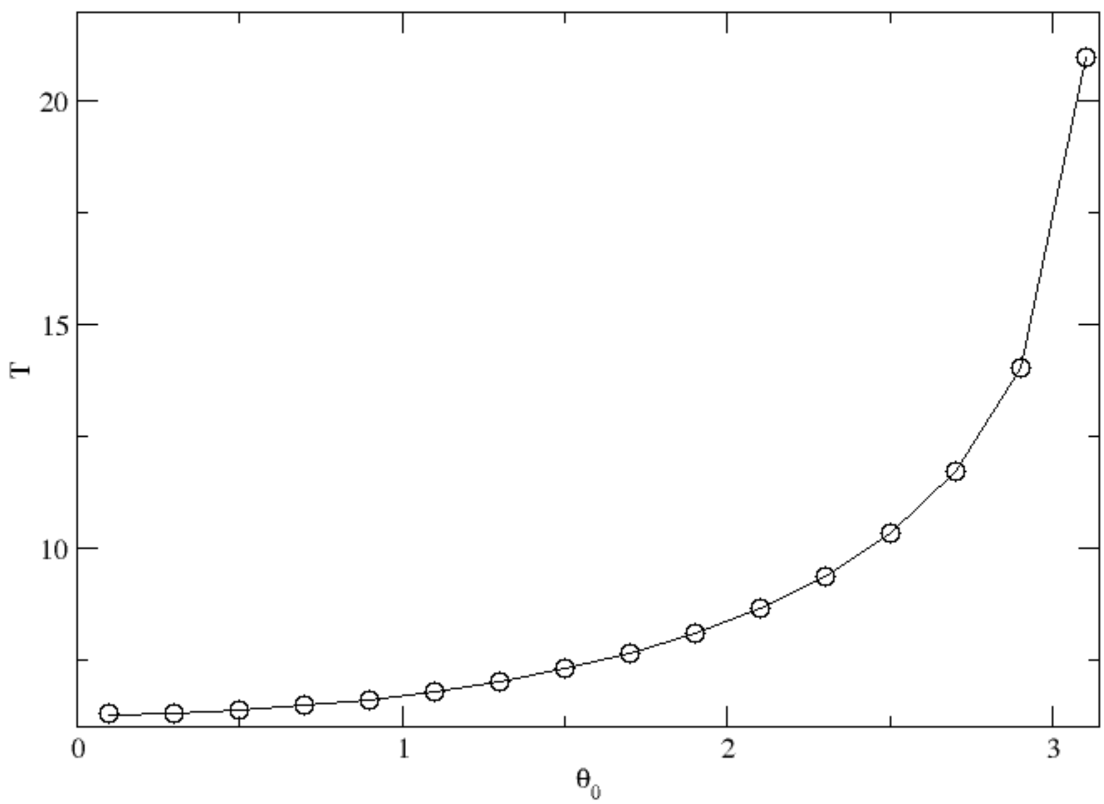
Code with automatic n

Assumes relative error less than 1.0e-06

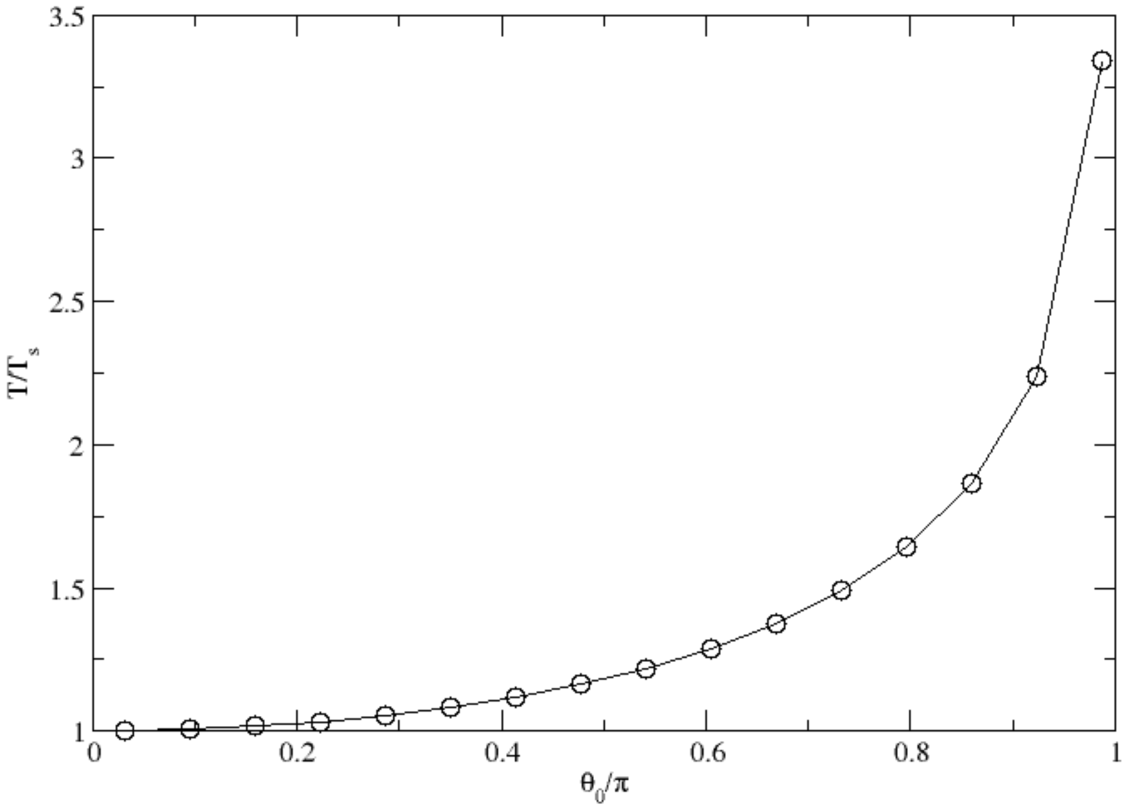
```
theta0 N     period          period/2pi
0.10  4      6.28711455      1.00062536
0.30  4      6.31871154      1.00565418
0.50  4      6.38278970      1.01585253
0.70  4      6.48118971      1.03151338
0.90  7      6.61686647      1.05310701
1.10  7      6.79416186      1.08132444
1.30  7      7.01925030      1.11714838
1.50  7      7.30086480      1.16196872
1.70 10      7.65151350      1.21777620
1.90 10      8.08961444      1.28750213
```

