1 Reviews

1. What are the differences between functions and subroutines?
2. How are arguments passed to procedures in FORTRAN?
3. What does the term “dummy argument” refer to and contrast it to an “actual argument”?
4. Explain why dummy and actual arguments have to agree on position and data type.
5. Explain what an interface block is and why it is useful in programming.
6. What are the differences between an internal procedure and an external procedure?

2 Coordinate transformation

The transformations between cartesian coordinates \((x, y)\) and polar coordinates \((r, \theta)\) are

\[
\begin{align*}
  r &= \sqrt{x^2 + y^2} \\
  \theta &= \tan^{-1} \frac{y}{x}
\end{align*}
\]

and its inverse \[
\begin{align*}
  x &= r \cos \theta \\
  y &= r \sin \theta
\end{align*}
\] (1)

Write a module with subroutines \texttt{cart2pol}(r,t,x,y) and \texttt{pol2cart}(x,y,r,t) that transform cartesian coordinates \(x,y\) into polar coordinates \((r, \theta)\) and vice versa. Test these subroutines on the 4 points \((\pm 1, \pm 1)/\sqrt{2}\), use the inverse transformation to recover the original points. Use the intrinsic function \texttt{atan2} to find the angle which should be reported in degrees.

3 Function and derivative evaluation

Write a module with a subroutine \texttt{disp}(f,df,x) that calculates the function \(f(x) = x \tanh x - 1/4\) and its derivative \(f'(x) = \tanh x + x(1 - \tanh^2 x)\) given \(x\). Use this subroutine to calculate these functions on an equally spaced grid between -5 and 5, and plot the two functions \(f\) and \(f'\) using matlab/octave. Calculate the \texttt{tanh} function once to save computational time. To simplify the plotting print out the results in a three-column format:

\[
\begin{array}{ccc}
  x & f & df \\
\end{array}
\]
4 Trapezoidal integration module

By the end of this exercise you should have a module called quadrature that can estimate an integral in a variety of ways. The starting point is the vanilla version of the trapezoid function that can be used without prior estimate. Refer to figure 1 for a graphical illustration. Using \( h_N \) and \( h_{2N} \) for the grid spacing for the \( N \) and \( 2N \) estimates, respectively, we have \( h_{2N} = h_N / 2 = (b-a)/(2N) \); The two trapezoidal estimates can now be written as

\[
S_N = \frac{b - a}{N} \left[ \frac{f(a) + f(b)}{2} + \sum_{i=1}^{N-1} f(a + ih_N) \right] \tag{2}
\]

\[
S_{2N} = \frac{b - a}{2N} \left[ \frac{f(a) + f(b)}{2} + \sum_{i=1}^{2N-1} f(a + ih_{2N}) \right] \tag{3}
\]

Forming the term \( S_{2N} - S_N / 2 \) from equations (2) and (3) we have:

\[
S_{2N} = \frac{S_N}{2} + \frac{b - a}{2N} \sum_{i=1,3,5,...}^{2N-1} f(ih_{2N} + a) \tag{4}
\]

Equation (4) expresses the new estimate in terms of the old ones without having to evaluate the function multiple times at the same points; the new evaluations are needed only at the half way points, in other words only the odd indeces from \( i = 1, 3, \ldots, 2N - 1 \) are required. Implement a function that uses equations (4) to calculate the improved estimate and incorporate it in the adaptive routine.

- The first requirement is to create a module that contains the trapezoidal function, and to modify the main program to use the quadrature module to perform the integration. Verify that both codes work by integrating the function \( \cos^2 \pi x \) between 0 and 1. You will have to make \texttt{myfunc} an external function for the executable to build properly. There is a skeleton code available to get you started copy, make sure it works, and start modifying it:

\[
\$ \text{cp -rp } \sim \text{mohamed/Quads .}
\]

\[
\$ \text{make quadmain}
\]

\[
\$ \sim ./quadmain
\]
There is a README file to get you started.

- Write a second version of the trapezoid function, trapezoid2, that avoids computing the functions twice over the same points when the number of points is doubled.
- Test your code by integrating the function \( \cos^4 \pi x \) between 0 and 1 (the answer should be \( 3/8 \)), and use a tolerance of \( 10^{-6} \) starting with a single interval. Monitor the decrease of the error \( |S_N - 3/8| \) as a function of \( N \) and confirm that each doubling decreases the error by a factor of 4.
- The Simpson integration rule uses parabola to approximate the function \( f \) instead of straight lines; it can be coded with little extra efforts by recycling the results of 2 trapezoidal approximations with \( N \) and \( 2N \) points and combining them as:
  \[
  S = \frac{4S_{2N} - S_N}{3}
  \]

Write a function \texttt{Simpson(a,b,N,myfunc)} that performs these tasks. Try it on the same function as before and confirm numerically its convergence rate. How many points does the Simpson quadrature require to achieve the same accuracy, \( 10^{-6} \), as the trapezoidal rule?

### A \hspace{1cm} \texttt{atan2}

\texttt{atan2(y,x)} computes the arctangent of the with cartesian coordinates \((x,y)\). The following caveats hold

- The type of \( y \) must be \texttt{REAL(*)}.
- The type and kind type parameter of \( x \) shall be the same as \( y \). If \( y \) is zero, then \( x \) must be nonzero.
- The return value has the same type and kind type parameter as \( y \). If \( x \) is nonzero, then it lies in the range \(-\pi \leq \arccos(x) \leq \pi\). The sign is positive if \( y \) is positive. If \( y \) is zero, then the return value is zero if \( x \) is positive and \( \pi \) if \( x \) is negative. Finally, if \( x \) is zero, then the magnitude of the result is \( \pi/2 \).
- Example:

```fortran
program test_atan2
    real :: x = 1.e0_4, y = 0.5e0_4
    x = atan2(y,x)
end program test_atan2
```