1. Introduction. This is an implementation of the Irregular Terrain Model (`ITM`), version 1.2.2. The properties of this model and the algorithm defining it have been given in the references:

Hufford, G. A., A. G. Longley, and W. A. Kissick (1982), A guide to the use of the ITS Irregular Terrain Model in the area prediction mode, NTIA Report 82-100. (NTIS Order No. PB82-217977)

Hufford, G. A. (1995), The ITS Irregular Terrain Model, version 1.2.2, the Algorithm.

In particular, the implementation here follows the latter description ("the Algorithm") as closely as seems possible. Indeed, there are below direct references to the equation numbers in "the Algorithm." These are indicated with terminology such as "[Alg 3.5]" which thus refers to equation (3.5) in that document.

2.

We begin with a declaration of the two important common blocks. The first lists the primary output values and the primary input parameters. The second contains important secondary or derived parameters.

For the primary parameters, the input values (which are often introduced by subroutines such as *qlrps*) are

> The controlling mode *mdp*, distance *dist*, antenna structural heights *hg*, wave number (radio frequency) *wn*, terrain irregularity parameter *dh*, surface refractivity *ens*, earth's effective curvature *gme*, surface transfer impedance of the ground *zgnd*, antenna effective heights *he*, horizon distances *dl*, and horizon elevation angles *the*.

while the output values are

> The error indicator *kwx*, and the reference attenuation *aref*.

⟨ Primary parameters 2 ⟩ ≡

common */prop/ kwx*, *aref*, *mdp*, *dist*, *hg*(2), *wn*, *dh*, *ens*, *gme*, *zgnd*, *he*(2), *dl*(2), *the*(2)
complex *zgnd*

This code is used in sections 4, 10, 17, 22, 28, 41, 42, 43, and 47.

3. The secondary parameters are computed in *lrprop* and consist of

> The line-of-sight distance *dlsa*, scatter distance *dx*, line-of-sight coefficients *ael*, *ak1*, *ak2*, diffraction coefficients *aed*, *emd*, scatter coefficients *aes*, *ems*, smooth earth horizon distances *dls*, total horizon distance *dla*, and total bending angle *tha*.

⟨ Secondary parameters 3 ⟩ ≡

common */propa/ dlsa*, *dx*, *ael*, *ak1*, *ak2*, *aed*, *emd*, *aes*, *ems*, *dls*(2), *dla*, *tha*

This code is used in sections 4, 10, 17, and 22.

4. LRprop. The Longley-Rice propagation program. This is the basic program; it returns the reference attenuation *aref*.

```
subroutine lrprop•(d)

@(
   Version 1.2.2 (Aug 71/Mar 77/Aug 84) of the Irregular Terrain
 Model
     by Longley and Rice (1968)
@)
      ⟨ Primary parameters 2 ⟩
      ⟨ Secondary parameters 3 ⟩
      save wlos, wscat, dmin, xae
      logical wlos, wscat, wq
      parameter (third = 1./3.)

      ⟨ LRprop 5 ⟩
      aref = max(aref, 0.)
      return
   end
```

5. The value of *mdp* controls some of the program flow. When it equals -1 we are in the point-to-point mode, when 1 we are beginning the area mode, and when 0 we are continuing the area mode. The assumption is that when one uses the area mode, one will want a sequence of results for varying distances.

⟨ LRprop 5 ⟩ ≡

```
   if (mdp ≠ 0) then
      ⟨ Do secondary parameters 6 ⟩
      ⟨ Check parameter ranges 7 ⟩
      ⟨ Diffraction coefficients 9 ⟩
   endif
   if (mdp ≥ 0) then
      mdp = 0
      dist = d
   endif
   if (dist > 0.) then
      ⟨ Check distance 8 ⟩
   endif
   if (dist < dlsa) then
      ⟨ Line-of-sight calculations 15 ⟩
   endif
   if (dist ≤ 0. | dist ≥ dlsa) then
      ⟨ Troposcatter calculations 20 ⟩
   endif
```

This code is used in section 4.

6.

$\langle$ Do secondary parameters 6 $\rangle \equiv$

```
do j = 1,2
   dls(j) = sqrt(2.*he(j)/gme)     // [Alg 3.5]
enddo
dlsa = dls(1) + dls(2)     // [Alg 3.6]
dla = dl(1) + dl(2)     // [Alg 3.7]
tha = max(the(1) + the(2), -dla*gme)     // [Alg 3.8]
wlos = F
wscat = F
```

This code is used in section 5.

7.

$\langle$ Check parameter ranges 7 $\rangle \equiv$

```
if (wn < 0.838 | wn > 210.)  kwx = max(kwx, 1)
do j = 1,2
   if (hg(j) < 1. | hg(j) > 1000.)  kwx = max(kwx, 1)
enddo
do j = 1,2
   if (abs(the(j)) > 200·10^-3 | dl(j) < 0.1*dls(j) | dl(j) > 3.*dls(j))  kwx = max(kwx, 3)
enddo
if ( ens < 250. | ens > 400. | gme < 75·10^-9 | gme > 250·10^-9 | Real(zgnd) ≤ abs(Imag(zgnd)) |
      wn < 0.419 | wn > 420. ) kwx = 4
do j = 1,2
   if (hg(j) < 0.5 | hg(j) > 3000.)  kwx = 4
enddo
dmin = abs(he(1) - he(2))/200·10^-3
```

This code is used in section 5.

8.

$\langle$ Check distance 8 $\rangle \equiv$

```
if (dist > 1000·10^3)  kwx = max(kwx, 1)
if (dist < dmin)  kwx = max(kwx, 3)
if (dist < 1·10^3 | dist > 2000·10^3)  kwx = 4
```

This code is used in section 5.

9. The Diffraction Region. This is the region beyond the smooth-earth horizon at d_{Lsa} and short of where tropospheric scatter takes over. It is a key central region and the associated coefficients must always be computed.

⟨ Diffraction coefficients 9 ⟩ ≡

$q = adiff_{10}(0.)$
$xae = (wn*gme^2)^{-third}$ // [Alg 4.2]
$d3 = \mathbf{max}(dlsa, 1.3787*xae + dla)$ // [Alg 4.3]
$d4 = d3 + 2.7574*xae$ // [Alg 4.4]
$a3 = adiff_{10}(d3)$ // [Alg 4.5]
$a4 = adiff_{10}(d4)$ // [Alg 4.6]
$emd = (a4 - a3)/(d4 - d3)$ // [Alg 4.7]
$aed = a3 - emd*d3$ // [Alg 4.8]

This code is used in section 5.

