ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Διδακτικής της Τεχνολογίας και Ψηφιακών Συστημάτων

Μελέτη Επίδοσης Αναμεταδοτών για Ψηφιακές Διαμορφώσεις σε Ασύρματα Κανάλια Διάδοσης

Νίκος Σ. Τσαπέκος

Μεταπτυχιακή Διπλωματική Εργασία

Ιανουάριος 2012

Περίληψη

Η παρούσα εργασία αποτελεί μια μελέτη επίδοσης ενός dual-hop ασύρματου συστήματος επικοινωνιών το οποίο εξυπηρετείται από αναμεταδότες (Relays). Εξετάζονται οι περιπτώσεις και των δύο ειδών αναμεταδοτών: Regenerative Relays και Non Regenerative Relays.

Πιο συγκεκριμένα, στο πρώτο κεφάλαιο γίνεται μια εισαγωγή για τα χαρακτηριστικά του ασύρματου καναλιού διάδοσης. Παρουσιάζονται οι συναρτήσεις πυκνότητας πιθανότητας (PDF) για την κατανομή Nakagami-m, την κατανομή Rayleigh, η οποία αποτελεί μια ειδική περίπτωση της Nakagami-m κατανομής, η κατανομή Rice και η κατανομή Generalized-K η οποία είναι περίπτωση Composite Fading.

Στη συνέχεια, στο δεύτερο κεφάλαιο μελετάται η μέθοδος Moment Generating Function (MGF). Στη θεωρία των πιθανοτήτων και στη στατιστική η moment generating function μιας τυχαίας μεταβλητής αποτελεί έναν εναλλακτικό ορισμό για την κατανομή της πιθανότητας. Οι προσομοιώσεις αφορούν σε ψηφιακές διαμορφώσεις BPSK και QPSK ενώ τα περιβάλλοντα διάδοσης χαρακτηρίζονται από τις κατανομές Nakagami-m και Rayleigh.

Στο τρίτο κεφάλαιο παρουσιάζονται τα αποτελέσματα της επίδοσης των αναμεταδοτών. Καταρχάς, μελετήθηκαν οι Non Regenerative CSI (Channel State Information) Relays οι οποίοι μελετούν τα χαρακτηριστικά του καναλιού και με βάση αυτά αποφασίζουν το Gain με το οποίο θα πολλαπλασιάσουν το εισερχόμενο σήμα. Ενδεικτικά παρουσιάζονται και οι τιμές του Gain που επιλέχθηκαν από τον αναμεταδότη για διάφορες περιπτώσεις. Εν συνεχεία, εξετάζεται η περίπτωση των Non Regenerative Relays with Fixed Gain, όπου γίνεται μια σύγκριση όσον αφορά στην τιμή του Gain που επιλέγεται. Στο τέλος του κεφαλαίου αυτού, παρουσιάζονται τα αποτελέσματα για τους Regenerative Relays σε σύγκριση με τους Non Regenerative CSI Relays. Οι προσομοιώσεις έγιναν σε περιβάλλοντα διαλείψεων Nakagami-m και Generalized-K, ενώ οι ψηφιακές διαμορφώσεις που χρησιμοποιήθηκαν ήταν οι QPSK και 16-QAM. Η επίδοση των αναμεταδοτών μελετάται ως προς τους ρυθμούς σφαλμάτων SER και BER. Για κάθε γραφική παράσταση ακολουθεί ο κώδικας matlab που την υλοποιεί, ενώ στο τέλος την εργασίας παρατίθεται με τη μορφή παραρτήματος συνολικά ο κώδικας για όλες τις γραφικές παραστάσεις.

Περιεχόμενα

Κατάλογος Σχημάτων

1. ΚΑΝΑΛΙ ΔΙΑΔΟΣΗΣ

1.1 Rayleigh Fading

Οι διαλείψεις Rayleigh είναι ένα στατιστικό μοντέλο, το οποίο μελετάει την επίδραση του περιβάλλοντος διάδοσης σε ένα σήμα. Σύμφωνα με το μοντέλο αυτό, το μέγεθος του σήματος που διαπερνά ένα μέσο διάδοσης (κανάλι επικοινωνίας) μεταβάλλεται τυχαία, ή φθίνει, σύμφωνα με την κατανομή Rayleigh – μια συνιστώσα που είναι αποτέλεσμα του αθροίσματος δύο μη συσχετισμένων τυχαίων Gaussian μεταβλητών.

Το μοντέλο Rayleigh χρησιμοποιείται για να περιγράψει διαλείψεις για τροποσφαιρικά και ιονοσφαιρικά σήματα διάδοσης, καθώς και αστικά περιβάλλοντα έντονα καλυμμένα με κτίρια. Επιπρόσθετα, το συγκεκριμένο μοντέλο είναι το πλέον κατάλληλο για περιπτώσεις όπου δεν υπάρχει απευθείας συνιστώσα (line of sight) μεταξύ πομπού και δέκτη.

Το μοντέλο Rayleigh είναι κατάλληλο να περιγράψει περιβάλλον στο οποίο μεσολαβούν πολλά εμπόδια που αδυνατίζουν το σήμα μέχρι αυτό να φτάσει στον δέκτη. Σύμφωνα με το Κεντρικό Οριακό Θεώρημα, αν υπάρχει αρκετή διασπορά, η κρουστική απόκριση του καναλιού θα μοντελοποιείται σαν μια διαδικασία Gauss, ανεξάρτητα από την κατανομή των επιμέρους συνιστωσών. Αν δε, δεν υπάρχει κάποια κυρίαρχη συνιστώσα σε αυτή τη διασπορά, τότε σε μια τέτοια περίπτωση η μέση τιμή θα είναι μηδενική, μ=0 και η φάση θα είναι ομοιόμορφα κατανεμημένη μεταξύ 0 και 2π. Η περιβάλλουσα της απόκρισης του καναλιού ακολουθεί επομένως, κατανομή Rayleigh. Η συνάρτηση πυκνότητα πιθανότητας της κατανομής Rayleigh είναι:

$$p(a) = \frac{a}{\sigma^2} e^{\left(\frac{-a^2}{2\sigma^2}\right)}, \ \alpha \geq 0 \qquad (1.1)$$

Συχνά, το κέρδος και τα στοιχεία της φάσης της παραμόρφωσης του καναλιού παρουσιάζονται σαν ένας μιγαδικός αριθμός. Σε αυτή την περίπτωση το μοντέλο Rayleigh προϋποθέτει ότι το πραγματικό και το φανταστικό μέρος της απόκρισης αυτής μοντελοποιούνται από ανεξάρτητες και πανομοιότυπες κατανομές Gauss με μηδενικό μέσο, έτσι ώστε το πλάτος της απόκρισης να είναι το άθροισμα των δύο μερών.

- **Joint Probability**

Η συνάρτηση πυκνότητας πιθανότητας της συνιστώσας x είναι:

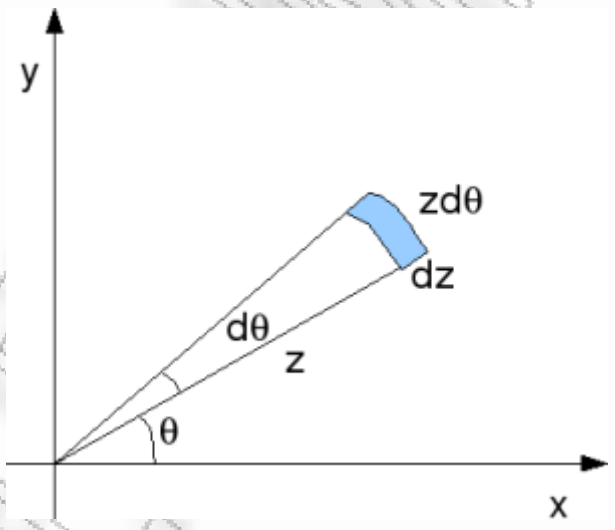
$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(\frac{-x^2}{2\sigma^2}\right)}$ (1.2), ενώ αντίστοιχα της y είναι $p(y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(\frac{-y^2}{2\sigma^2}\right)}$ (1.3).

Η από κοινού πιθανότητα για τις μεταβλητές X που βρίσκεται μεταξύ *x* και *x+dx* και της μεταβλητής Y που βρίσκεται μεταξύ *y* και *y+dy* είναι:

$$P(x \le X + dx, y \le Y + dy) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-\left(x^2+y^2\right)}{2\sigma^2}} dxdy \qquad (1.4).$$

- **Μετατροπή σε πολικές συντεταγμένες**

Δεδομένου ότι *(x,y)* βρίσκονται στο Καρτεσιανό επίπεδο μπορούμε να τις μεταφέρουμε σε πολικές συντεταγμένες *(z,ϑ)*, όπου $Z = \sqrt{X^2 + Y^2}$ και $\Theta = \tan^{-1}\left(\frac{Y}{X}\right)$.



