APPENDICES FOR:

A Study of the Effects of Binary Motion on the Detection of Radio Pulsars

*by Owen Giersch*

# Appendix 1: Program Psuedo Code

Type Declarations

**Type TOrbitElements**

m1 as double

m2 as double

a as double

e as double

omega as double

i as double

tau as double

T as double

n as double

**end type**

### **MAIN**

integer mpierror, mpisize, mpirank

integer mpistatus(MPI\_STATUS\_SIZE)

integer tag

string inputfile

integer numsys,sys

TOrbitElements oestore() (Dynamically Allocated)

integer maxe(),maxom() (Dynamically Allocated)

string filename(:) (Dynamically Allocated)

integer arr,maxarr,modarr,start

double precision spec(), phase(), det() (Dynamically Allocated)

string sysnotes() (Dynamically Allocated)

integer ind2

integer numpoints

TOrbitElements OE

double precision pi

double precision AU,S0

integer i, j, k, l, counter

double precision deltae, deltaom

double precision deltat, deltaf, sigfreq

integer minf, maxf, absminf, absmaxf

logical success

integer tminf, tmaxf, tarr

string datafile, detfile, axisfile, logfile

double precision starttime,stoptime

double precision maxpow

constant ind2=19

constant numpoints=2^ind2

constant pi=3.1415926535897933

constant AU=1.49d11, S0=1.9891d30

initialise Message Passing Interface (MPI)

set mpisize to number of CPU’s in MPI communicator

set mpirank to rank of CPU for this instance

if mpirank=0 then

Ask user to enter a file that contains pulsar data

read inputfile from user

get current time and store in starttime

open inputfile

read numsys from inputfile

end if

broadcast numsys to all CPU’s

dimension arrays: oestore(1 to numsys), maxe(1 to numsys) maxom(1 to numsys), filename(1 to numsys))

if mpirank=0 then

for sys=1, numsys

read from input file: OEstore(sys).m1, oestore(sys).m2, oestore(sys).a, oestore(sys).i, maxe(sys), maxom(sys), filename(sys)

end for

close inputfile

end if

for sys=1 to numsys

broadcast oestore(sys).m1 to all CPU’s

broascast oestore(sys).m2 to all CPU’s

broadcast oestore(sys).a to all CPU’s

broadcast oestore(sys).i to all CPU’s

end for

broadcast all maxe’s to all CPU’s

broadcast all maxom’s to all CPU’s

set maxarr to the value of of first element in maxom

for sys=1 to numsys

find largest maxom and stick in maxarr

end for

maxarr=maxarr+1

modarr = maxarr MOD mpisize

if mpirank<modarr then

maxarr=(maxarr/mpisize)+1

else

maxarr=maxarr/mpisize

end if

Dimension Arrays: spec(1 to numpoints/2, 0 to maxarr-1), phase(1 to numpoints/2, 0 to maxarr-1), det(0 to maxarr-1), sysnotes(0 to maxarr-1))

for sys=1 to numsys

copy Oestore(sys) to OE

OE.m1=oestore(sys).m1\*S0 (mass in kg)

OE.m2=oestore(sys).m2\*S0 (mass in kg)

OE.a=oestore(sys).a\*AU (distance in AU)

OE.i=oestore(sys).i

OE.e=0

OE.omega=0

maxpow =0

call initialiseparameters(OE)

deltat=OE.T/numpoints

deltaf=1.d0/OE.T

sigfreq=DBLE(numpoints)\*deltaf/4.d0

(Note that deltat, deltaf and sigfreq are the same for each system)

deltae=1.d0/maxe(sys)

deltaom=2.d0\*pi/maxom(sys)

absminf=numpoints/2

absmaxf=0

arr=maxom(sys)+1

modarr=arr MOD mpisize

if mpirank<modarr then

arr=(arr/mpisize)+1

start=mpirank\*arr

else

arr=(arr/mpisize)

start=mpirank\*arr+modarr

end if

if mpirank=0 then

datafile=filename(sys) + ".dat"

detfile=filename(sys) + ".det"

axisfile=filename(sys) + ".axs"

logfile=filename(sys) + ".log"

open datafile

open detfile

open axisfile

open logfile

end if

for i=0 to maxe(sys)

minf=numpoints/2

maxf=0

OE.e=i\*deltae

counter=0

for j=start to start+arr-1

OE.omega=j\*deltaom

call initialiseparameters(OE)

call calcspecdata(spec(1,counter),phase(1,counter),

det(counter),sysnotes(counter),minf,maxf,OE,sigfreq, deltat, success,numpoints,ind2)

if (mpirank==0) then

stoptime=mpi\_wtime()