10. The function *adiff* finds the "diffraction attenuation" at the distance d. It uses a convex combination of smooth earth diffraction and double knife-edge diffraction. A call with $d = 0.$ sets up initial constants.

function *adiff*$_\bullet(d)$

⟨ Primary parameters 2 ⟩
⟨ Secondary parameters 3 ⟩
save *wd1*, *xd1*, *afo*, *qk*, *aht*, *xht*
parameter (*third* = 1./3.)

if ($d \equiv 0.$) **then**
⟨ Prepare initial diffraction constants 11 ⟩
else
⟨ Compute diffraction attenuation 12 ⟩
endif
return
end

11.

⟨ Prepare initial diffraction constants 11 ⟩ ≡

$q = hg(1)*hg(2)$
$qk = he(1)*he(2) - q$
if $(mdp < 0)\ q = q + 10.$
$wd1 = \boldsymbol{sqrt}(1. + qk/q)$
$xd1 = dla + tha/gme$ // $xd1$ and $wd1$ are parts of Q in [Alg 4.9]
$q = (1. - 0.8*\boldsymbol{exp}(-dlsa/50 \cdot 10^3))*dh$
$q = 0.78*q*\boldsymbol{exp}(-(q/16.)^{0.25})$ // $\sigma_h(dlsa)$
$afo = \boldsymbol{min}(15., 2.171*\boldsymbol{log}(1. + 4.77 \cdot 10^{-4}*hg(1)*hg(2)*wn*q))$ // [Alg 4.10]
$qk = 1./\boldsymbol{cabs}(zgnd)$
$aht = 20.$ // [Alg 6.7]
$xht = 0.$
do $j = 1, 2$
  $a = 0.5*dl(j)^2/he(j)$ // [Alg 4.15]
  $wa = (a*wn)^{third}$ // [Alg 4.16]
  $pk = qk/wa$ // [Alg 4.17]
  $q = (1.607 - pk)*151.0*wa*dl(j)/a$ // [Alg 4.18] and [Alg 6.2]
  $xht = xht + q$ // [Alg 4.19], the height-gain part
  $aht = aht + fht_{14}(q, pk)$ // [Alg 4.20]
enddo
$adiff_{10} = 0.$

This code is used in section 10.

12.

⟨ Compute diffraction attenuation 12 ⟩ ≡

$th = tha + d*gme$ // [Alg 4.12]
$ds = d - dla$
$q = 0.0795775*wn*ds*th^2$
$adiff_{10} = aknfe_{13}(q*dl(1)/(ds + dl(1))) + aknfe_{13}(q*dl(2)/(ds + dl(2)))$ // [Alg 4.14]
$a = ds/th$
$wa = (a*wn)^{third}$ // [Alg 4.16]
$pk = qk/wa$ // [Alg 4.17]
$q = (1.607 - pk)*151.0*wa*th + xht$ // [Alg 4.18] and [Alg 6.2]
$ar = 0.05751*q - 4.343*\boldsymbol{log}(q) - aht$ // [Alg 4.20]
$q = (wd1 + xd1/d)*\boldsymbol{min}(((1. - 0.8*\boldsymbol{exp}(-d/50 \cdot 10^3))*dh*wn), 6283.2)$
$wd = 25.1/(25.1 + \boldsymbol{sqrt}(q))$ // [Alg 4.9]
$adiff_{10} = ar*wd + (1. - wd)*adiff_{10} + afo$ // [Alg 4.11]

This code is used in section 10.

13. The attenuation due to a single knife edge—the Fresnel integral (in decibels) as a function of v^2. The approximation is that given in [Alg 6.1].

```
function aknfe•(v2)
  if (v2 < 5.76) then
    aknfe• = 6.02 + 9.11*sqrt(v2) − 1.27*v2
  else
    aknfe• = 12.953 + 4.343*log(v2)
  endif
  return
end
```

14. The height-gain over a smooth spherical earth—to be used in the "three radii" method. The approximation is that given in [Alg 6.4].

```
function fht•(x,pk)
  if (x < 200.) then
    w = −log(pk)
    if (pk < 1·10^−5 | x*w^3 > 5495.) then
      fht• = −117.
      if (x > 1.) fht• = 17.372*log(x) + fht•      // [Alg 6.5]
    else
      fht• = 2.5·10^−5*x^2/pk − 8.686*w − 15.     // [Alg 6.6]
    endif
  else
    fht• = 0.05751*x − 4.343*log(x)     // [Alg 6.3]
    if (x < 2000) then
      w = 0.0134*x*exp(−0.005*x)
      fht• = (1. − w)*fht• + w*(17.372*log(x) − 117.)     // [Alg 6.4]
    endif
  endif
  return
end
```

15. The Line-of-sight Region.

$\langle$ Line-of-sight calculations 15 $\rangle \equiv$

```
if (¬wlos) then
   ⟨ Line-of-sight coefficients 16 ⟩
   wlos = T
endif
if (dist > 0.)  aref = ael + ak1*dist + ak2*log(dist)    // [Alg 4.1]
```

This code is used in section 5.

16.

$\langle$ Line-of-sight coefficients 16 $\rangle \equiv$

```
q = alos_17(0.)
d2 = dlsa
a2 = aed + d2*emd
d0 = 1.908*wn*he(1)*he(2)    // [Alg 4.38]
if (aed ≥ 0.) then
   d0 = min(d0, 0.5*dla)    // [Alg 4.28]
   d1 = d0 + 0.25*(dla − d0)    // [Alg 4.29]
else
   d1 = max(−aed/emd, 0.25*dla)    // [Alg 4.39]
endif
a1 = alos_17(d1)    // [Alg 4.31]
wq = F
if (d0 < d1) then
   a0 = alos_17(d0)    // [Alg 4.30]
   q = log(d2/d0)
   ak2 = max(0., ((d2 − d0)*(a1 − a0) − (d1 − d0)*(a2 − a0))/((d2 − d0)*log(d1/d0) − (d1 − d0)*q))
     // [Alg 4.32]
   wq = aed > 0. | ak2 > 0.
   if (wq) then
     ak1 = (a2 − a0 − ak2*q)/(d2 − d0)    // [Alg 4.33]
     if (ak1 < 0.) then
       ak1 = 0.    // [Alg 4.36]
       ak2 = dim(a2, a0)/q    // [Alg 4.35]
       if (ak2 ≡ 0.)  ak1 = emd    // [Alg 4.37]
     endif
   endif
endif
if (¬wq) then
   ak1 = dim(a2, a1)/(d2 − d1)    // [Alg 4.40]
   ak2 = 0.    // [Alg 4.41]
   if (ak1 ≡ 0.)  ak1 = emd    // [Alg 4.37]
endif
ael = a2 − ak1*d2 − ak2*log(d2)    // [Alg 4.42]
```

This code is used in section 15.

