

September 6, 2023

The diagram illustrates the progression of COVID-19 infection through various stages, starting from a Susceptible state at Time 0. The progression is as follows:

- Susceptible** (Initial state)
- Latent** (Probability p from Susceptible, $1-p$ from Susceptible)
- Presymptomatic** (Probability q from Latent, $1-q$ from Latent)
- Asymptomatic** (Probability $1-q$ from Presymptomatic, $1-q$ from Presymptomatic)
- Mild** (Probability $1-r$ from Asymptomatic, $1-r$ from Asymptomatic)
- Severe** (Probability r from Mild, r from Mild)
- Resolved Symptomatic** (Probability 1 from Severe, Probability 2 from Mild, Probability 3 from Asymptomatic, Probability 4 from Presymptomatic, Probability 5 from Latent, Probability 6 from Susceptible)
- Resolved Asymptomatic** (Probability 1 from Severe, Probability 2 from Mild, Probability 3 from Asymptomatic, Probability 4 from Presymptomatic, Probability 5 from Latent, Probability 6 from Susceptible)

Legend: s Test sensitivity

1

```

library(coga,quietly=TRUE)

.Sus = 1
.Lat = 2
.Asy = 3
.Pre = 4
.Mld = 5
.Svr = 6
.ResS = 7
.ResA = 8
.NStates = .ResA

.U = 1
.V = 2
.W = 3
.X = 4
.Y = 5
.Z = 6

stateNames = c("Sus","Lat","Asy","Pre","Mld","Svr","ResS","ResA")

stateCols = c("green","yellow","pink","orange","magenta","red","cyan","cyan")

```

Model functions

deltaParams()

A function to make a data structure holding parameters to model the Delta SARS-CoV-2 variant.

```

deltaParams = function()
{
  msus = 1000

  list (  Psymp = 0.5,
          Psevere = 0.1,
          Msus = msus,
          Vsus = msus^2,
          Mlat = 5.1,
          Vlat = 7.5,
          Masy = 7,
          Vasy = 7.5,
          Mpre = 3,
          Vpre = 1.0,

```

```

        Mmld = 5,
        Vmld = 45.5,
        Msev = 11,
        Vsev = 45.5
    )
}

```

omicronParams()

A function to make a data structure holding parameters to model the Omicron SARS-CoV-2 variant.

```

omicronParams = function()
{
    ommod = 0.6
    ommod2 = 0.9
    msus = 1000

    list( Psymp = 0.5,
          Psevere = 0.1,
          Msus = msus,
          Vsus = msus^2,
          Mlat = 5.1 * ommod,
          Vlat = 7.5 * ommod^2,
          Masy = 7 * ommod2,
          Vasy = 7.5 * ommod2^2,
          Mpre = 3 * ommod2,
          Vpre = 1.0 * ommod2^2,
          Mmld = 5 * ommod2,
          Vmld = 45.5 * ommod2^2,
          Msev = 11 * ommod2,
          Vsev = 45.5 * ommod2^2,
          ommod = ommod,
          ommod2 = ommod2
    )
}

```

makeModel()

A function to convert a SARS-CoV-2 model specified by means and variances into a collection of Gamma shape and rate parameters, with associated path splitting probabilities.

```

makeModel = function(Q)
{
  # Order must match definitions above
  means = c(Q$Msus,Q$Mlat,Q$Masy,Q$Mpre,Q$Mmld,Q$Msev)
  vars = c(Q$Vsus,Q$Vlat,Q$Vasy,Q$Vpre,Q$Vmld,Q$Vsev)
  list(q=Q$Psymp, r=Q$Psevere, shape=means^2/vars, rate=means/vars)
}

```

Exact computation of state probabilities

FF()

A function calling `pcoga()` to calculate the cumulative distribution function of a sum selected Gamma random variables.

```

FF = function(x,P,use)
{
  pcoga(x,P$shape[use],P$rate[use])
}

```

StateProbs()

A function to calculate the state probabilities for a vector of times `x`, using model `M`, with probability `p` that a transmission occurred at time 0.

```

StateProbs = function(x,p,M)
{
  S1 = matrix(0,nrow=.NStates,ncol=length(x))
  S1[.Sus,] = 0
  fv = FF(x,M,c(.V))
  S1[.Lat,] = 1-fv
  fvw = FF(x,M,c(.V,.W))
  S1[.Asy,] = (1-M$q)*(fv-fvw)
  fvx = FF(x,M,c(.V,.X))
  S1[.Pre,] = M$q*(fv-fvx)
  fvxxy = FF(x,M,c(.V,.X,.Y))
  S1[.Mld,] = M$q*(1-M$r)*(fvx-fvxxy)
  fvxz = FF(x,M,c(.V,.X,.Z))
  S1[.Svr,] = M$q*M$r*(fvx-fvxz)
  S1[.ResS,] = M$q*(M$r*fvxz + (1-M$r)*fvxy)
  S1[.ResA,] = (1-M$q)*fvw
}