PRINT OE.e,OE.omega,det(counter),stoptime-starttime,counter

end if

counter=counter+1

end for

if mpirank <> 0 then

send minf to CPU 0

send maxf to CPU 0

send arr to CPU 0

send spec to CPU 0

send phase to CPU 0

send det to CPU 0

send sysnotes to CPU 0

else

for j=1,mpisize-1

receive minf from CPU j and store in tminf

receive maxf from CPU j and store in tmaxf

if tminf<minf then minf=tminf

if tmaxf>maxf then maxf=tmaxf

end for

if minf<absminf then absminf=minf

if maxf>absmaxf then absmaxf=maxf

tarr=arr

write minf,maxf to datafile

convert current m1 and m2 back to solar masses and a to astronomical units

OE.m1=OE.m1/S0

OE.m2=OE.m2/S0

OE.a=OE.a/AU

counter=0

for j=0 to mpisize-1

for k=0 to arr-1

for l=minf to maxf

write spec(l,k),phase(l,k) to datafile

end for

write det(k) to detfile

if det(k)>maxpow then maxpow=det(k)

OE.omega=counter\*deltaom

write 'm1=',OE.m1,'m2=',OE.m2,'a=',OE.a,'om=',OE.omega,'i=',OE.i,'e=',OE.e,'f=', sigfreq to logfile

write (14,\*) sysnotes(k) to logfile

counter=counter+1

end for

if j<mpisize-1 then

receive arr from CPU j+1

receive spec from CPU j+1

receive phase from CPU j+1

receive det from CPU j+1

receive sysnotes from CPU j+1

end if

end for

! reset m's and a to SI units

OE.m1=OE.m1\*S0

OE.m2=OE.m2\*S0

OE.a=OE.a\*AU

arr=tarr

if minf<absminf then absminf=minf

if maxf>absmaxf then absmaxf=maxf

end if

end for

if mpirank=0 then

OE.m1=OE.m1/S0

OE.m2=OE.m2/S0

OE.a=OE.a/AU

maxpow=maxpow\*0.25

WRITE 'm1=',OE.m1,'m2=',OE.m2,'a=',OE.a,'i=',OE.i,'f=',sigfreq,'max pow=',maxpow to axisfile

write absminf,absmaxf,deltaf to axisfile

write maxe(sys),deltae to axisfile

write maxom(sys),deltaom to axisfile

close datafile

close detfile

close logfile

close axisfile

end if

end for

Finalise mpi

**end program MAIN**

**SUBROUTINE calcspecdata(power,phase,det,sysnotes,minf,maxf,OE,sigfreq,deltat,success,numpoints,ind2)**

Calcspecdata models the orbit of a binary pulsar and calculates the power spectrum, phase and detectability

Import Variables:

TOrbitElements OE

INTEGER numpoints

DOUBLE PRECISION sigfreq

DOUBLE PRECISION deltat

INTEGER ind2

Export Variables:

DOUBLE PRECISION power(1:numpoints/2)

DOUBLE PRECISION phase(1:numpoints/2)

DOUBLE PRECISION det

LOGICAL success

string sysnotes

integer minf,maxf

Local Variables:

INTEGER i

DOUBLE PRECISION pi

DOUBLE PRECISION Tb

DOUBLE PRECISION signal(0:numpoints-1)

COMPLEX\*16 fourier (1:numpoints)

DOUBLE PRECISION sigpower,specpower

LOGICAL orbitcheck

DOUBLE PRECISION prat

integer tminf,tmaxf

DOUBLE PRECISION a,b

### Start Subroutine:

pi=3.1415926535897933

tminf=minf

tmaxf=maxf

success=false

det=0

call InitialiseParameters(OE)

Tb=CalcOrbitDecay(OE)

if Tb>10 million years then

call GetOrbit(signal,OE,numpoints,deltat,sigfreq,orbitcheck)

if orbit calculated ok (orbitcheck=true then

sigpower=0

for i=0 to numpoints-1

sigpower=sigpower+signal(i)^2 (power in the signal)

fourier(i+1)=complex(signal(i),0.d0)

end for

sigpower=sigpower/numpoints

call fft(fourier,ind2,0) (calculate fourier spectrum)

specpower=0

for i=1 to numpoints/2

power(i)=ABS(fourier(i))^2 (Power in the spectrum)

specpower=specpower+power(i)+ABS(fourier(i+numpoints/2))^2

a=re(fourier(i))

b=im(fourier(i))

if a=0 and b=0 then

phase(i)=0

else

phase(i)=complex tangent (b,a)

end if

if power(i)>10^-9 then

if i<tminf then tminf=i

if i>tmaxf tmaxf=i

if power(i)>det det=power(i)

end if

end for

det=det/0.25 (divide det total power in spectrum)

prat=specpower/sigpower

if prat>0.98 and prat<1.02 then

sysnotes="Orbit OK"

minf=tminf

maxf=tmaxf

success=true

else

sysnotes="Rounding errors"

end if

else

sysnotes='Too many undefined points on orbit'

end if

else

sysnotes='Orbit Decay too Fast' ! set sysnotes to Orbit Decay too Fast

end if

if success = false then det=0.d0

**end subroutine**

**Subroutine GetOrbit(signal,OE,numpoints,deltat,sigfreq,orbitcheck)**

getorbit calculates the signal for 1 complete orbit of the binary pulsar

Import Variables:

TOrbitElements OE

INTEGER numpoints

double precision deltat

DOUBLE PRECISION sigfreq

LOGICAL orbitcheck

Export Variables:

DOUBLE PRECISION signal(0 to numpoints-1)

Local Variables:

double precision G

double precision pi

double precision nu

double precision EA

double precision M

INTEGER j

double precision r

DOUBLE PRECISION s

DOUBLE PRECISION z

DOUBLE PRECISION c

DOUBLE PRECISION tphase

DOUBLE PRECISION T1

DOUBLE PRECISION t0

Start Subroutine:

G=6.6726\*10^-11

pi=3.141592653589793

c=299792458

nu=0.d0

EA=0.d0

M=0.d0

t0=0.d0

j=0

while t0<deltat and j<numpoints

s=t0+j\*deltat

M=OE.n\*(s-OE.tau)

EA=solveE(M+OE.e\*sin(M),OE.e,M)

r=OE.a\*(1-OE.e\*cos(EA))

nu=2\*ATAN(SQRT(-(1+OE.e)/(OE.e-1))\*TAN(EA/2))

z=r\*sin(nu+OE.omega)\*sin(OE.i) ! calculate z position of pulsar

if not (OE.omega+nu=pi/2 and OE.i=pi/2) then

T1=(G\*OE.m2/(OE.a\*c^2))\*((OE.m1+2.d0\*OE.m2)/(OE.m1+OE.m2))\*(OE.e\*OE.T/(2\*pi))

tphase=(z/c)+T1\*SIN(EA)

signal(j)=SIN(2\*pi\*sigfreq\*(s-tphase))

j=j+1

else

j=0

t0=t0+deltat/4

end if

end while

if t0 < deltat then

orbitcheck = true

else

orbitcheck = false

end if

**end subroutine**

**double precision Function SolveE(EA,e,M)**

Solves the eccentric anomaly using the Newton-Raphson Method

Import Variables:

double precision EA

double precision e

double precision M

Export Variables:

None

Local Variables:

double precision En

LOGICAL check

Start Function:

check=true

WHILE check=true

En=EA

EA=En+(M-En+e\*sin(En))/(1-e\*cos(En))

check= not (ABS(En-EA)<4\*10^-8)

end while

SolveE=EA

**end Function**

**Subroutine InitialiseParameters(OE)**

This subroutine initialises the parameters stored in OE

Import Variables:

None

Export Variables:

TYPE(TOrbitElements) :: OE

Local Variables:

double precision G

double precision pi

double precision nu

double precision EA

double precision M

double precision time

Start Subroutine:

G=6.6726\*10^-11

pi=3.141592653589793

OE.T=sqrt(4\*pi^2\*OE.a^3/(G\*(OE.m1+OE.m2)))

if OE.e <> 1 then

EA=-2\*arctan(tan(0.5\*OE.omega)/sqrt(-(1+OE.e)/(OE.e-1))

else

EA=0

end if

M=EA-OE.e\*sin(EA)

OE.n=sqrt(G\*(OE.m1+OE.m2)/(OE.a^3))

OE.tau=(-EA+OE.e\*sin(EA))/OE.n

**end subroutine**

**Double Precision Function CalcOrbitDecay(OE)**

This function calculates the time that a system will decay in.

Import Variables:

TorbitElements OE

Export Variables:

None

Local Variables:

double precision G

double precision c

double precision f

double precision ITB

DOUBLE PRECISION m

DOUBLE PRECISION mu

Start Function:

G=6.6726\*10^-11

c=299792458

m=OE.m1+OE.m2

mu=(OE.m1\*OE.m2)/m

if OE.e=1 then

calcorbitdecay=0.d0

else

f=(1+(73/24)\*OE.e^2+(37 /96)\*OE.e^4)/((1-OE.e^2)^(7/2))

ITB=f\*(96/5)\*(G^3)\*OE.m1\*(OE.m2^5)/((m^3)\*(OE.a^4)\*(c^5))

CalcOrbitDecay=1.d0/ITB

end if

**end function**

# Appendix 2: Output File Formats

## 2.1 Axis File Output (\*.axs)

*Fmin* *Fmax* *f*

*emax* *e*

*max* **

*Fmin* is an integer specifying the lowest frequency element stored in the data file.

*Fmax* is an integer highest frequency element stored in the file.

*f* is a double precision number that is the step size for frequency.