17. The function *alos* finds the "line-of-sight attenuation" at the distance d. It uses a convex combination of plane earth fields and diffracted fields. A call with $d = 0.$ sets up initial constants.

function $alos_\bullet(d)$
⟨ Primary parameters 2 ⟩
⟨ Secondary parameters 3 ⟩
save *wls*
complex r

$abq(r) = \boldsymbol{Real}(r)^2 + Imag(r)^2$

if $(d \equiv 0.)$ **then**
⟨ Prepare initial line-of-sight constants 18 ⟩
else
⟨ Compute line-of-sight attenuation 19 ⟩
endif
return
end

18.

⟨ Prepare initial line-of-sight constants 18 ⟩ ≡

$wls = 0.021/(0.021 + wn*dh/\boldsymbol{max}(10 \cdot 10^3, dlsa))$ // [Alg 4.43]
$alos_{17} = 0.$

This code is used in section 17.

19.

⟨ Compute line-of-sight attenuation 19 ⟩ ≡

$q = (1. - 0.8*\boldsymbol{exp}(-d/50 \cdot 10^3))*dh$ // $\Delta h(d)$
$s = 0.78*q*\boldsymbol{exp}(-(q/16.)^{0.25})$ // $\sigma_h(d)$
$q = he(1) + he(2)$
$sps = q/\boldsymbol{sqrt}(d^2 + q^2)$ // $\sin\psi$
$r = (sps - zgnd)/(sps + zgnd)*\boldsymbol{exp}(-\boldsymbol{min}(10., wn*s*sps))$ // [Alg 4.47]
$q = abq(r)$
if $(q < 0.25 \mid q < sps)$ $r = r*\boldsymbol{sqrt}(sps/q)$ // [Alg 4.48]
$alos_{17} = emd*d + aed$ // [Alg 4.45]
$q = wn*he(1)*he(2)*2./d$ // [Alg 4.49]
if $(q > 1.57)$ $q = 3.14 - 2.4649/q$ // [Alg 4.50]
$alos_{17} = (-4.343*\boldsymbol{log}(abq(\boldsymbol{cmplx}(\boldsymbol{cos}(q), -\boldsymbol{sin}(q)) + r)) - alos_{17})*wls + alos_{17}$
// [Alg 4.51] and [Alg 4.44]

This code is used in section 17.

20. The Troposcatter Region.

⟨ Troposcatter calculations 20 ⟩ ≡

```
if (¬wscat) then
   ⟨ Troposcatter coefficients 21 ⟩
   wscat = 𝒯
endif
if (dist > dx) then
   aref = aes + ems∗dist
else
   aref = aed + emd∗dist     // [Alg 4.1]
endif
```

This code is used in section 5.

21.

⟨ Troposcatter coefficients 21 ⟩ ≡

```
q = ascat_22(0.)
d5 = dla + 200 · 10^3     // [Alg 4.52]
d6 = d5 + 200 · 10^3     // [Alg 4.53]
a6 = ascat_22(d6)     // [Alg 4.54]
a5 = ascat_22(d5)     // [Alg 4.55]
if (a5 < 1000.) then
   ems = (a6 − a5)/200 · 10^3     // [Alg 4.57]
   dx = max(dlsa, dla + 0.3∗xae∗log(47.7∗wn), (a5 − aed − ems∗d5)/(emd − ems))     // [Alg 4.58]
   aes = (emd − ems)∗dx + aed     // [Alg 4.59]
else
   ems = emd
   aes = aed
   dx = 10 · 10^6     // [Alg 4.56]
endif
```

This code is used in section 20.

22. The function *ascat* finds the "scatter attenuation" at the distance d. It uses an approximation to the methods of NBS TN101 with checks for inadmissable situations. For proper operation, the larger distance ($d = d_6$) must be the first called. A call with $d = 0.$ sets up initial constants.

```
function ascat•(d)
   ⟨ Primary parameters 2 ⟩
   ⟨ Secondary parameters 3 ⟩
   save ad, rr, etq, h0s

   if (d ≡ 0.) then
      ⟨ Prepare initial scatter constants 23 ⟩
   else
      ⟨ Compute scatter attenuation 24 ⟩
   endif
   return
end
```

23.

⟨ Prepare initial scatter constants 23 ⟩ ≡

$ad = dl(1) - dl(2)$
$rr = he(2)/he(1)$
if $(ad < 0.)$ **then**
  $ad = -ad$
  $rr = 1./rr$
endif
$etq = (5.67 \cdot 10^{-6} * ens - 2.32 \cdot 10^{-3}) * ens + 0.031$ // Part of [Alg 4.67]
$h0s = -15.$
$ascat_{22} = 0.$

This code is used in section 22.

24.

@m *Noscat*$_\bullet$ **#**:0

⟨ Compute scatter attenuation 24 ⟩ ≡

```
if (h0s > 15.) then
  h0 = h0s
else
  th = the(1) + the(2) + d*gme    // [Alg 4.61]
  r2 = 2.*wn*th
  r1 = r2*he(1)
  r2 = r2*he(2)    // [Alg 4.62]
  if (r1 < 0.2 ∧ r2 < 0.2) then
    ascat_22 = 1001.    // The function is undefined
    go to Noscat•

  endif
  ss = (d − ad)/(d + ad)    // [Alg 4.65]
  q = rr/ss
  ss = max(0.1, ss)
  q = min(max(0.1, q), 10.)
  z0 = (d − ad)*(d + ad)*th*0.25/d    // [Alg 4.66]
  et = (etq*exp(−min(1.7, z0/8.0·10^3)^6) + 1.)*z0/1.7556·10^3    // [Alg 4.67]
  ett = max(et, 1.)
  h0 = (h0f_25(r1, ett) + h0f_25(r2, ett))*0.5    // [Alg 6.12]
  h0 = h0 + min(h0, (1.38 − log(ett))*log(ss)*log(q)*0.49)    // [Alg 6.10] and [Alg 6.11]
  h0 = dim(h0, 0.)
  if (et < 1.)
  h0 = et*h0 + (1. − et)*4.343*log(((1. + 1.4142/r1)*(1. + 1.4142/r2))^2*(r1 + r2)/(r1 + r2 + 2.8284))
    // [Alg 6.14]
  if (h0 > 15. ∧ h0s ≥ 0.)  h0 = h0s
endif
h0s = h0
th = tha + d*gme    // [Alg 4.60]
ascat_22 = ahd_26(th*d) + 4.343*log(47.7*wn*th^4) − 0.1*(ens − 301.)*exp(−th*d/40·10^3) + h0
    // [Alg 4.63] and [Alg 6.8]
Noscat•: continue
```

This code is used in section 22.