Σχήμα 1.1 *Μετατροπή Καρτεσιανών σε Πολικές Συντεταγμένες*

Η περιοχή του Καρτεσιανού επιπέδου *dxdy* είναι η αντίστοιη *zdzdϑ* στο Πολικό επίπεδο συντεταγμένων. $P(x \le X + dx, y \le Y + dy) = P(z \le Z + dz, \theta \le \Theta + d\theta) \qquad (1.5)$

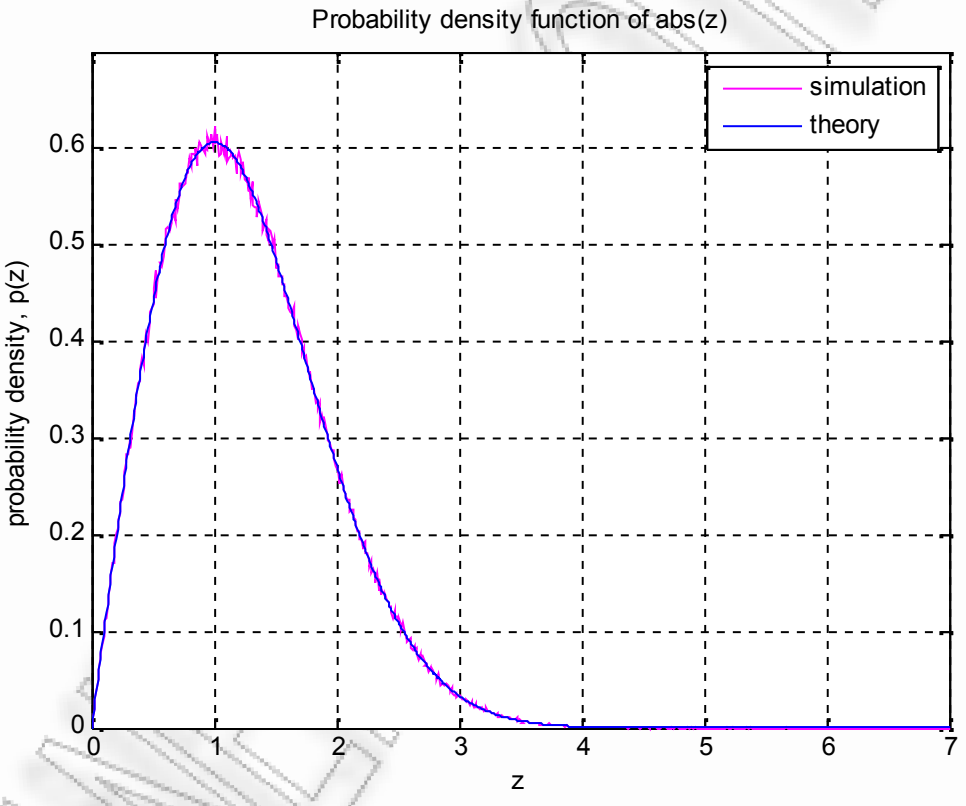
Απλοποιώντας:

$$P(z \le Z + dz, \theta \le \Theta + d\theta) = \frac{1}{2\pi\sigma^2} e^{\frac{-\left(x^2+y^2\right)}{2\sigma^2}} z dz d\theta = \frac{z}{\sigma^2} e^{\frac{-z^2}{2\sigma^2}} dz \frac{1}{2\pi} d\theta \quad (1.6)$$

Συνοψίζοντας, η κοινή Probability Density Function είναι η παρακάτω:

$$p(z,\theta) = \frac{z}{2\pi\sigma^2} e^{\frac{-z^2}{2\sigma^2}} \quad (1.7)$$

με $p(z) = \frac{z}{\sigma^2} e^{\frac{-z^2}{2\sigma^2}}, z \ge 0$ και $p(\theta) = \frac{1}{2\pi}, -\pi \le \theta \le \pi$

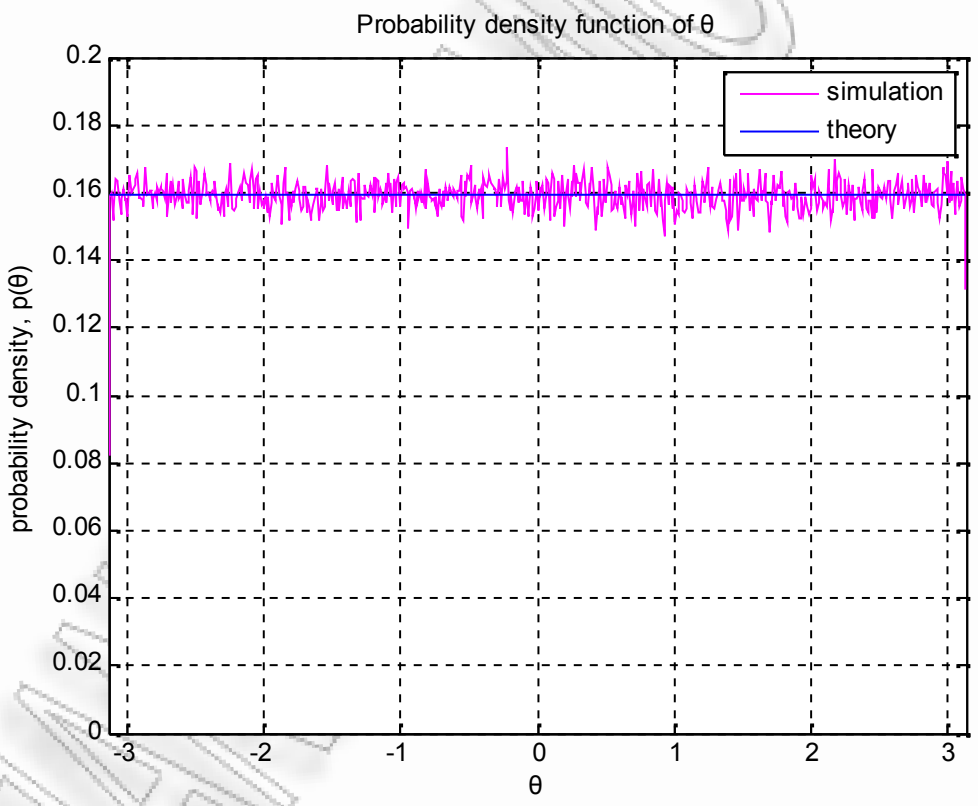


Σχήμα 1.2 *Probability Density Function of Rayleigh distribution*

1.1.1 Matlab script for Rayleigh PDF

```
N = 10^6;
x = randn(1,N); % gaussian random variable, mean 0, variance 1
y = randn(1,N); % gaussian random variable, mean 0, variance 1
```

```
z = (x + j*y);  % complex random variable

%probability density function of abs(z)
zBin = [0:0.01:7];
sigma2 = 1;
pzTheory = (zBin/sigma2).*exp(-(zBin.^2)/(2*sigma2)); % theory
[nzSim zBinSim] = hist(abs(z),zBin); % simulation

plot(zBinSim,nzSim/(N*0.01),'m');
hold on
plot(zBin,pzTheory,'b-')
xlabel('z');
ylabel('probability density, p(z)');
legend('simulation','theory');
title('Probability density function of abs(z)' )
axis([0 7 0 0.7]);
grid on
```



Σχήμα 1.3 *Probability Density of ϑ*

1.1.2 Matlab script for Probability Density of θ

```
% Matlab script for plotting the probability density
% of θ of Rayleigh random variable

close all
clear all
N = 10^6;
x = randn(1,N); % gaussian random variable, mean 0, variance 1
y = randn(1,N); % gaussian random variable, mean 0, variance 1
z = (x + j*y);  % complex random variable

% probability density of theta
thetaBin = [-pi:0.01:pi];
pThetaTheory = 1/(2*pi)*ones(size(thetaBin));
[nThetaSim thetaBinSim] = hist(angle(z),thetaBin); % simulation

plot(thetaBinSim,nThetaSim/(N*0.01),'m');
hold on
plot(thetaBin,pThetaTheory,'b-')
xlabel('θ');
ylabel('probability density, p(θ)');
legend('simulation','theory');
title('Probability density function of θ')
axis([-pi pi 0 0.2])
grid on
```

1.2 Rician Fading

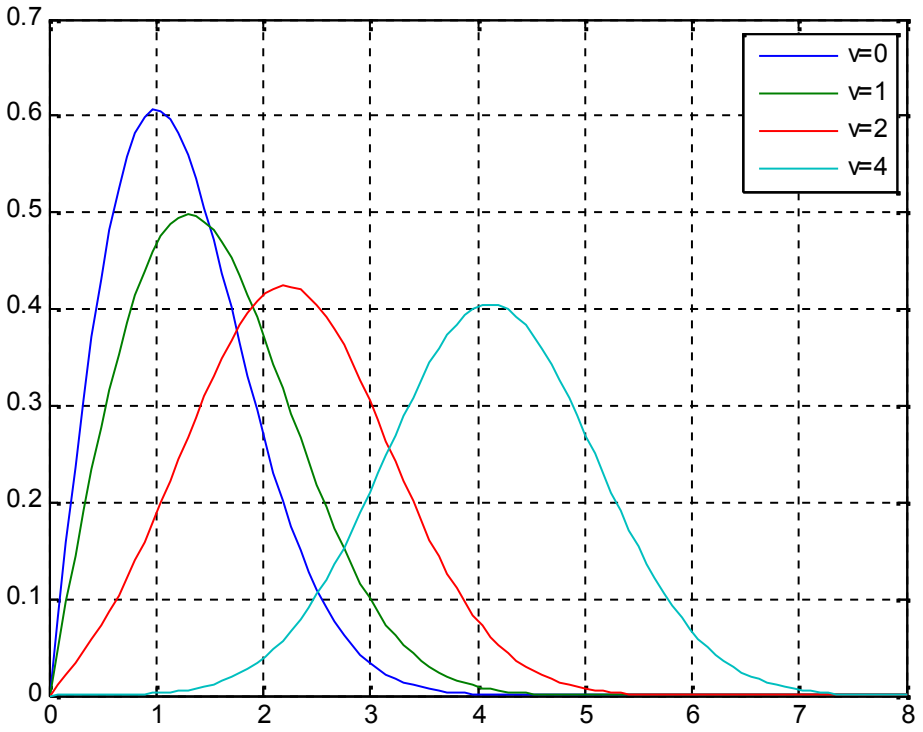
Το μοντέλο Rice είναι ένα στοχαστικό μοντέλο που περιγράφει τη μερική ακύρωση του σήματος από το ίδιο, λόγω πολυδιαδρομικής μετάδοσης. Διαλείψεις Rician υφίστανται όταν τουλάχιστον μία από τις συνιστώσες της πολυδιαδρομικής διάδοσης είναι πολύ ισχυρότερη από τις υπόλοιπες (line of sight). Σε αυτό το μοντέλο το μέγεθος του κέρδους χαρακτηρίζεται από την Rician κατανομή.