*emax* is an integer specifying how many eccentricities have been modelled.

*e* is a double precision number that is the step size for eccentricity.

*max* is an integer specifying how many arguments of periastron to model for each eccentricity.

** is a double precision number that is the step size for argument of periastron.

The frequency range in the data file will be *Fmin**f* to *Fmax**f* with an increment size of *f*. Individual spectra in the data file will have a smaller range than that specified by *Fmin* and *Fmax*.

The eccentricity range in the data file and detection file is 0 to *emax**e*.

The argument of periastron range in the data file and detection file is 0 to *max***

## 2.2 Detection File Output (\*.det)

**0,0)

**0,1)

**0,2)

⇩

**0,*max*)

**(1,0)

⇩

**1,*max*)

**2,0)

⇩

**2,*max*)

⇩

***emax*,*max*)

***e* is the detection efficiency for eccentricity *e* and argument of periastron **. A simple algorithm to retrieve detection values is as follows:

for i=0 to emax

for j=0 to omegamax

read gamma(i,j)

end for

end for

## 2.3 Data File Output (\*.dat)

*fmin*(0) fmax(0)

*P*(0,0,0 ) **0,0,0)

*P*(0,0,1 ) **0,0,1)

⇩

*P*(0,0,*n*(0)) **0,0,*n*(0))

*P*(0,1,0) **0,1,0)

*P*(0,1,1) **0,1,1)

⇩

*P*(0,1,*n*(0)) **0,1,*n*(0))

⇩

P(0,*max*,*n*(0)) **0,*max*,*n*(0))

*fmin*(1) *fmax*(1)

*P*(1,0,0) **1,0,0)

⇩

P(1,*max*,*n*(1)) **1,*max*,*n*(1))

*fmin*(1) *fmax*(1)

⇩

P(*emax*-1,*max*-1,*n*(*emax*-1)) ** (*emax*-1,*max*-1,*n*(*emax*-1))

*fmin*(*emax*) fmax(*emax*)

*P*(*emax*,*max*,0) ** *emax*,*max*,0)

*P*(*emax*,*max*,1) ** *emax*,*max*,1)

⇩

*P*(*emax*,*max*,*n*(*emax*)) ** *emax*,*max*,*n*(*emax*))

*P*(*e*,**,**) and **(*e*,**,**) are the power and phase for eccentricity *e*, argument of periastron ** and frequency *fmin*+**-1. *n*(*e*) is simply given by:

 (9.3.1)

A simple algorithm to retrieve data values is as follows:

for i=0 to emax

read fmin(i),fmax(i)

n(i)=(fmax(i)-fmin(i))+1

for j=0 to omegamax

for k=0 to n(i)

read power(i,j,k),phase(i,j,k)

end for

end for

end for

## 2.4 Log File Output (\*.log)

sysspec(0,0)

notes(0,0)

sysspec(0,1)

notes(0,1)

⇩

sysspec(0,*max*)

notes(0,*max*)

sysspec(1,0)

notes(1,0)

⇩

sysspec(1,*max*)

notes(1,*max*)

sysspec(2,0)

notes(2,0)

⇩

sysspec(*emax*,*max*)

notes(*emax*,*max*)

sysspec*e,* is a string that contains the orbital parameters for the orbit of eccentricity *e* and argument of periastron **.

notes*e,* is a string that contains notes on whether the orbit of eccentricity *e* and argument of periastron ** was calculated succesfully.

A simple algorithm to retrieve these strings is as follows:

for i=0 to emax

for j=0 to omegamax

read sysspec(i,j)

read notes(i,j)

end for

end for

# Appendix 3: Input File Format

*n*

*m*1(1), *m*2(1), *a*(1), *i*(1), *emax*(1), *max*(1), *filename*(1)

*m*1(2), *m*2(2), *a*(2), *i*(2), *emax*(2), *max*(2),*filename*(2)

⇩

*m*1(*n*), *m*2(*n*), *a*(*n*), *i*(*n*), *emax*(*n*), *max*(*n*),*filename*(*n*)

n is an integer which specify how many systems to model.

*m*1(*k*) is the pulsar mass for system *k*

*m*2(*k*) is the companion mass for system *k*

*a*(*k*) is the semi-major axis for system *k*

*i*(*k*) is the inclination for system *k*

*emax*(*k*)+1 is the total number of eccentricities to be modelled for system *k*

*max*(*k*)+1 is the number of arguments of periastron to model for eccentricity in system *k*

*filename*(*k*) is the output filename without an extension for system *k*. The program will append appropriate extensions for each of the output files.

The eccentricity for a given orbit is found by:

 5.3.1

where k is an integer from 0 to *emax*.

Similarly for argument of periastron:

 5.3.2

Thus *e* will have values of 0 to 1 and ** values of 0 to 2**.