25. This is the H_{01} funmction for scatter fields as defined in [Alg §6]

```
function h0f•(r,et)
  dimension a(5), b(5)
  data a(1), a(2), a(3), a(4), a(5)/25., 80., 177., 395., 705./
  data b(1), b(2), b(3), b(4), b(5)/24., 45., 68., 80., 105./

  it = et
  if (it ≤ 0) then
    it = 1
    q = 0.
  else if (it ≥ 5) then
    it = 5
    q = 0.
  else
    q = et − it
  endif
  x = (1./r)^2
  h0f• = 4.343*log((a(it)*x + b(it))*x + 1.)     // [Alg 6.13]
  if (q ≠ 0.)  h0f• = (1. − q)*h0f• + q*4.343*log((a(it + 1)*x + b(it + 1))*x + 1.)
  return
end
```

26. This is the $F(\theta d)$ function for scatter fields

```
function ahd•(td)
  dimension a(3), b(3), c(3)
  data a(1), a(2), a(3)/133.4, 104.6, 71.8/
  data b(1), b(2), b(3)/0.332·10^-3, 0.212·10^-3, 0.157·10^-3/
  data c(1), c(2), c(3)/−4.343, −1.086, 2.171/

  if (td ≤ 10·10^3) then
    i = 1
  else if (td ≤ 70·10^3) then
    i = 2
  else
    i = 3
  endif
  ahd• = a(i) + b(i)*td + c(i)*log(td)     // [Alg 6.9]
  return
end
```

27. The Statistics.

LRprop will stand alone to compute *aref*. To complete the story, however, one must find the quantiles of the attenuation and this is what *avar* will do. It, too, is a stand alone subroutine, except that it requires the output from *LRprop*, as well as values in a "variability parameters" common block. These latter values consist of

> A control switch *lvar*, the standard deviation of situation variability (confidence) *sgc*, the desired mode of variability *mdvar*, and the climate indicator *klim*.

Of these, *sgc* is output and may be used to answer the inverse problem: with what confidence will a threshold signal level be exceeded.

⟨ Variability parameters 27 ⟩ ≡

common */propv/ lvar*, *sgc*, *mdvar*, *klim*

This code is used in sections 28, 42, and 43.

28. When in the area prediction mode, one needs a threefold quantile of attenuation which corresponds to the fraction q_T of time, the fraction q_L of locations, and the fraction q_S of "situations." In the point-to-point mode, one needs only q_T and q_S. For efficiency, *avar* is written as a function of the "standard normal deviates" z_T, z_L, and z_S corresponding to the requested fractions. Thus, for example, $q_T = Q(z_T)$ where $Q(z)$ is the "complementary standard normal distribution." For the point-to-point mode one sets $z_L = 0$ which corresponds to the median $q_L = 0.50$.

The subprogram is written trying to reduce duplicate calculations. This is done through the switch *lvar*. On first entering, set $lvar = 5$. Then all parameters will be initialized, and *lvar* will be changed to 0. If the program is to be used to find several quantiles with different values of z_T, z_L, or z_S, then *lvar* should be 0, as it is. If the distance is changed, set $lvar = 1$ and parameters that depend on the distance will be recomputed. If antenna heights are changed, set $lvar = 2$; if the frequency, $lvar = 3$; if the mode of variability *mdvar*, set $lvar = 4$; and finally if the climate is changed, set $lvar = 5$. The higher the value of *lvar*, the more parameters will be recomputed.

function *avar*•(*zzt*,*zzl*,*zzc*)

⟨Primary parameters 2⟩
⟨Variability parameters 27⟩
save *kdv*, *wl*, *ws*, *dexa*, *de*, *vmd*, *vs0*, *sgl*, *sgtm*, *sgtp*, *sgtd*, *tgtd*, *gm*, *gp*, *cv1*, *cv2*, *yv1*, *yv2*, *yv3*, *csm1*, *csm2*, *ysm1*, *ysm2*, *ysm3*, *csp1*, *csp2*, *ysp1*, *ysp2*, *ysp3*, *csd1*, *zd*, *cfm1*, *cfm2*, *cfm3*, *cfp1*, *cfp2*, *cfp3*

dimension *bv1*(7), *bv2*(7), *xv1*(7), *xv2*(7), *xv3*(7)
dimension *bsm1*(7), *bsm2*(7), *xsm1*(7), *xsm2*(7), *xsm3*(7)
dimension *bsp1*(7), *bsp2*(7), *xsp1*(7), *xsp2*(7), *xsp3*(7)
dimension *bsd1*(7), *bzd1*(7)
dimension *bfm1*(7), *bfm2*(7), *bfm3*(7), *bfp1*(7), *bfp2*(7), *bfp3*(7)

logical *ws*, *wl*

parameter (*third* = 1./3.)

⟨Climatic constants 29⟩

⟨Function *curv* 30⟩

if ($lvar > 0$) **then**
⟨Set up variability coefficients 31⟩
$lvar = 0$
endif
⟨Correct normal deviates 37⟩
⟨Resolve standard deviations 38⟩
⟨Resolve deviations *yr*, *yc* 39⟩
$avar_\bullet = aref - vmd - yr - sgc{*}zc$ // [Alg 5.1]
if ($avar_\bullet < 0.$) $avar_\bullet = avar_\bullet{*}(29. - avar_\bullet)/(29. - 10.{*}avar_\bullet)$ // [Alg 5.2]
return
end

29.