```

```

S0 = matrix(0,nrow=.NStates,ncol=length(x))
fu = FF(x,M,c(.U))
S0[.Sus,] = 1-fu
fuv = FF(x,M,c(.U,.V))
S0[.Lat,] = fu-fuv
fuvw = FF(x,M,c(.U,.V,.W))
S0[.Asy,] = (1-M$q)*(fuv-fuvw)
fuvx = FF(x,M,c(.U,.V,.X))
S0[.Pre,] = M$q * (fuv - fuvx)
fuvxy = FF(x,M,c(.U,.V,.X,.Y))
S0[.Mld,] = M$q*(1-M$r) * (fuvx - fuvxy)
fuvxz = FF(x,M,c(.U,.V,.X,.Z))
S0[.Svr,] = M$q*M$r * (fuvx - fuvxz)
S0[.ResS,] = M$q* ( M$r*fuvxz + (1-M$r) * fuvxy)
S0[.ResA,] = (1-M$q) * fuvw

p * S1 + (1-p) * S0
}

```

Computation of state probabilities by simulation

rCovid()

A function to simulate the infection states at times x using model M , infected with probability p at time 0.

```

rCovid = function(x,p,M)
{
  G = rgamma(length(M$shape),M$shape,M$rate)

  sus = G[.U]
  if (runif(1) < p)
    sus = 0

  if (runif(1) < M$q)
  {
    s = rep(.ResS,length(x))
    if (runif(1) < M$r)
    {
      s[x < sus + G[.V] + G[.X] + G[.Z]] = .Svr
    }
    else
    {
      s[x < sus + G[.V] + G[.X] + G[.Y]] = .Mld
    }
  }
}

```

```

    }
    s[x < sus + G[.V] + G[.X]] = .Pre
  }
  else
  {
    s = rep(.ResA,length(x))
    s[x < sus + G[.V] + G[.W]] = .Asy
  }

  s[x < sus + G[.V]] = .Lat
  s[x < sus] = .Sus

  s
}

```

asStateMatrix()

A function to convert the vector of infections states into a 0/1 matrix of state probabilities.

```

asStateMatrix = function(y)
{
  s = matrix(0,nrow=.NStates,ncol=length(y))
  #for (j in 1:ncol(s))
  # s[y[j],j] = s[y[j],j]+1
  for (i in 1:nrow(s))
    s[i,y==i] = s[i,y==i]+1
  s
}

```

SimStateProbs()

A function to calculate the state probabilities for a vector of times **x**, using the parameters in the model **M**, with probability **p** that a transmission occurred at time 0, based on **sims** individual state simulations.

```

SimStateProbs = function(x,p,M,sims=10000)
{
  s = matrix(0,nrow=.NStates,ncol=length(x))
  for (i in 1:sims)
    s = s+asStateMatrix(rCovid(x,p,M))
  s/sims
}

```

Conditioning events

Vectors and functions specifying vectors that give the probabilities of events given the underlying states. Used to calculate, by Bayes rule, the probabilities of the underlying states given the observed events. The events include:

- **Unconditional**: no observation.
- **NoSymptoms**: no symptoms are observed.
- **Infected**: the individual is infected.
- **NegPcrTest()**: a NEGATIVE PCR test with specified sensitivity and specificity is observed.
- **PosPcrTest()**: a POSITIVE PCR test with specified sensitivity and specificity is observed.
- **NegAntigenTest()**: a NEGATIVE antigen test with specified sensitivity and specificity is observed.
- **PosAntigenTest()**: a POSITIVE antigen test with specified sensitivity and specificity is observed.

```
Unconditional = c(1,1,1,1,1,1,1,1)
NoSymptoms = c(1,1,1,1,0,0,0,1)
Infected = c(0,1,1,1,1,1,0,0)

NegPcrTest = function(sens,spec=1)
{
  c(spec,spec,1-sens,1-sens,1-sens,1-sens,1-sens,1-sens)
}

PosPcrTest = function(sens,spec=1)
{
  1-c(spec,spec,1-sens,1-sens,1-sens,1-sens,1-sens,1-sens)
}

NegAntigenTest = function(sens,spec=1)
{
  c(spec,spec,1-sens,1-sens,1-sens,1-sens,spec,spec)
}

PosAntigenTest = function(sens,spec=1)
{
  1-c(spec,spec,1-sens,1-sens,1-sens,1-sens,spec,spec)
}
```