2.10	13	8.64351941	1.37565884
2.30	16	9.35983654	1.48966425
2.50	19	10.32315984	1.64298192
2.70	28	11.71347810	1.86425794
2.90	49	14.04611196	2.23550815
3.10	49	20.97181119	3.33776742

Period as a function of theta0:



With the period and angle normalized:



Discussion

Notice that with one integration point (n=1), the Gauss-Chebyshev formula returns the exact result for the period in the limit of small oscillations (T=2pi). As the initial angle approaches pi, the period diverges, and the integral becomes more and more difficult to compute numerically.

```fortran
! calculates the period of a pendulum with L/g = 1
!
!  RT Clay 09/2015
!
program pendulum
  implicit none

  real(kind=8),parameter :: pi=4.0d0*atan(1.0d0)
  real(kind=8) :: theta0,k2,t
  integer :: i

  theta0=0.1d0
  k2=sin(theta0*0.5d0)**2
  do i=1,10,2
     t=integ(k2,i)
     print *,i,t,t/(2.0d0*pi)
  enddo

contains

  ! integration function
  function f(x,k2)
    real(kind=8) :: f,x,k2
    f=2.0d0/sqrt(1.0d0-k2*x*x)
  end function f

  !Gauss-Chebshev integration
  function integ(k2,n)
    real(kind=8) :: k2,integ
    integer :: n,i
    real(kind=8) :: x,w,sum

    sum=0.0d0
    w=pi/n
    do i=1,n
       x=cos(pi*(real(i)-0.5d0)/n)
       sum=sum+f(x,k2)*w
    enddo
    integ=sum

  end function integ

end program pendulum
```

```fortran
! calculates the period of a pendulum with L/g = 1
!
!  RT Clay 09/2015
!
program pendulum
  implicit none

  real(kind=8),parameter :: pi=4.0d0*atan(1.0d0)
  real(kind=8) :: theta0,k2,t,t2
  integer :: i,n

  theta0=0.1d0
  do while (theta0<pi)
     k2=sin(theta0*0.5d0)**2

     do i=1,100,3
        t=integ(k2,i)
        t2=integ(k2,i+2)
        if (abs((t-t2)/t2)<1.0d-6) then
           n=i
           exit
        endif
     enddo
     print "(F6.2,i3,f16.8,f16.8)",theta0,n,t2,t2/(2.0d0*pi)

     theta0=theta0+0.2d0
  enddo

contains

  ! integration function
  function f(x,k2)
    real(kind=8) :: f,x,k2
    f=2.0d0/sqrt(1.0d0-k2*x*x)
  end function f

  !Gauss-Chebshev integration
  function integ(k2,n)
    real(kind=8) :: k2,integ
    integer :: n,i
    real(kind=8) :: x,w,sum

    sum=0.0d0
    w=pi/n
    do i=1,n
       x=cos(pi*(real(i)-0.5d0)/n)
       sum=sum+f(x,k2)*w
    enddo
    integ=sum

  end function integ

end program pendulum
```