Worksheet 3. Datatypes and Control Structures.

Error, accuracy and stability

Associativity To look at the effect of truncation error type the following into Matlab

```
x=logspace(-12,-17,100);
y=((1+x)-1)./x;
% Plot with logarithmic x-axis, linear y-axis
semilogx(x,y,'r+',x,ones(size(x)),'b-')
```

If the computer had infinite space to store numbers the point and line would coincide.

Quadratic formula Standard quadratic formula

$$ax^2 + bx + c = 0 \tag{1}$$

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{2}$$

Analytically correct, but susceptible to roundoff error. Instead use

$$q = -\frac{1}{2}\left(b + \text{sgn}(b)\sqrt{b^2 - 4ac}\right) \tag{3}$$

$$x_1 = \frac{q}{a} \qquad x_2 = \frac{c}{q} \tag{4}$$

To compare the two

```
a=logspace(-5,-9,100);
c=a;
b=ones(size(a));
% Root: warning susceptible to roundoff error
root_a=(-b+sqrt(b.^2-4*a.*c))./(2*a);
% Roundoff adjusted formula
q=-0.5*(b+sign(b).*sqrt(b.^2-4*a.*c));
root_b=c./q;
% Plot results, logarithmic x- and y-axes
loglog(-root_b,-root_a,'r+',-root_b,-root_b,'b-')
```

Stability Try to use the recurrence relation to calculate powers of the golden ratio $\phi = \dfrac{\sqrt{5}-1}{2}$

$$\phi^n = \phi^{n-2} - \phi^{n-1} \tag{5}$$

```
n=50;
phi=(sqrt(5)-1)/2;
phi1=1;
phi2=phi;
for i=2:n
  phi3=phi1-phi2;
  phi1=phi2; phi2=phi3;
  fprintf('%8d %18.6e %18.6e\n',i,phi3,phi^i)
end
```

Problem is that recurrence relation is also solved by $-\dfrac{\sqrt{5}+1}{2}$.

Control structures and program units

To illustrate the use of these we will look at a typical case of using Matlab and Fortran together. We develop and test an algorithm using Matlab: its interactive nature and visualisation characteristics make it ideal for rapid prototyping.

However sometimes a matlab implementation is too slow and so we need to translate the algorithm we have developed into Fortran.

As our example we take the case of finding the root of an equation by the bisection method. We assume the root has already been bisected.
Function (put in file called func.m)

```
function y=func(x)
% FUNC function for testing
  y=10*(x-10);
end
```

Bisection routine (put in a file called bisect.m)

```
function xroot=bisect(f,x1,x2,tol)
% BISECT find bracketed root by bisection method
% with visualisation of root finding process
% OUTPUTS
%   xroot best guess of root location
% INPUTS
%   func function whose root is to be found
```

```
%   x1,x2 locations bracketing root
%   tol tolerance of root location

% check root is bracketed
y1=feval(f,x1); y2=feval(f,x2);
if(y1*y2>0)
  error('Root not bracketed')
end

% make sure y1<0<y2
if(y1>0)
  x3=x1; y3=y1;
  x1=x2; y1=y2;
  x2=x3; y2=y3;
end

% store these
x10=x1; x20=x2;

while abs(x1-x2)>tol
  x3=0.5*(x1+x2); y3=feval(f,x3);
  if(y3>0)
    x2=x3; y2=y3;
  else
    x1=x3; y1=y3;
  end
  fplot(@func,[x10,x20],'b-')
  hold on
  plot([x10,x20],[0,0],'b--')
  plot([x1,x2],[y1,y2],'r+','markersize',20)
  s=sprintf('|x1-x2|=%.3e',abs(x2-x1));
  title(s)
  hold off
  pause(1)
end
xroot=0.5*(x1+x2);

end
```

To find the root type

```
octave:51> xroot=bisect(@func,-20,20)
```

`@func` is a function handle - it allows functions to be passed as arguments to functions.

Once we are happy with the algorithm (and have tried our best to break it) we can produce a Fortran implementation. (This can all go in one file or in three files.)

```
PROGRAM rootfind
  IMPLICIT NONE
  INTERFACE ! Interfaces are used to describe the properties of
            ! functions used by the program
     FUNCTION func(x)
       REAL(kind=kind(0d0)) :: func
       REAL(kind=kind(0d0)), INTENT(in):: x
     END FUNCTION func
     FUNCTION bisect(f,x1,x2,tol)
       REAL(kind=kind(0d0)) :: bisect
       REAL(kind=kind(0d0)), INTENT(out) :: x1,x2
       REAL(kind=kind(0d0)), INTENT(in) :: tol
       INTERFACE
          FUNCTION f(x)
            REAL(kind=kind(0d0)) :: f
            REAL(kind=kind(0d0)), INTENT(in):: x
          END FUNCTION f
       END INTERFACE
     END FUNCTION bisect
  END INTERFACE
 REAL(kind=kind(0d0)) :: x1,x2,tol, xroot
  x1=-100
  x2=100
  tol=1e-4
  xroot=bisect(func,x1,x2,tol)
  PRINT *, "The root is at x=",xroot
END PROGRAM rootfind



! This is the function whose root we will be finding
FUNCTION func(x)
  IMPLICIT NONE
```

```
  REAL(kind=kind(0d0)) :: func
  REAL(kind=kind(0d0)), INTENT(in) :: x
  func=10*(x-sqrt(2e0))
END FUNCTION func


! This function implements the bisection method
FUNCTION bisect(f,x1,x2,tol)
  IMPLICIT NONE
  REAL(kind=kind(0d0)) :: bisect
  REAL(kind=kind(0d0)), INTENT(out) :: x1,x2
  REAL(kind=kind(0d0)), INTENT(in) :: tol
  INTERFACE
     FUNCTION f(x)
       REAL(kind=kind(0d0)) :: f
       REAL(kind=kind(0d0)), INTENT(in):: x
     END FUNCTION f
  END INTERFACE
  REAL(kind=kind(0d0)) :: x3,y1,y2,y3
  INTEGER :: i
  INTEGER, PARAMETER :: maxiterations=50
  ! Check root is bracketed
  y1=f(x1); y2=f(x2)
  IF(y1*y2.GT.0) STOP "ERROR in bisect: root not bracketed"

  ! make sure y1<0<y2
  IF(y1.GT.y2) THEN
     y3=y1; y1=y2; y2=y3
     x3=x1; x1=x2; x2=x3
  END IF

  ! perform the bisection iterations
  DO i=1,maxiterations
     x3=0.5*(x1+x2); y3=f(x3)
     IF(y3>0e0) THEN
        x2=x3; y2=y3
     ELSE
        x1=x3; y1=y3
     END IF
     ! Have we found the root?
     IF(abs(x1-x2).LT.TOL) THEN
```

```
        bisect=0.5*(x1+x2); RETURN
     END IF
  END DO
  STOP "ERROR in bisect maximum iterations exceeded."
END FUNCTION bisect
```