Στη θεωρία των πιθανοτήτων η κατανομή Rice είναι η κατανομή της πιθανότητας της απόλυτης τιμής μιας κυκλικής διμετάβλητης κανονικής τυχαίας μεταβλητής με μη μηδενική μέση τιμή. Η συνάρτηση πυκνότητας πιθανότητας της κατανομής Rice είναι η παρακάτω:

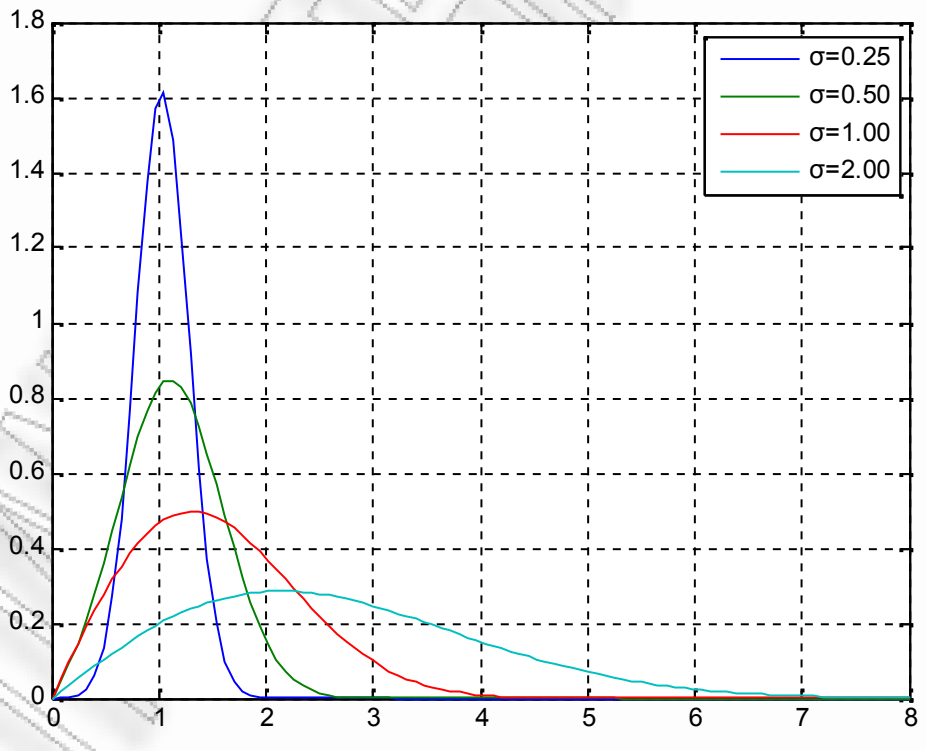
$$f(x|\nu,\sigma) = \frac{x}{\sigma^2} e^{\left(\frac{-(x^2+\nu^2)}{2\sigma^2}\right)} I_0\left(\frac{x\nu}{\sigma^2}\right) \qquad (1.8)$$

όπου $I_0(z)$ είναι η μορφοποιημένη συνάρτηση Bessel πρώτου είδους, μηδενικής τάξης.

Για *v=0* η κατανομή περιορίζεται σε Rayleigh.



Σχήμα 1.4 *Probability density function of Rice distribution with σ=1*



Σχήμα 1.5 *Probability density function of Rice distribution with v=1*

1.2.1 Matlab scripts for Rician PDF

```
function y = ricepdf(x, v, s)
%RICEPDF Rice probability density function (pdf).
%y = ricepdf(x, v, s) returns the pdf of the Rice distribution
%with parameters v and s, evaluated at the values in x.
s2 = s.^2; % (neater below)

try
    y = (x ./ s2) .*...
        exp(-0.5 * (x.^2 + v.^2) ./ s2) .*...
        besseli(0, x .* v ./ s2);
        % besseli(0, ...) is the zeroth order modified Bessel
        % function of the first kind.
    y(x <= 0) = 0;
catch
    error('ricepdf:InputSizeMismatch',...
        'Non-scalar arguments must match in size.');
end

%% The Rician PDF
x = linspace(0, 8, 100);
close;
%subplot(3, 1, 1)
figure;
plot(x, ricepdf(x, 0, 1), x, ricepdf(x, 1, 1),...
     x, ricepdf(x, 2, 1), x, ricepdf(x, 4, 1))
%title('Rice PDF with σ = 1')
legend('v=0', 'v=1', 'v=2', 'v=4')
grid on;
%subplot(3,1,2)
figure;
plot(x, ricepdf(x, 1, 0.25), x, ricepdf(x, 1, 0.50),...
     x, ricepdf(x, 1, 1.00), x, ricepdf(x, 1, 2.00))
%title('Rice PDF with v = 1')
legend('σ=0.25', 'σ=0.50', 'σ=1.00', 'σ=2.00')
```

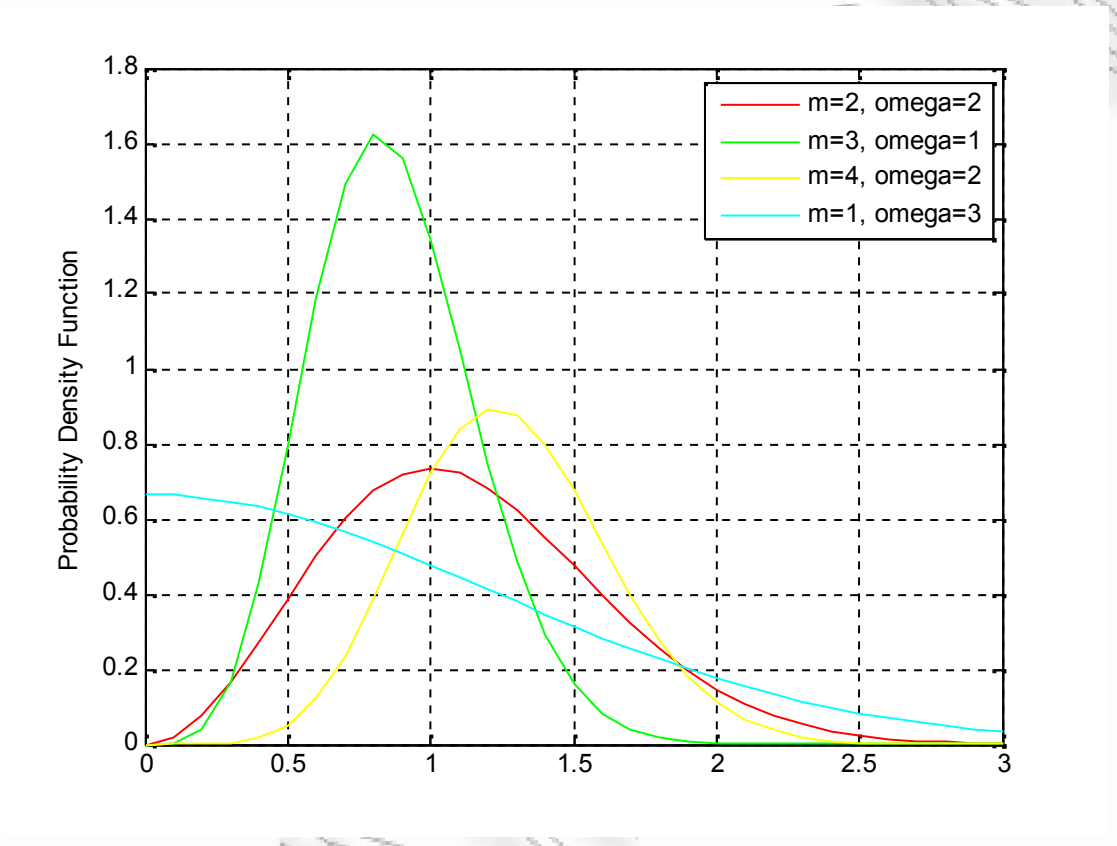
1.3 Nakagami-m Fading

Εκτός από τα μοντέλα Rayleigh και Rician έχουν προταθεί και άλλα που μελετούν την PDF του πλάτους ενός σήματος που υφίσταται διαλείψεις. Ένα τέτοιο μοντέλο είναι αυτό που ακολουθεί την κατανομή Nakagami-m.