⟨ Climatic constants 29 ⟩ ≡

// equatorial, continental subtropical, maritime subtropical, desert, continental temperate, maritime over land, maritime over sea

data $bv1$ /−9.67, −0.62, 1.26, −9.21, −0.62, −0.39, 3.15/
data $bv2$ /12.7, 9.19, 15.5, 9.05, 9.19, 2.86, 857.9/
data $xv1$ /$144.9 \cdot 10^3, 228.9 \cdot 10^3, 262.6 \cdot 10^3, 84.1 \cdot 10^3, 228.9 \cdot 10^3, 141.7 \cdot 10^3, 2222. \cdot 10^3$/
data $xv2$ /$190.3 \cdot 10^3, 205.2 \cdot 10^3, 185.2 \cdot 10^3, 101.1 \cdot 10^3, 205.2 \cdot 10^3, 315.9 \cdot 10^3, 164.8 \cdot 10^3$/
data $xv3$ /$133.8 \cdot 10^3, 143.6 \cdot 10^3, 99.8 \cdot 10^3, 98.6 \cdot 10^3, 143.6 \cdot 10^3, 167.4 \cdot 10^3, 116.3 \cdot 10^3$/
data $bsm1$ /2.13, 2.66, 6.11, 1.98, 2.68, 6.86, 8.51/
data $bsm2$ /159.5, 7.67, 6.65, 13.11, 7.16, 10.38, 169.8/
data $xsm1$ /$762.2 \cdot 10^3, 100.4 \cdot 10^3, 138.2 \cdot 10^3, 139.1 \cdot 10^3, 93.7 \cdot 10^3, 187.8 \cdot 10^3, 609.8 \cdot 10^3$/
data $xsm2$ /$123.6 \cdot 10^3, 172.5 \cdot 10^3, 242.2 \cdot 10^3, 132.7 \cdot 10^3, 186.8 \cdot 10^3, 169.6 \cdot 10^3, 119.9 \cdot 10^3$/
data $xsm3$ /$94.5 \cdot 10^3, 136.4 \cdot 10^3, 178.6 \cdot 10^3, 193.5 \cdot 10^3, 133.5 \cdot 10^3, 108.9 \cdot 10^3, 106.6 \cdot 10^3$/
data $bsp1$ /2.11, 6.87, 10.08, 3.68, 4.75, 8.58, 8.43/
data $bsp2$ /102.3, 15.53, 9.60, 159.3, 8.12, 13.97, 8.19/
data $xsp1$ /$636.9 \cdot 10^3, 138.7 \cdot 10^3, 165.3 \cdot 10^3, 464.4 \cdot 10^3, 93.2 \cdot 10^3, 216.0 \cdot 10^3, 136.2 \cdot 10^3$/
data $xsp2$ /$134.8 \cdot 10^3, 143.7 \cdot 10^3, 225.7 \cdot 10^3, 93.1 \cdot 10^3, 135.9 \cdot 10^3, 152.0 \cdot 10^3, 188.5 \cdot 10^3$/
data $xsp3$ /$95.6 \cdot 10^3, 98.6 \cdot 10^3, 129.7 \cdot 10^3, 94.2 \cdot 10^3, 113.4 \cdot 10^3, 122.7 \cdot 10^3, 122.9 \cdot 10^3$/
data $bsd1$ /1.224, 0.801, 1.380, 1.000, 1.224, 1.518, 1.518/
data $bzd1$ /1.282, 2.161, 1.282, 20., 1.282, 1.282, 1.282/
data $bfm1$ /1., 1., 1., 1., 0.92, 1., 1./
data $bfm2$ /0., 0., 0., 0., 0.25, 0., 0./
data $bfm3$ /0., 0., 0., 0., 1.77, 0., 0./
data $bfp1$ /1., 0.93, 1., 0.93, 0.93, 1., 1./
data $bfp2$ /0., 0.31, 0., 0.19, 0.31, 0., 0./
data $bfp3$ /0., 2.00, 0., 1.79, 2.00, 0., 0./

data rt, rl /7.8, 24./

This code is used in section 28.

30.

⟨ Function $curv$ 30 ⟩ ≡

$$curv(c1, c2, x1, x2, x3) = (c1 + c2/(1. + ((de - x2)/x3)^2))*((de/x1)^2)/(1. + ((de/x1)^2))$$

This code is used in section 28.

31.

```
@m Climate• #:0
@m Mode_var• #:0
@m Frequency• #:0
@m System• #:0
@m Distance• #:0
```

⟨ Set up variability coefficients 31 ⟩ ≡

```
           if (lvar < 5) go to (Distance•,System•,Frequency•,Mode_var•),lvar
Climate•:  continue
           if (klim ≤ 0 | klim > 7) then
              klim = 5
              kwx = max(kwx, 2)
           endif
           ⟨ Climatic coefficients 32 ⟩
Mode_var•: continue
           ⟨ Mode of variability coefficients 33 ⟩
Frequency•: continue
           ⟨ Frequency coefficients 34 ⟩
System•:   continue
           ⟨ System coefficients 35 ⟩
Distance•: continue
           ⟨ Distance coefficients 36 ⟩
```

This code is used in section 28.

32.

$\langle$ Climatic coefficients 32 $\rangle \equiv$

$$
\begin{aligned}
&cv1 = bv1(klim)\\
&cv2 = bv2(klim)\\
&yv1 = xv1(klim)\\
&yv2 = xv2(klim)\\
&yv3 = xv3(klim)\\
&csm1 = bsm1(klim)\\
&csm2 = bsm2(klim)\\
&ysm1 = xsm1(klim)\\
&ysm2 = xsm2(klim)\\
&ysm3 = xsm3(klim)\\
&csp1 = bsp1(klim)\\
&csp2 = bsp2(klim)\\
&ysp1 = xsp1(klim)\\
&ysp2 = xsp2(klim)\\
&ysp3 = xsp3(klim)\\
&csd1 = bsd1(klim)\\
&zd = bzd1(klim)\\
&cfm1 = bfm1(klim)\\
&cfm2 = bfm2(klim)\\
&cfm3 = bfm3(klim)\\
&cfp1 = bfp1(klim)\\
&cfp2 = bfp2(klim)\\
&cfp3 = bfp3(klim)
\end{aligned}
$$

This code is used in section 31.

33.

$\langle$ Mode of variability coefficients 33 $\rangle \equiv$

$$
\begin{aligned}
&kdv = mdvar\\
&ws = kdv \geq 20\\
&\textbf{if}\,(ws)\ \ kdv = kdv - 20\\
&wl = kdv \geq 10\\
&\textbf{if}\,(wl)\ \ kdv = kdv - 10\\
&\textbf{if}\,(kdv < 0 \,|\, kdv > 3)\ \textbf{then}\\
&\quad kdv = 0\\
&\quad kwx = \boldsymbol{max}(kwx, 2)\\
&\textbf{endif}
\end{aligned}
$$

This code is used in section 31.