ProbInfected()

A function to calculate the probability that an individual is infected at a vector of times \mathbf{x} following exposure, where the probability of transmission is \mathbf{p} , the SARS-CoV-2 model is specified by the structure \mathbf{M} . The probabilities are conditional given the observations specified by the vector **observation**.

```
ProbInfected = function(probs,observation)
{
    bot = observation %*% probs
    top = (Infected * observation) %*% probs
    top/bot
}
```

ConditionalStateProbs()

A function to calculate the conditional state probabilities for an individual at a vector of times \mathbf{x} following exposure, where the probability of transmission is \mathbf{p} , the SARS-CoV-2 model is specified by the structure \mathbf{M} . The probabilities are conditional given the observations specified by the vector **observation**.

```
ConditionalStateProbs = function(probs,observation)
{
    y = probs
    for (j in 1:length(y[,]))
    {
        y[,j] = y[,j]*observation
        if (sum(y[,j] > 0))
            y[,j] = y[,j]/sum(y[,j])
        else
            y[,j] = 0*y[,j]
    }
    y
}
```

PostProbTransmission()

A function to calculate the posterior probability that a transmission occurred at time 0, given an observation specified by **observation**. The prior probability of transmission is specified by \mathbf{p} .

```
PostProbTransmission = function(P0,P1,observation,p)
{
    q1 = observation %*% P1
}
```



```

    q0 = observation %*% P0
    q1 * p / (q1 * p + q0 * (1-p))
}

```

Functions to plot output

Functions to plot a pile up of state probabilities

First a function to draw a frame suitable for plotting probabilities.

```

frame = function(x,xl="Days from exposure",yl="Probability",ymin=0,ymax=1)
{
    plot(x,x,type="n",ylim=c(ymin,ymax), ylab=yl, xlab=xl)
}

```

Then a function to find the first index into a vector where the value is the maximum of the vector.

```

maximizer = function(x)
{
    (1:length(x))[x==max(x)][1]
}

```

Then a function to draw the pileup given the vector of times, the state probabilities, some colours and labels.

```

skeletonPileup = function(x,ss,stCols,stNames,ymin=0,ymax=1)
{
    frame(x,yl="Cumulative probability",ymin=ymin,ymax=ymax)
    s = apply(ss,2,cumsum)
    lns = rev(1:length(s[,1]))
    for (i in lns)
    {
        lines(x,s[i,])
        polygon(c(x,max(x),0),c(s[i,],0,0),col=stCols[i],border=NA)
    }

    for (i in lns)
    {
        zz = ss[i,]
        if (i > 1)
            z = s[i-1,]
        else
            z = rep(0,length(zz))
    }
}

```

```

        if (max(zz) > 0)
        {
            whj = maximizer(zz)
            whx = x[whj]
            whx = min(c(whx,0.90*max(x)))
            whx = max(c(whx,0.05*max(x)))
            why = z[whj] + zz[whj]/2
            text(whx,why,stNames[i])
        }
    }
}

```

A function to draw a pileup of all the states.

```

pileUp = function(x,ss,ymin=0,ymax=1)
{
    skeletonPileup(x,ss,stateCols,stateNames,ymax=ymax,ymin=ymin)
}

```

A function to draw a pileup with some states combined.

```

pileup = function(x,sss,opt=2,ymin=0,ymax=1)
{
    ss = matrix(0,ncol=ncol(sss),nrow=nrow(sss)-2)

    if (opt == 2)
    {
        stCols = c("green","yellow","pink","orange","red","cyan")
        stNames = c("Sus","Lat","Asy","Pre","Sym","Res")
        ss[1:4,] = sss[1:4,]
        ss[5,] = sss[5,] + sss[6,]
        ss[6,] = sss[7,] + sss[8,]
    }

    if (opt == 1)
    {
        stCols = c("yellow","pink","orange","red","cyan","green")
        stNames = c("Lat","Asy","Pre","Sym","Res","Sus")
        ss[1:3,] = sss[2:4,]
        ss[4,] = sss[5,] + sss[6,]
        ss[5,] = sss[7,] + sss[8,]
        ss[6,] = sss[1,]
    }
}

```

```
skeletonPileup(x,ss,stCols,stNames,ymax=ymax,ymin=ymin)  
}
```