Αν η μιγαδική περιβάλλουσα ακολουθεί κατανομή Nakagami-m, τότε η αντίστοιχη στιγμιαία ισχύς ακολουθεί την κατανομή Γάμμα. Η παράμετρος μ, καλείται “shape factor” της κατανομής και στην ειδική περίπτωση που μ=1, οι διαλείψεις περιγράφονται από το μοντέλο Rayleigh με εκθετικά κατανεμημένη στιγμιαία ισχύ. Όσο αυξάνεται το μ, οι διακυμάνσεις της ισχύος του σήματος μειώνονται, συγκριτικά με το μοντέλο του Rayleigh.

Η συνάρτηση πυκνότητας πιθανότητας της κατανομής Nakagami-m είναι:

$$f(x|\mu,\omega)=\frac{2\mu^{\mu}}{\Gamma(\mu)\Omega^{\mu}}x^{2\mu-1}e^{\left(\frac{-\mu}{\omega}x^2\right)} \qquad (1.9)$$



Σχήμα 1.6 *Probability Density Function of Nakagami-m distribution*

1.3.1 Matlab script for Nakagami-m PDF

```
clear;
x=0:0.1:3;

m1=1;
m2=2;
m3=3;
m4=4;
m5=5;
omega1=1;
omega2=2;
omega3=3;

y2=((2.*(m2.^m2))./(1.*(omega2.^m2))).*(x.^(2.*(m2-1))).*(exp(-
(m2.*(x.^2)./omega2)));
```

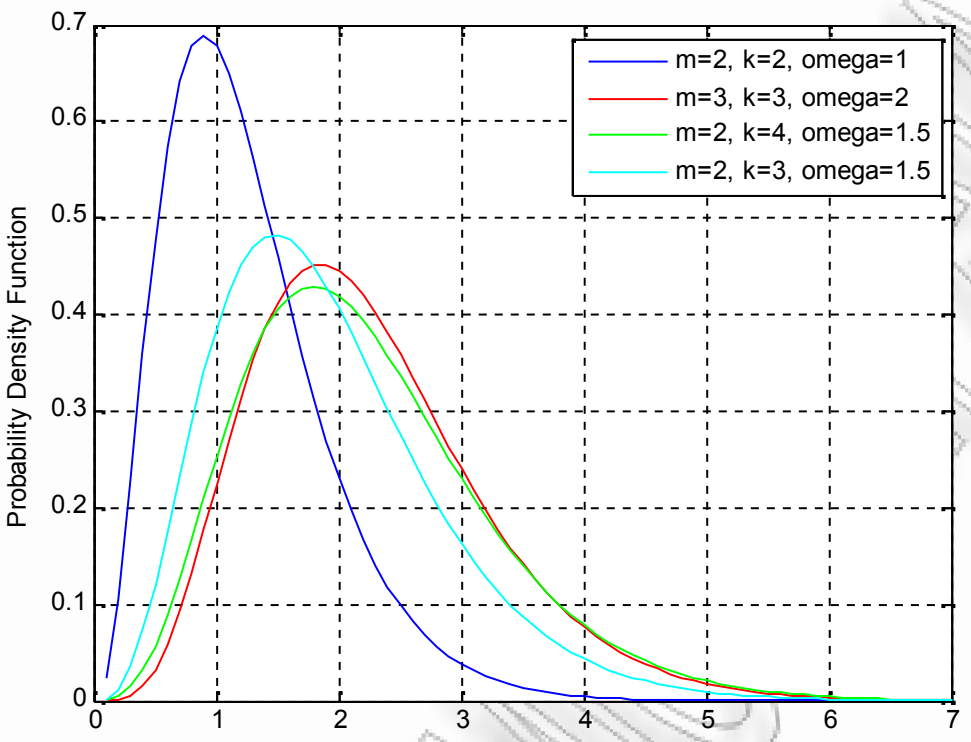
```
y3=((2.*(m3.^m3))./(2.*(omega1.^m3))).*(x.^(2.*(m3-1))).*(exp(-
(m3.*(x.^2)./omega1)));
y4=((2.*(m4.^m4))./(6.*(omega2.^m4))).*(x.^(2.*(m4-1))).*(exp(-
(m4.*(x.^2)./omega2)));
y5=((2.*(m1.^m1))./(1.*(omega3.^m1))).*(x.^(2.*(m1-1))).*(exp(-
(m1.*(x.^2)./omega3)));
```

1.4 Composite Fading

Στα ασύρματα κανάλια τα φαινόμενα των πολυδιαδρομικών διαλείψεων και της επικάλυψης συμβαίνουν ταυτόχρονα, με αποτέλεσμα τις σύνθετες διαλείψεις. Οι διαλείψεις μικρής κλίμακας που οφείλονται στην πολυδιαδρομική διάδοση μοντελοποιούνται από τις κατανομές Rayleigh, Rician και Nakagami-m. Η τελευταία μάλιστα είναι αρκετά γενικευμένη, ώστε να περικλείει την κατανομή Rayleigh σαν μια ειδική περίπτωσή της, αλλά και να προσεγγίζει επαρκώς την κατανομή Rice. Οι διαλείψεις μεγάλης κλίμακας (shadow fading) προσεγγίζονται από τη λογαριθμική κατανομή. Παρόλα αυτά, τα μοντέλα σύνθετων διαλείψεων που βασίζονται σε αυτή την κατανομή δεν καταλήγουν σε πολύ ακριβείς εκφράσεις της συνάρτησης πυκνότητας πιθανότητας της εισερχόμενης ισχύος. Εναλλακτικά, έχει προταθεί η χρήση της κατανομής Γάμμα για τη μοντελοποίηση της μέσης τυχαίας διακύμανσης ισχύος λόγω επικαλύψεων. Υποθέτοντας ότι οι διαλείψεις και οι επικαλύψεις είναι ανεξάρτητες μεταξύ τους και χρησιμοποιώντας το γεγονός ότι η τετραγωνική ρίζα μιας Nakagami-m τυχαίας μεταβλητής ακολουθεί την κατανομή Γάμμα, αναπτύχθηκε μια έκφραση για τη συνάρτηση πυκνότητας πιθανότητας των σύνθετων διαλείψεων της μορφής Γάμμα-Γάμμα PDF (Generalized-K). Η PDF της κατανομής Generalized-K φαίνεται παρακάτω:

$$f_x(x) = \frac{4m^{(k+m)/2}}{\Gamma(m)\Gamma(k)\Omega^{(k+m)/2}} x^{k+m-1} K_{k-m}\left(2\left(\frac{m}{\Omega}\right)^{1/2} x\right) \qquad (1.10)$$

όπου K_{k-m} η μορφοποιημένη συνάρτηση Bessel δεύτερου βαθμού και τάξης $(k-m)$.



Σχήμα 1.7 *Probability Density Function of Generalized-K Distribution*

1.4.1 Matlab script for Generalized-K PDF

```
clear;

x=0:0.1:7;
m=2;
k=2;
omega=1;
m1=3;
k1=3;
omega1=2;
m2=2;
k2=4;
omega2=1.5;
m3=2;
k3=3;
omega3=1.5;

nu=k-m;
z=2.*((m./omega).^(1/2)).*x;
Kei=besselk(nu,z);
nu1=k1-m1;
z1=2.*((m1./omega1).^(1/2)).*x;
Kei1=besselk(nu1,z1);
nu2=k2-m2;
z2=2.*((m2./omega2).^(1/2)).*x;
```

```
Kei2=besselk(nu2,z2);
nu3=k3-m3;
z3=2.*((m3./omega3).^(1/2)).*x;
Kei3=besselk(nu3,z3);

y=(((4.*m.^((k+m)./2))./(1.*1.*omega.^((k+m)./2))).*(x.^(k+m-
1))).*Kei;
y1=(((4.*m1.^((k1+m1)./2))./(2.*2.*omega1.^((k1+m1)./2))).*(x.^(k1+m1
-1))).*Kei1;
y2=(((4.*m2.^((k2+m2)./2))./(1.*6.*omega2.^((k2+m2)./2))).*(x.^(k2+m2
-1))).*Kei2;
y3=(((4.*m3.^((k3+m3)./2))./(1.*2.*omega3.^((k3+m3)./2))).*(x.^(k3+m3
-1))).*Kei3;

plot(x,y);
hold on;
plot(x,y1,'r-');
hold on;
plot(x,y2,'g-');
hold on;
plot(x,y3,'c-');
legend('m=2, k=2, omega=1','m=3, k=3, omega=2','m=2, k=4, omega=1.5',
'm=2, k=3, omega=1.5')
ylabel('Probability Density Function');
grid on;
```

2. MOMENT GENERATING FUNCTION

2.1 Εισαγωγή

Στη θεωρία των πιθανοτήτων και στη στατιστική, η moment generating function μιας τυχαίας μεταβλητής αποτελεί έναν εναλλακτικό ορισμό για την κατανομή της πιθανότητας. Παρέχει τη βάση για μια εναλλακτική διαδρομή σε αναλυτικά αποτελέσματα, χωρίς τη χρήση της συνάρτησης πυκνότητας πιθανότητας και της αθροιστικής συνάρτησης κατανομής.

