Programowanie współbieżne

Laboratorium - 7b: Fortran 95

Andrzej Baran
baran@kft.umcs.lublin.pl

Pochodne 1, 2 I

Fortran 95

- Wyprowadź formuły różnicowe na pierwsze i drugie pochodne funkcji jednej zmiennej. Przyjmij stały krok, równy Δx.
- Przedstawiony niżej program oblicza pierwszą i drugą pochodna funkcji jednej zmiennej $f(x)$. Wykonaj testy programu używając różnych funkcji, porównując wartości numeryczne pochodnych z dokładnymi. Obliczenia wykonaj w pojedynczej i podwójnej precyzji

---

```
!
! Derivatives 1, 2 in 1-dim
!
module derivatives

CONTAINS

  subroutine FirstDeriv_1d(npts, dx, u, u_x)
    implicit none
    integer, intent(in) :: npts
    real, intent(in) :: dx
    real, dimension(:), intent(in) :: u
    real, dimension(:), intent(out) :: u_x
    real two_invdx
    integer i
    two_invdx = 1d0/(2d0*dx)
    ! central...
    do i=2,npts-1
```

Pochodne 1, 2 II

Fortran 95

```
      u_x(i)=(u(i+1)-u(i-1))*two_invdx
    end do
    ! forward...
    u_x(1)=(-3.*u(1)+4.*u(2)-u(3))*two_invdx
    ! backward
    u_x(npts)=(3.*u(npts)-4.*u(npts-1)+u(npts-2))*two_invdx
  end subroutine FirstDeriv_1d

  subroutine SecondDeriv_1d(npts, dx, u, u_xx)
    implicit none
    integer, intent(in) :: npts
    real, intent(in) :: dx
    real, dimension(:), intent(in) :: u
    real, dimension(:), intent(out) :: u_xx
    real inv_dx2
    integer i
    inv_dx2=1d0/(dx*dx)
    ! forward...
    u_xx(1)=(2.*u(1)-5.*u(2)+4.*u(3)-u(4))*inv_dx2
    ! central...
    do i=2,npts-1
      u_xx(i)=(u(i+1)-2.*u(i)+u(i-1))*inv_dx2
    end do
    ! backward
    u_xx(npts)=(2.*u(npts)-5.*u(npts-1)+4.*u(npts-2)-u(npts-3))*inv_dx2
  end subroutine SecondDeriv_1d

end module derivatives
```

Pochodne 1, 2 III

Fortran 95

```
!    Test of Derivatives routines

program TestDerivs
  use derivatives
  implicit none
  real func, func_x, func_xx
  integer, parameter :: levels=10   ! liczba poziomow do testowania
  real, parameter :: a=0d0          ! dolna granica przedzialu
  real, parameter :: b=1d0          ! gorna granica przedzialu
  real, dimension(:), allocatable :: u, u_x, u_xx
  integer i, j, npts, iall
  real dx, ux_error, uxx_error

  write(*,'(a)') "    npts     Error(1_Der)        Error(2_Der)"
  do i=2,levels+1,1
     npts = 2**i             ! liczba punktów rowna jest 2^level
     dx = 1d0/(npts-1)       ! ustaw dx na podstawie liczby punktow

     ! rezerwuj pamiec, dynamicznie
     allocate(u   (npts), stat=iall)
     allocate(u_x (npts), stat=iall)
     allocate(u_xx(npts), stat=iall)

     ! function
     do j=1,npts
        u(j) = func((j-1)*dx)
     end do

     call FirstDeriv_1d(npts, dx, u, u_x)
```

Pochodne 1, 2 IV

Fortran 95

```
      call SecondDeriv_1d(npts, dx, u, u_xx)

      ! error
      ux_error  = 0d0
      uxx_error = 0d0
      do j=1,npts
         ux_error  = ux_error  + dx*(u_x (j)-func_x ((j-1)*dx))**2
         uxx_error = uxx_error + dx*(u_xx(j)-func_xx((j-1)*dx))**2
      end do

      write(*,'(i8, 2e20.9)') npts, sqrt(ux_error), sqrt(uxx_error)

      ! zwolnij pamiec
      deallocate(u, u_x, u_xx)

  end do ! i

end program TestDerivs

real function func(x)
  func = x*x*x*x
end function func

real function func_x(x)
  func_x = 4d0*x*x*x
end function func_x

real function func_xx(x)
  func_xx = 12d0*x*x
```

Pochodne 1, 2 V

Fortran 95

```
end function func_xx

! wyniki obliczeń:
!
!    npts      Error(1_Der)          Error(2_Der)
!       4      0.448540793E+00       0.200411100E+01
!       8      0.696229028E-01       0.242946550E+00
!      16      0.130890098E-01       0.367216199E-01
!      32      0.276197408E-02       0.616467539E-02
!      64      0.627306127E-03       0.110691791E-02
!     128      0.148910393E-03       0.211078793E-03
!     256      0.362347450E-04       0.428963548E-04
!     512      0.893428971E-05       0.929152239E-05
!    1024      0.221800198E-05       0.212429174E-05
!    2048      0.552552792E-06       0.504635577E-06
```

Porównajz wyniki własnych obliczeń z wynikami wypisanymi wyżej dla funkcji $f(x) = x^4$.