34.

⟨ Frequency coefficients 34 ⟩ ≡

$q = \boldsymbol{log}(0.133 * wn)$
$gm = cfm1 + cfm2/((cfm3 * q)^2 + 1.)$
$gp = cfp1 + cfp2/((cfp3 * q)^2 + 1.)$

This code is used in section 31.

35.

⟨ System coefficients 35 ⟩ ≡

$dexa = \boldsymbol{sqrt}(18 \cdot 10^6 * he(1)) + \boldsymbol{sqrt}(18 \cdot 10^6 * he(2)) + (575.7 \cdot 10^{12}/wn)^{third}$ // [Alg 5.3]

This code is used in section 31.

36.

⟨ Distance coefficients 36 ⟩ ≡

if $(dist < dexa)$ **then**
  $de = 130 \cdot 10^3 * dist/dexa$
else
  $de = 130 \cdot 10^3 + dist - dexa$ // [Alg 5.4]
endif
$vmd = curv(cv1, cv2, yv1, yv2, yv3)$ // [Alg 5.5]
$sgtm = curv(csm1, csm2, ysm1, ysm2, ysm3) * gm$
$sgtp = curv(csp1, csp2, ysp1, ysp2, ysp3) * gp$ // [Alg 5.7]
$sgtd = sgtp * csd1$ // [Alg 5.8]
$tgtd = (sgtp - sgtd) * zd$
if (wl) **then**
  $sgl = 0.$
else
  $q = (1. - 0.8 * \boldsymbol{exp}(-dist/50 \cdot 10^3)) * dh * wn$
  $sgl = 10. * q/(q + 13.)$ // [Alg 5.9]
endif
if (ws) **then**
  $vs0 = 0.$
else
  $vs0 = (5. + 3. * \boldsymbol{exp}(-de/100 \cdot 10^3))^2$ // [Alg 5.10]
endif

This code is used in section 31.

37.

$\langle$ Correct normal deviates 37 $\rangle \equiv$

$zt = zzt$
$zl = zzl$
$zc = zzc$
if $(kdv \equiv 0)$ **then**
  $zt = zc$
  $zl = zc$
else if $(kdv \equiv 1)$ **then**
  $zl = zc$
else if $(kdv \equiv 2)$ **then**
  $zl = zt$
endif
if $(\boldsymbol{abs}(zt) > 3.10 \mid \boldsymbol{abs}(zl) > 3.10 \mid \boldsymbol{abs}(zc) > 3.10)$ $kwx = \boldsymbol{max}(kwx, 1)$

This code is used in section 28.

38.

$\langle$ Resolve standard deviations 38 $\rangle \equiv$

if $(zt < 0.)$ **then**
  $sgt = sgtm$
else if $(zt \le zd)$ **then**
  $sgt = sgtp$
else
  $sgt = sgtd + tgtd/zt$ // [Alg 5.6]
endif
$vs = vs0 + (sgt*zt)^2/(rt + zc^2) + (sgl*zl)^2/(rl + zc^2)$ // [Alg 5.11]

This code is used in section 28.

39.

$\langle$ Resolve deviations yr, yc 39 $\rangle \equiv$

if $(kdv \equiv 0)$ **then**
  $yr = 0.$
  $sgc = \boldsymbol{sqrt}(sgt^2 + sgl^2 + vs)$
else if $(kdv \equiv 1)$ **then**
  $yr = sgt*zt$
  $sgc = \boldsymbol{sqrt}(sgl^2 + vs)$
else if $(kdv \equiv 2)$ **then**
  $yr = \boldsymbol{sqrt}(sgt^2 + sgl^2)*zt$
  $sgc = \boldsymbol{sqrt}(vs)$
else
  $yr = sgt*zt + sgl*zl$
  $sgc = \boldsymbol{sqrt}(vs)$
endif

This code is used in section 28.

40. Preparatory Subroutines.

The next three subroutines may be used to introduce input parameters for *LRprop*. One first calls *qlrps* 41 and then either *qlra* 42 (for the area prediction mode) or *qlrpfl* 43 (for the point-to-point mode).

41. This subroutine converts the frequency *fmhz*, the surface refractivity reduced to sea level *en0* and general system elevation *zsys*, and the polarization and ground constants *eps*, *sgm*, to wave number *wn*, surface refractivity *ens*, effective earth curvature *gme*, and surface impedance *zgnd*. It may be used with either the area prediction or the point-to-point mode.

```
subroutine qlrps•(fmhz, zsys, en0, ipol, eps, sgm)
   ⟨Primary parameters 2⟩
   complex zq
   data gma/157 · 10^−9/
   wn = fmhz/47.7     // [Alg 1.1]
   ens = en0
   if (zsys ≠ 0.)  ens = ens∗exp(−zsys/9460.)     // [Alg 1.2]
   gme = gma∗(1. − 0.04665∗exp(ens/179.3))     // [Alg 1.3]
   zq = cmplx(eps, 376.62∗sgm/wn)     // [Alg 1.5]
   zgnd = csqrt(zq − 1.)
   if (ipol ≠ 0)  zgnd = zgnd/zq     // [Alg 1.4]
   return
end
```

42. This is used to prepare the model in the area prediction mode. Normally, one first calls *qlrps* and then *qlra*. Before calling the latter, one should have defined in the ⟨ Primary parameters 2 ⟩ the antenna heights *hg*, the terrain irregularity parameter *dh* , and (probably through *qlrps*) the variables *wn*, *ens*, *gme*, and *zgnd*. The input *kst* will define siting criteria for the terminals, *klimx* the climate, and *mdvarx* the mode of variability. If $klimx \le 0$ or $mdvarx < 0$ the associated parameters remain unchanged.