Η μέση τιμή της πιθανότητας συμβόλου δίνεται από τη σχέση:

$$\overline{P_s} = a_M \int_0^\infty Q\left(\sqrt{2g_M\gamma_s}\right) p(\gamma_s)\, d\gamma_s = a_M \int_0^\infty \frac{1}{\pi}\int_0^{\pi/2} \exp\left[\frac{-2g_M\gamma_s}{2\sin^2\varphi}\right] d\varphi\, p(\gamma_s)\, d\gamma_s \quad (2.1)$$

Η παραπάνω σχέση προκύπτει από τη:

$$P_s(\gamma_s) = a_M Q\left(\sqrt{2g_s M\gamma_s}\right) \quad (2.2)$$

και την εναλλακτική έκφραση της συνάρτησης $Q(\ )$:

$$Q(x) = \frac{1}{\pi}\int_0^{\pi/2} \exp\left[\frac{-x^2}{2\sin^2\varphi}\right] d\varphi \quad (2.3)$$

Αλλάζοντας τη σειρά ολοκλήρωσης, έχοντας το εσωτερικό ολοκλήρωμα ορισμένο παίρνουμε το εξής αποτέλεσμα:

$$\overline{P_s} = \frac{a_M}{\pi}\int_0^{\pi/2} M\left(\frac{g_M}{\sin^2\varphi}; \overline{\gamma_s}\right) d\varphi \quad (2.4)$$

Η moment generating function της $p(\gamma_s)$ είναι:

$$M\left(s;\overline{\gamma_s}\right)=\int_0^{\infty} e^{-s\gamma}p\left(\gamma_s\right)d\gamma_s \qquad (2.5)$$

η οποία είναι σε μορφή μετασχηματισμού Laplace. Η MGF για κάθε κατανομή fading που μας ενδιαφέρει μπορεί να υπολογιστεί με έναν τύπο κάνοντας κλασικούς μετασχηματισμούς Laplace, ή ακόμη να υπολογιστεί αριθμητικά για τις περισσότερες μορφές εξασθένισης.

2.1.1 MGF for Nakagami-m

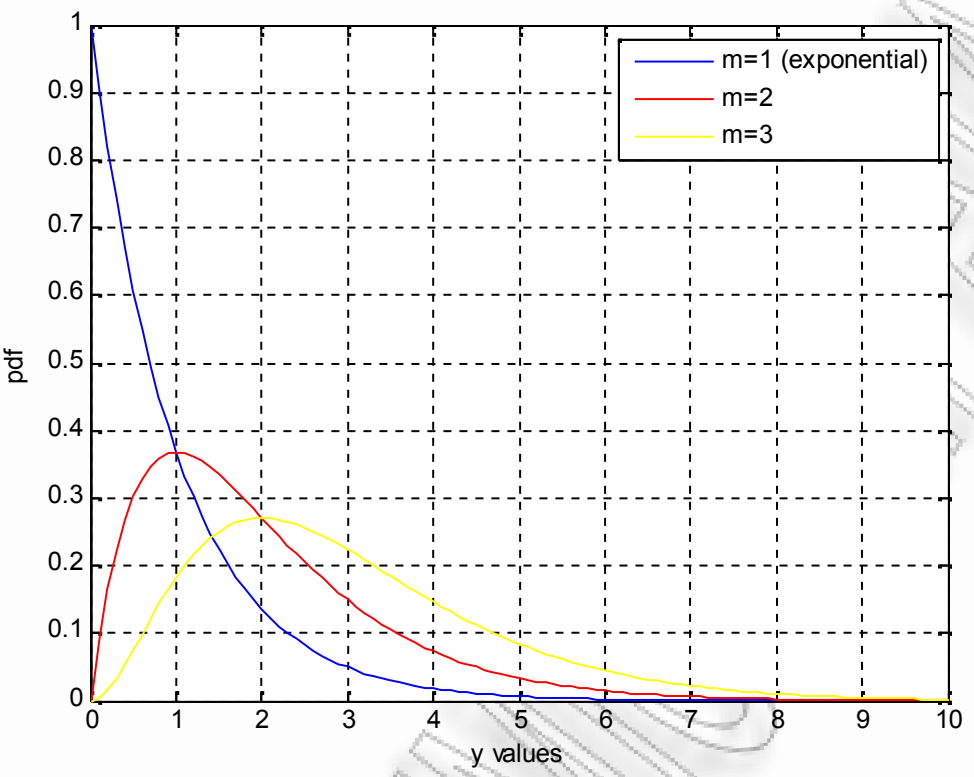
Το πλάτος του καναλιού *R* δίνεται από την κατανομή:

$$p_R=(a;\Omega;m)=\frac{2_m{}^m a^{2m-1}}{\Omega^m\Gamma(m)}\exp\left(-\frac{ma^2}{\Omega}\right) \qquad (2.6)$$

όπου $\Gamma(\ )$ είναι η συνάρτηση *Gamma* και *m* η παράμετρος της *Nakagami-m* διάλειψης, η οποία παίρνει τιμές από $1/2$ έως $\propto$.

Η κατανομή του $SNR = R^2 Es / No$ με αλλαγή μεταβλητής είναι:

$$f_{\gamma_s}(\gamma;\overline{\gamma_s},m)=\frac{m^m}{\Gamma(m)\overline{\gamma_s}^{m-1}}\gamma^{m-1}\exp\left(-\frac{m\gamma}{\overline{\gamma_s}}\right) \qquad (2.7)$$



Σχήμα 2.1 *Probability Density Function of Gamma distribution*

Σημείωση: Όσο η τιμή του m αυξάνεται, τόσο μειώνεται η πιθανότητα να έχουμε τιμές του y κοντά στο 0. Αυτό συνεπάγεται λιγότερες διαλείψεις, επομένως καλύτερο κανάλι διάδοσης.

2.1.2 Matlab script for Gamma PDF

```
clear;
x = 0:0.1:10;
omega = 1;
m1 = 1;
m2 = 2;
m3 = 3;
y = gampdf(x,m1,omega);
y2 = gampdf(x,m2,omega);
y3 = gampdf(x,m3,omega);

figure;
plot(x,y,'b-',x,y2,'r-',x,y3,'y-');
xlabel('y values');
ylabel('pdf');
legend('m=1 (exponential)','m=2','m=3');
```

Παρακάτω φαίνονται οι MGF για τις *Nakagami-m, Rayleigh* και *Rice* κατανομές.

- **Nakagami-m:**

$$MGF\left(\frac{g_M}{\sin^2\varphi};\overline{\gamma_s}\right)=\left(1+\frac{g_M\overline{\gamma_s}}{m\sin^2\varphi}\right)^{-m} \qquad (2.8)$$

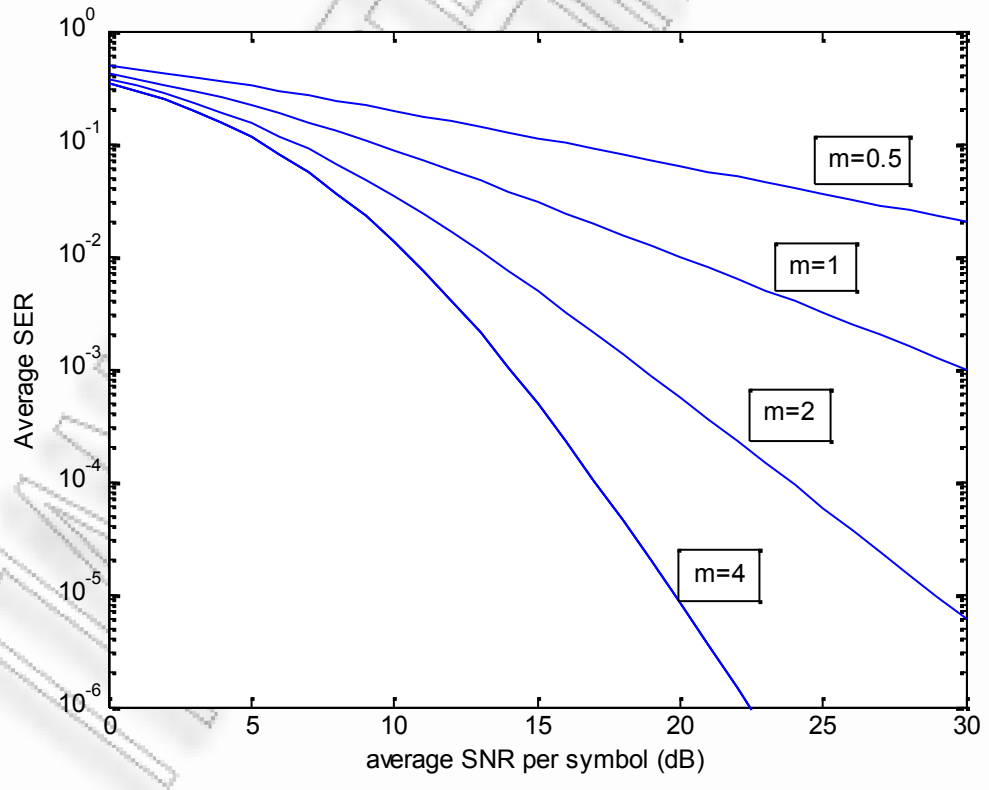
- **Rayleigh:**

$$MGF\left(\frac{g_M}{\sin^2\varphi};\overline{\gamma_s}\right)=\left(1+\frac{g_M\overline{\gamma_s}}{\sin^2\varphi}\right)^{-1} \qquad (2.9)$$

- **Rice:**

$$MGF\left(\frac{g_M}{\sin^2\varphi};\overline{\gamma_s}\right)=\frac{(1+k)\sin^2\varphi}{(1+k)\sin^2\varphi+g_M\overline{\gamma_s}}\exp\left(-\frac{kg_M\overline{\gamma_s}}{(1+k)\sin^2\varphi+g_M\overline{\gamma_s}}\right) \qquad (2.10)$$

Σημείωση: Η κατανομή *Nakagami-m* για *m=1* δίνει την κατανομή *Rayleigh*.



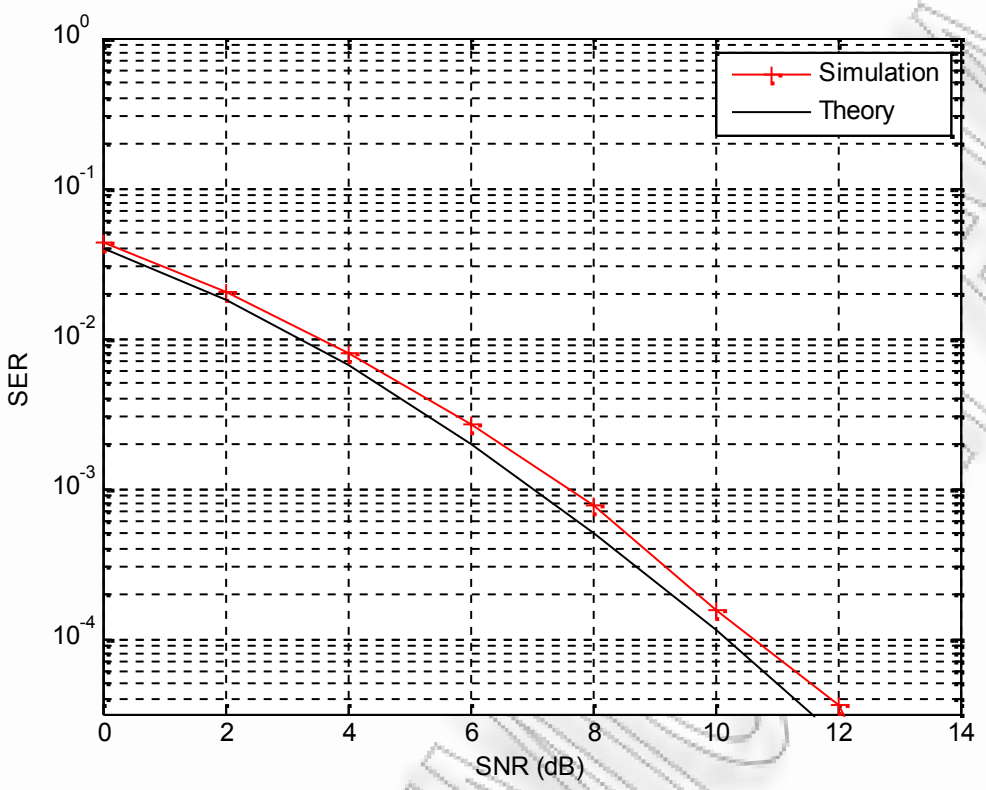
Σχήμα 2.2 *SER of QPSK in Nakagami-m fading channel*

2.1.3 Matlab script for SER with MGF Method

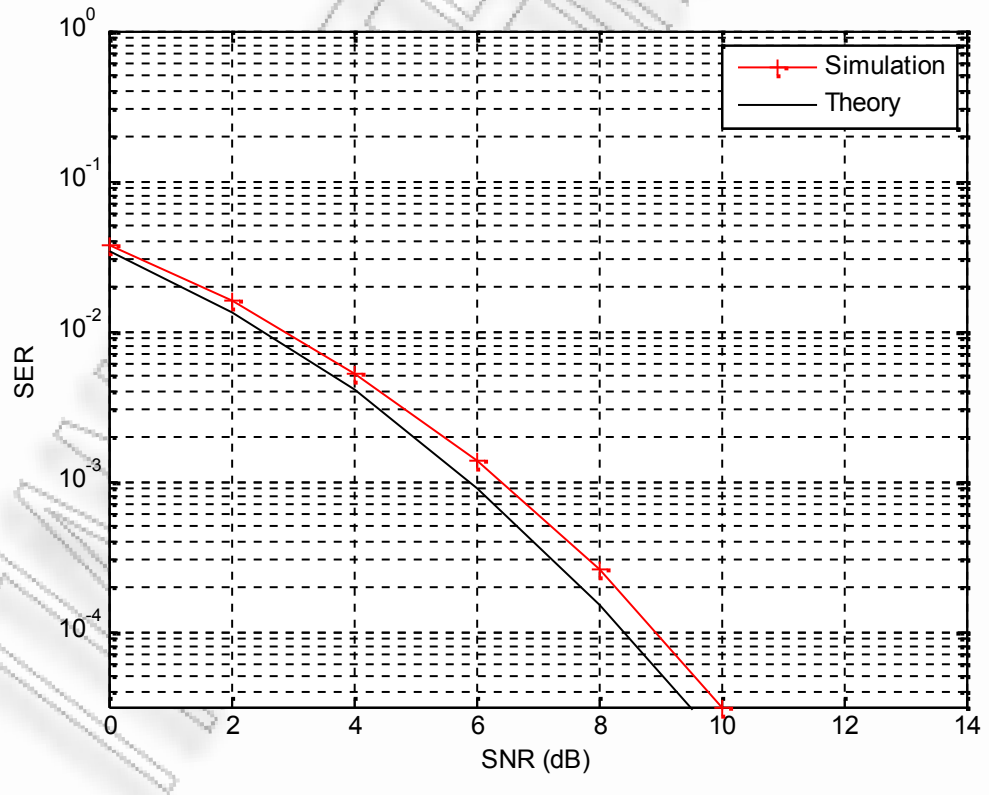
```
function y=mgf(phi);
global gM m x;
tmp = sin(phi).*sin(phi);
y = (1 + (gM.*x./(m.*tmp))).^(-m);


clear;
global gM m x;
m=4;
M=4;        %  QPSK
gM=sin(pi/M)*sin(pi/M);
aM=2;
for i=0:30      % i= SNR in dB
    x=10^(i/10);    % x=SNR in real number inside equations
ser(i+1) = (aM/pi)*quadl(@mgf,0,(pi/2),10^(-6));
snr(i+1)=i;
end
semilogy(snr, ser, '-')
xlabel('average SNR per symbol (dB)')
ylabel('Average SER')
axis([0, 30, 10^(-6), 10^(-0)])
```

2.2 Προσομοίωση

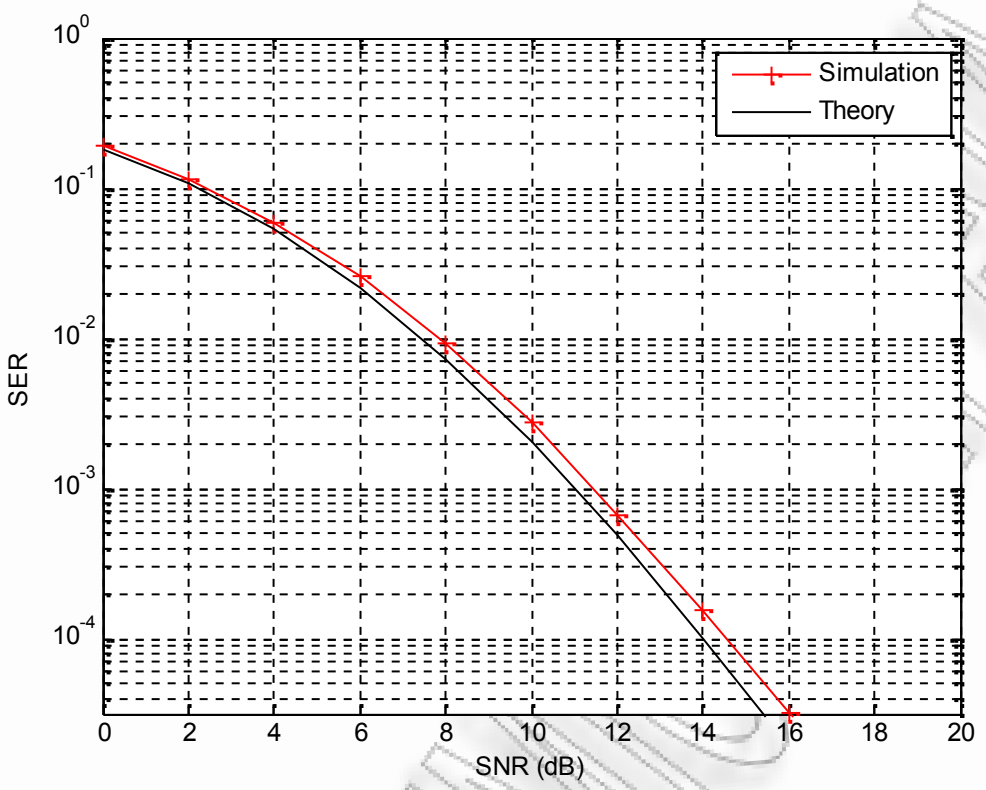
Στα σχήματα που ακολουθούν παρουσιάζονται τα αποτελέσματα της προσομοίωσης σε σύγκριση με αυτά της Moment Generating Function. Για τις προσομοιώσεις χρησιμοποιήθηκαν οι BPSK και QPSK διαμορφώσεις, ενώ τα περιβάλλοντα διάδοσης χαρακτηρίζονται από διαλείψεις Nakagami-m και Rayleigh. Στο τέλος παρατίθεται ο κώδικας της MGF και της προσομοίωσης.