The operational flow of a calling program might appear as follows.

```
set kwx = 0, lvar = 5;
define hg, dh and call qlrps;
optionally, define mdvar, klim;
call qlra;
loop for selected distances d:
   set lvar = max(lvar, 1);
   call lrprop(d);
   loop for selected quantiles:
      A = avar(...);
      output A;
   repeat;
repeat;
check kwx;
   end
```

```
subroutine qlra•(kst,klimx,mdvarx)
  dimension kst(2)

  ⟨ Primary parameters 2 ⟩
  ⟨ Variability parameters 27 ⟩

  do j = 1,2
    if (kst(j) ≤ 0) then
      he(j) = hg(j)
    else
      q = 4.
      if (kst(j) ≠ 1)  q = 9.
      if (hg(j) < 5.)  q = q*sin(0.3141593*hg(j))
      he(j) = hg(j) + (1. + q)*exp(−min(20., 2.*hg(j)/max(1 · 10^−3, dh)))
    endif
    q = sqrt(2.*he(j)/gme)
    dl(j) = q*exp(−0.07*sqrt(dh/max(he(j), 5.)))
    the(j) = (0.65*dh*(q/dl(j) − 1.) − 2.*he(j))/q
  enddo

  mdp = 1
  lvar = max(lvar, 3)
  if (mdvarx ≥ 0) then
    mdvar = mdvarx
    lvar = max(lvar, 4)
  endif
  if (klimx > 0) then
    klim = klimx
    lvar = 5
  endif
  return
end
```

43. This subroutine may be used to prepare for the point-to-point mode. Since the path is fixed, it has only one value of *aref* and therefore at the end of the routine there is a call to *lrprop*. To complete the process one needs to call *avar* for whatever quantiles are desired.

This mode requires the terrain profile lying between the terminals. This should be a sequence of surface elevations at points along the great circle path joining the two points. It should start at the ground beneath the first terminal and end at the ground beneath the second. In the present routine it is assumed that the elevations are *equispaced* along the path. They are stored in the array *pfl* along with two defining parameters. We will have $pfl(1) = enp$, the number (as a real value) of increments in the path; $pfl(2) = xi$, the length of each increment; $pfl(3) = z(0)$, the beginning elevation; and then $pfl(np + 3) = z(np)$, the last elevation.

The operational flow of a calling program might appear as follows.

```
set kwx = 0, lvar = 5;
define pfl, hg and call qlrps;
optionally, define mdvar, klim;
call qlrpfl;
loop for selected quantiles:
   A = avar(...);
   output A;
repeat;
check kwx;
   end
```

```
subroutine qlrpfl•(pfl,klimx,mdvarx)
   dimension pfl(*)

   ⟨Primary parameters 2⟩
   ⟨Variability parameters 27⟩

   dimension xl(2)

   dist = pfl(1)*pfl(2)
   np = pfl(1)
   ⟨Horizons and dh from pfl 44⟩
   if (dl(1) + dl(2) > 1.5*dist) then
      ⟨Redo line-of-sight horizons 45⟩
   else
      ⟨Transhorizon effective heights 46⟩
   endif

   mdp = −1
   lvar = max(lvar,3)
   if (mdvarx ≥ 0) then
      mdvar = mdvarx
      lvar = max(lvar,4)
   endif
   if (klimx > 0) then
      klim = klimx
      lvar = 5
   endif

   call lrprop_4(0.)

   return
end
```

44. Here we call the subroutine *hzns* to find the horizons and *dlthx* to find *dh*.

⟨ Horizons and *dh* from *pfl* 44 ⟩ ≡

$$
\begin{array}{l}
\textbf{call } hzns_{47}(pfl) \\
\textbf{do } j = 1, 2 \\
\quad xl(j) = \boldsymbol{min}(15.{*}hg(j), 0.1{*}dl(j)) \\
\textbf{enddo} \\
xl(2) = dist - xl(2) \\
dh = dlthx_{48}(pfl, xl(1), xl(2))
\end{array}
$$

This code is used in section 43.

45. If the path is line-of-sight, we still need to know where the horizons might have been, and so we turn to techniques used in the area prediction mode.

⟨ Redo line-of-sight horizons 45 ⟩ ≡

$$
\begin{array}{l}
\textbf{call } zlsq1_{53}(pfl, xl(1), xl(2), za, zb) \\
he(1) = hg(1) + \boldsymbol{dim}(pfl(3), za) \\
he(2) = hg(2) + \boldsymbol{dim}(pfl(np+3), zb) \\
\textbf{do } j = 1, 2 \\
\quad dl(j) = \boldsymbol{sqrt}(2.{*}he(j)/gme){*}\boldsymbol{exp}(-0.07{*}\boldsymbol{sqrt}(dh/\boldsymbol{max}(he(j), 5.))) \\
\textbf{enddo} \\
q = dl(1) + dl(2) \\
\textbf{if } (q \le dist) \textbf{ then} \\
\quad q = (dist/q)^2 \\
\quad \textbf{do } j = 1, 2 \\
\quad\quad he(j) = he(j){*}q \\
\quad\quad dl(j) = \boldsymbol{sqrt}(2.{*}he(j)/gme){*}\boldsymbol{exp}(-0.07{*}\boldsymbol{sqrt}(dh/\boldsymbol{max}(he(j), 5.))) \\
\quad \textbf{enddo} \\
\textbf{endif} \\
\textbf{do } j = 1, 2 \\
\quad q = \boldsymbol{sqrt}(2.{*}he(j)/gme) \\
\quad the(j) = (0.65{*}dh{*}(q/dl(j) - 1.) - 2.{*}he(j))/q \\
\textbf{enddo}
\end{array}
$$

This code is used in section 43.

46.

⟨ Transhorizon effective heights 46 ⟩ ≡

$$
\begin{array}{l}
\textbf{call } zlsq1_{53}(pfl, xl(1), 0.9{*}dl(1), za, q) \\
\textbf{call } zlsq1_{53}(pfl, dist - 0.9{*}dl(2), xl(2), q, zb) \\
he(1) = hg(1) + \boldsymbol{dim}(pfl(3), za) \\
he(2) = hg(2) + \boldsymbol{dim}(pfl(np+3), zb)
\end{array}
$$

This code is used in section 43.

47. Here we use the terrain profile *pfl* to find the two horizons. Output consists of the horizon distances *dl* and the horizon take-off angles *the*. If the path is line-of-sight, the routine sets both horizon distances equal to *dist*.

```
@m End_hz• #:0

    subroutine hzns•(pfl)
      dimension pfl(*)
      ⟨ Primary parameters 2 ⟩
      logical wq
      np = pfl(1)
      xi = pfl(2)
      za = pfl(3) + hg(1)
      zb = pfl(np + 3) + hg(2)
      qc = 0.5*gme
      q = qc*dist
      the(2) = (zb − za)/dist
      the(1) = the(2) − q
      the(2) = −the(2) − q
      dl(1) = dist
      dl(2) = dist
      if (np < 2) go to End_hz•
      sa = 0.
      sb = dist
      wq = T
      do i = 2, np
        sa = sa + xi
        sb = sb − xi
        q = pfl(i + 2) − (qc*sa + the(1))*sa − za
        if (q > 0.) then
          the(1) = the(1) + q/sa
          dl(1) = sa
          wq = F
        endif
        if (¬wq) then
          q = pfl(i + 2) − (qc*sb + the(2))*sb − zb
          if (q > 0.) then
            the(2) = the(2) + q/sb
            dl(2) = sb
          endif
        endif
      enddo
End_hz•: return
    end
```