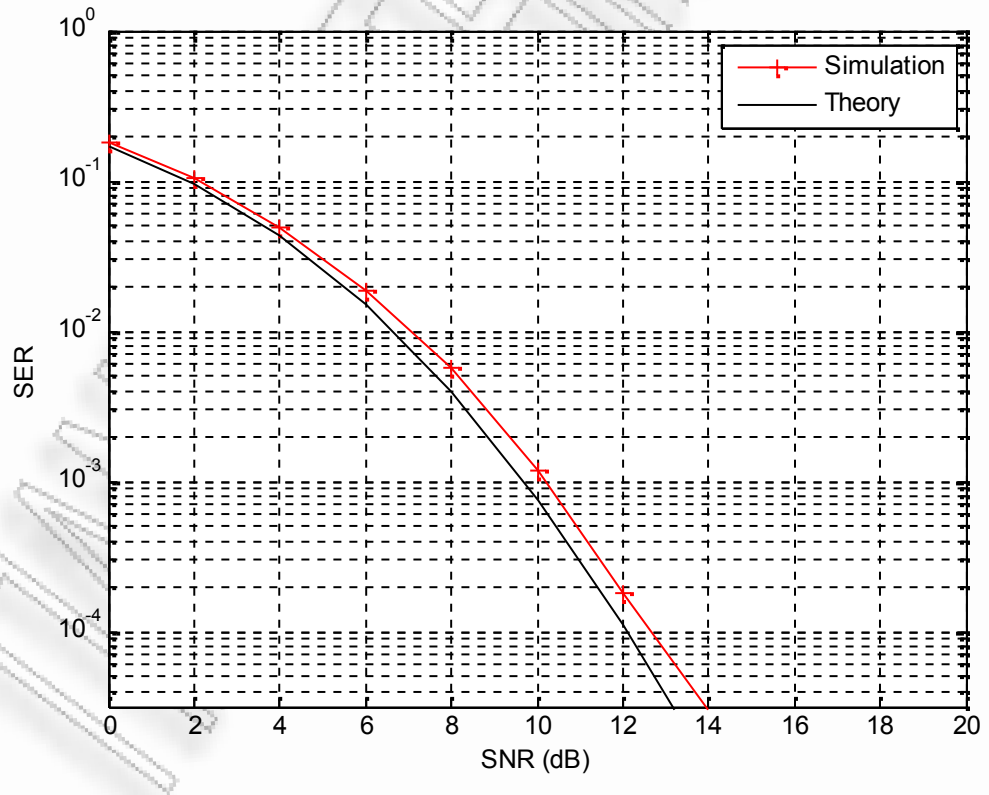
Σχήμα 2.3 *SER of BPSK in Nakagami-m fading channel (m=2)*



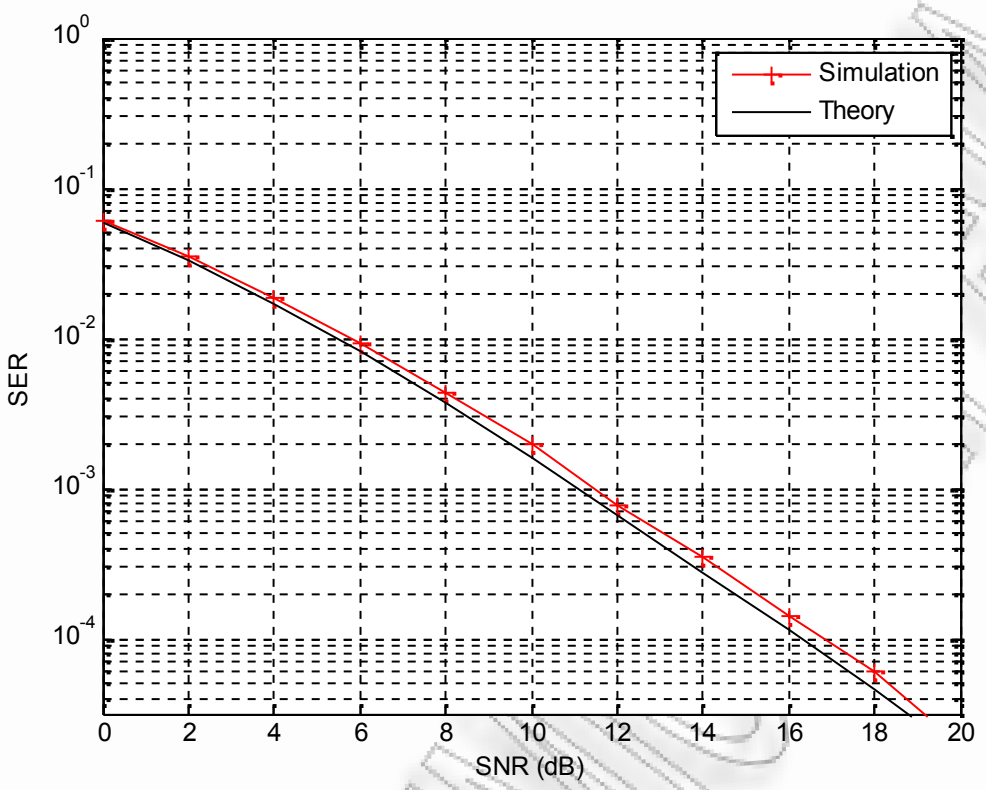
Σχήμα 2.4 *SER of BPSK in Nakagami-m fading channel (m=3)*



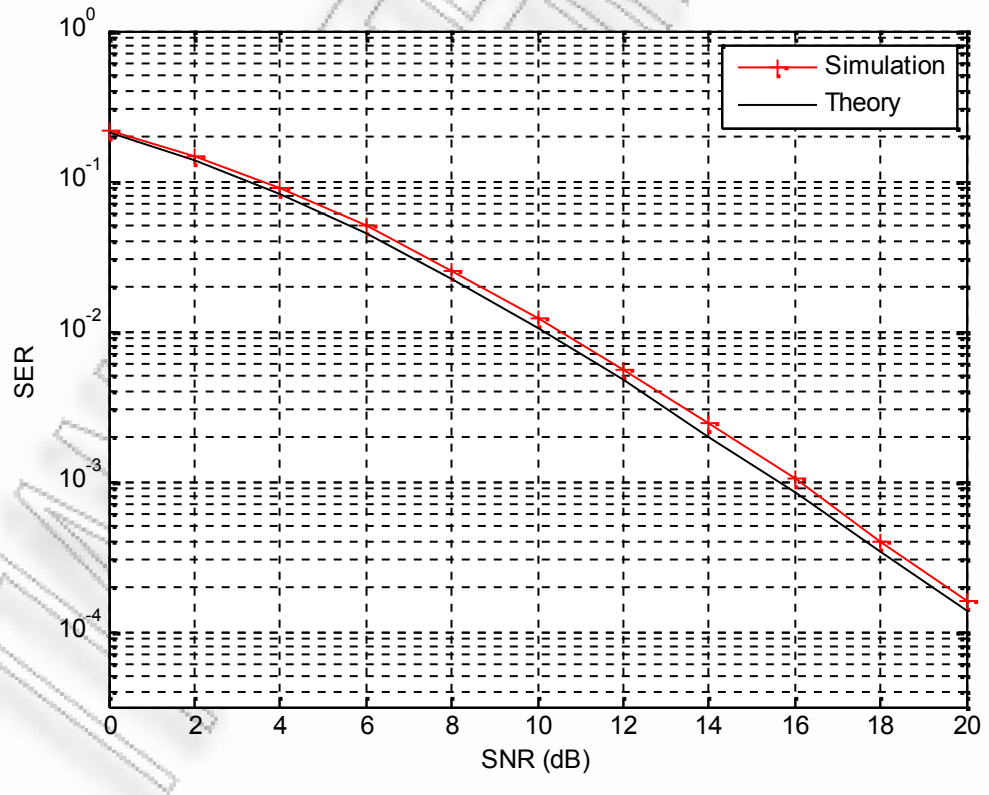
Σχήμα 2.5 *SER of QPSK in Nakagami-m fading channel (m=2)*