48. Using the terrain profile *pfl* we find Δh, the interdecile range of elevations between the two points *x1* and *x2*.

```
@m End_dh• #:0
@m Reduce• #:0

      function dlthx•(pfl,x1,x2)
        dimension pfl(*)
        dimension s(247)
        np = pfl(1)
        xa = x1/pfl(2)
        xb = x2/pfl(2)
        dlthx• = 0.
        if (xb - xa < 2.) go to End_dh•
        ka = 0.1*(xb - xa + 8.)
        ka = min0(max0(4, ka), 25)
        n = 10*ka - 5
        kb = n - ka + 1
        sn = n - 1
        s(1) = sn
        s(2) = 1.
        xb = (xb - xa)/sn
        k = xa + 1.
        xa = xa - float(k)
        do j = 1, n
Reduce•:  if (xa > 0. ∧ k < np) then
            xa = xa - 1.
            k = k + 1
            go to Reduce•
          endif
          s(j + 2) = pfl(k + 3) + (pfl(k + 3) - pfl(k + 2))*xa
          xa = xa + xb
        enddo
        call zlsq1_53(s, 0., sn, xa, xb)
        xb = (xb - xa)/sn
        do j = 1, n
          s(j + 2) = s(j + 2) - xa
          xa = xa + xb
        enddo
        dlthx• = qtile_52(n, s(3), ka) - qtile_52(n, s(3), kb)
        dlthx• = dlthx•/(1. - 0.8*exp(-(x2 - x1)/50·10^3))
End_dh•: return
      end
```

49. Miscellaneous Aids.

50. The standard normal complementary probability—the function $Q(x) = 1/\sqrt{2\pi}\int_x^\infty e^{-t^2/2}dt$. The approximation is due to C. Hastings, Jr. ("Approximations for digital computers," Princeton Univ. Press, 1955) and the maximum error should be 7.5×10^{-8}.

```
function qerf•(z)

data b1, b2, b3, b4, b5 /0.319381530, −0.356563782, 1.781477937, −1.821255987, 1.330274429/
data rp, rrt2pi /4.317008, 0.398942280/

   x = z
   t = abs(x)
   if (t < 10.) go to 1
   qerf• = 0.
   go to 2
1: t = rp/(t + rp)
   qerf• = exp(−0.5∗x²)∗rrt2pi∗((((b5∗t + b4)∗t + b3)∗t + b2)∗t + b1)∗t
2: if (x < 0.) qerf• = 1. − qerf•
   return
end
```

51. The inverse of *qerf*—the solution for x to $q = Q(x)$. The approximation is due to C. Hastings, Jr. ("Approximations for digital computers," Princeton Univ. Press, 1955) and the maximum error should be 4.5×10^{-4}.

```
function qerfi•(q)

data c0, c1, c2 /2.515516698, 0.802853, 0.010328/
data d1, d2, d3 /1.432788, 0.189269, 0.001308/

   x = 0.5 − q
   t = amax1(0.5 − abs(x), 0.000001)
   t = sqrt(−2.∗alog(t))
   qerfi• = t − ((c2∗t + c1)∗t + c0)/(((d3∗t + d2)∗t + d1)∗t + 1.)
   if (x < 0.) qerfi• = −qerfi•
   return
end
```

52. This routine provides a quantile. It reorders the array a so that $a(j), j = 1 \ldots i_r$ are all greater that or equal to all $a(i), i = i_r \ldots nn$. In particular, $a(i_r)$ will have the same value it would have if a were completely sorted in descending order. The returned value is $qtile = a(i_r)$.

```
@m Qt0• #:0
@m Qt1• #:0
@m Qt2• #:0
@m Qt3• #:0

    function qtile•(nn,a,ir)
      dimension a(nn)

      m = 1
      n = nn
      k = min(max(1, ir), n)
Qt0•: continue
      q = a(k)
      i0 = m
      j1 = n
Qt1•: continue
      do i = i0, n
        if (a(i) < q) go to Qt2•
      enddo
      i = n
Qt2•: do j = j1, m, −1
        if (a(j) > q) go to Qt3•
      enddo
      j = m
Qt3•: if (i < j) then
        r = a(i)
        a(i) = a(j)
        a(j) = r
        i0 = i + 1
        j1 = j − 1
        go to Qt1•
      else if (i < k) then
        a(k) = a(i)
        a(i) = q
        m = i + 1
        go to Qt0•
      else if (j > k) then
        a(k) = a(j)
        a(j) = q
        n = j − 1
        go to Qt0•
      endif
      qtile• = q
      return
    end
```

53. A linear least squares fit between x_1, x_2 to the function described by the array z. This array must have a special format: $z(1) = en$, the number of equally large intervals, $z(2) = \xi$, the interval length, and $z(j+3), j = 0, \ldots, n$, function values. The output consists of values of the required line, z_0 at 0, z_n at $x_t = n\xi$.

```
subroutine zlsq1•(z,x1,x2,z0,zn)
  dimension z(*)

  xn = z(1)
  xa = aint(dim(x1/z(2), 0.))
  xb = xn − aint(dim(xn, x2/z(2)))
  if (xb ≤ xa) then
    xa = dim(xa, 1.)
    xb = xn − dim(xn, xb + 1.)
  endif
  ja = xa
  jb = xb
  n = jb − ja
  xa = xb − xa
  x = −0.5*xa
  xb = xb + x
  a = 0.5*(z(ja + 3) + z(jb + 3))
  b = 0.5*(z(ja + 3) − z(jb + 3))*x
  do i = 2, n
    ja = ja + 1
    x = x + 1.
    a = a + z(ja + 3)
    b = b + z(ja + 3)*x
  enddo
  a = a/xa
  b = b*12./((xa*xa + 2.)*xa)
  z0 = a − b*xb
  zn = a + b*(xn − xb)
  return
end
```

54. Index.

COMMAND LINE: `"C:\DOSH\FWEB\FWEAVE.EXE ITM"`.

WEB FILE: `"ITM.web"`.

CHANGE FILE: `(none)`.

GLOBAL LANGUAGE: FORTRAN.

The Irregular Terrain Model

August 1, 2002
14:27