Σχήμα 2.6 *SER of QPSK in Nakagami-m fading channel (m=3)*



Σχήμα 2.7 *SER of BPSK in Rayleigh fading channel (m=1)*



Σχήμα 2.8 *SER of QPSK in Rayleigh fading channel (m=1)*

2.2.1 Matlab script for MGF function vs. Simulation

```
% MGF function
function y = mgf(phi)

global g m m2 x L

tmp = sin(phi).*sin(phi);

if L==1
    y = ((1+(g.*x./(m.*tmp))).^(-m));
end

if L==2
    y = ((1+(g.*x./(m.*tmp))).^(-m)).*((1+(g.*x./(m2.*tmp))).^(-m2));
end


% Theoritical vs Simulation Results - MGF Method vs Simulation
% Rayleigh Fading (m=1)
% Any other value for m-> Nakagami-m
clear;
global g m m2 x L

M = 4;  % M=2 for BPSK
g = (sin(pi/M))^2; % g=1 for BPSK
SNR_dB = 0:2:20;
omega_av = 1;
L=2;

m = 1;
k = 10.5;
m2 = 1;
k2 = 10.5;

N = 1000;
loops = 800;

for i = 1:length(SNR_dB)

    errors = 0;
    SNR = 10^(SNR_dB(i)/10);
    x = SNR;

    for z = 1:loops

        s = randint(1,N,M); %Random message
        u = pskmod(s,M);    %Modulate using PSK

        a = gamrnd(m,omega_av/m, [1,N]);    % Direct channel
        b = gamrnd(k,1/k, [1,N]);
        h = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
        n = sqrt((omega_av./(2*SNR))).*(randn(1,N)+j*randn(1,N));

        a2 = gamrnd(m2,omega_av/m2, [1,N]); % Relay channel
        b2 = gamrnd(k2,1/k2, [1,N]);
        h2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
```

```
        n2 = sqrt((omega_av./(2*SNR))).*(randn(1,N)+j*randn(1,N));

        r0 = (h.*u + n);     % Direct
        r2 = (h2.*u + n2);   % Relay

        y0 = conj(h).*r0;
        y2 = conj(h2).*r2;
        y = ((y0 + y2)./(conj(h).*h + conj(h2).*h2))/L;

        %DEMODULATION
        demodmsg = pskdemod(y,M);
        [nes,res] = symerr(s,demodmsg);

        %FIND TOTAL ERRORS FROM PREVIOUS LOOPS
        errors = errors + nes;
    end

ser(i) = errors/(N*loops);  % Simulation
ser_th(i) = (1/pi)*quadl(@mgf, 0, pi-pi/M, 10^(-6));     % MGF
end
```

3. RELAYS

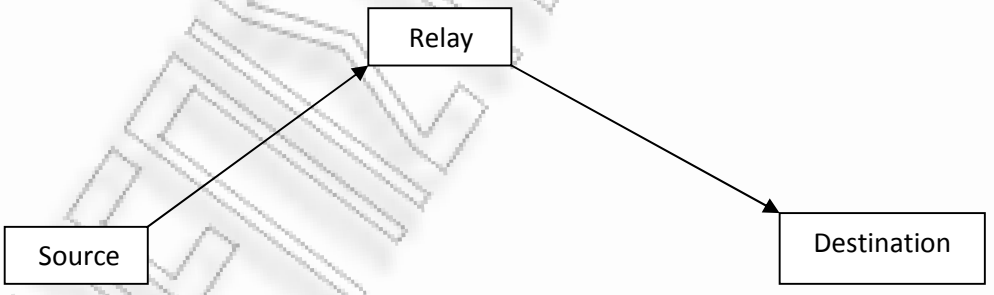
3.1 Εισαγωγή

Οι αναμεταδότες (relays) εξυπηρετούν δίκτυα επικοινωνιών που έχουν σαν βασική αρχή λειτουργίας τη μετάδοση της πληροφορίας από τον πομπό (transmitter or source terminal) στο δέκτη (receiver or destination terminal). Οι αναμεταδότες είναι οι ενδιάμεσοι κόμβοι που καθιστούν δυνατή αυτή την επικοινωνία. Βρίσκουν εφαρμογή στους παρακάτω τομείς:

- Κινητή Τηλεφωνία
- Δορυφορικές Επικοινωνίες
- WLAN
- Ad-Hoc (Δίκτυα επικοινωνίας χωρίς υποδομή)

Αρχικά μελετήθηκε από τον E. C. van der Meulen (1968), ενώ η χωρητικότητά τους απασχόλησε τους Cover και El Gamal το 1979. Η βασική τους αρχή είναι:

Ενδιάμεσοι κόμβοι που καλούνται αναμεταδότες (relays) επεξεργάζονται την πληροφορία από τον πομπό και την επανεκπέμπουν στο δέκτη.



Σχήμα 3.1 *Βασική αρχή λειτουργίας αναμεταδότη*

Οι αναμεταδότες χωρίζονται σε δύο βασικές κατηγορίες:

- Regenerative Relays (Digital Relays)
  Αποκωδικοποιεί (decode) το λαμβανόμενο σήμα, το κωδικοποιεί ξανά (encode) και το εκπέμπει στον επόμενο κόμβο.
- Non-Regenerative Relays (Amplify-and-Forward or Analog Relays)

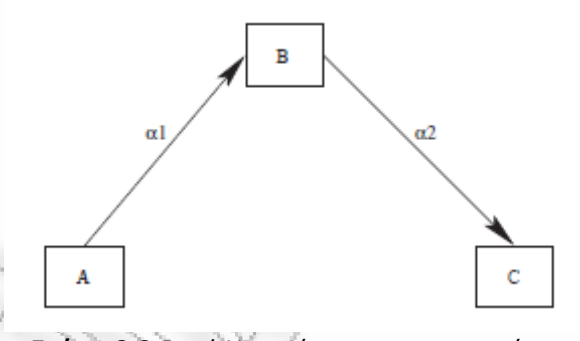
Κάθε κόμβος (relay) ενισχύει το λαμβανόμενο σήμα και το αναμεταδίδει στον επόμενο κόμβο.

Οι Regenerative Relays βρίσκουν εφαρμογή κυρίως στις δορυφορικές επικοινωνίες και στην κινητή τηλεφωνία. Οι κόμβοι που ανήκουν σε αυτή την κατηγορία δεν αναμεταδίδουν θόρυβο. Ο θόρυβος που λαμβάνουν αφαιρείται κατά την αποκωδικοποίηση. Αντίθετα, οι Non-Regenerative relays επανεκπέμπουν μαζί με το σήμα που δέχονται από την πηγή και τον θόρυβο που προστίθεται σε αυτό.

3.2 Non Regenerative Relays

3.2.1 Channel State Information Relays

Έστω το ακόλουθο σύστημα επικοινωνίας με έναν αναμεταδότη



Σχήμα 3.2 *Dual-Hop σύστημα επικοινωνίας*

Το πλάτος των διαλείψεων μπορεί να ακολουθεί οποιαδήποτε κατανομή (Rayleigh, Nakagami-*m*, Rice, etc.)

- **Rayleigh** $p_a(a) = \frac{2a}{\Omega}\exp\left(\frac{-a^2}{\Omega}\right), \qquad \alpha \geq 0 \text{ και } \Omega = \overline{a^2} \qquad (3.1)$
- **Nakagami-*m*** $p_a(a) = \frac{2m^m a^{2m-1}}{\Omega^m \Gamma(m)}\exp\left(\frac{-ma^2}{\Omega}\right), \; a \geq 0 \text{ και } m \geq 0.5 \quad (3.2)$
- **Rice** $p_a(a) = \frac{2(1+n^2)e^{-n^2}a}{\Omega}\exp\left(-\frac{(1+n^2)a^2}{\Omega}\right) I_0\left(2na\sqrt{\frac{1+n^2}{\Omega}}\right), \quad n \geq 0 \quad \text{και}$

  $a \geq 0, \; K = n^2, \; Ricean \;\; Factor \qquad (3.3)$

Το εισερχόμενο σήμα στην είσοδο του αναμεταδότη (B) είναι:

$$r_b(t) = a_1 s(t) + n_1(t) \quad (3.4)$$

Το σήμα στο δέκτη (C) είναι:

$$\begin{aligned} r_c(t) &= a_2 G r_b(t) + n_2(t) \\ &= a_2 G(a_1 s(t) + n_1(t)) + n_2(t) \end{aligned} \quad (3.5)$$

Το ισοδύναμο end-to-end SNR είναι:

$$\gamma_{end} = \frac{a_1^2 a_2^2 G^2}{a_2^2 G^2 N_{01} + N_{02}} = \frac{\frac{a_1^2}{N_{01}} \frac{a_2^2}{N_{02}}}{\frac{a_2^2}{N_{02}} + \frac{1}{G^2 N_{01}}} \quad (3.6)$$

- **Επιλογή του Relay Gain**

Μια πιθανή επιλογή για το κέρδος είναι η αντιστροφή του καναλιού:

$$G^2 = {}^{1}\!/_{a_1^2}, \quad \text{[Hasna, Alouini,COMML'03]} \quad (3.7)$$

Το ισοδύναμο end-to-end SNR για μια τέτοια περίπτωση είναι:

$$\gamma_{end} = \frac{\gamma_1 \gamma_2}{\gamma_1 + \gamma_2} \quad (3.8)$$

Η επιλογή αυτής της τιμής για το κέρδος του αναμεταδότη οδηγεί σε υψηλούς περιορισμούς στην απόδοσή του.

Μια δεύτερη επιλογή προτάθηκε από τους Laneman, Wornell και είναι η ακόλουθη:
Το Gain (G) επιλέγεται ώστε η ισχύς εξόδου του αναμεταδότη να μην ξεπερνά κάποια συγκεκριμένη τιμή, έστω ε_2:

$$G^2(\varepsilon_1 a_1^2 + N_0) \le \varepsilon_2 \Rightarrow \left\{G^2\right\}_{\max} = \frac{\varepsilon_2}{\varepsilon_1 a_1^2 + N_0} \quad \text{[Laneman, Wornell, WCNC'00]} \quad (3.9)$$

ή για κανονικοποιημένες (normalized) τιμές της ισχύος

$$\left\{G^2\right\}_{\max} = \frac{1}{a_1^2 + N_0} \quad \text{[Laneman, Wornell, WCNC'00]} \quad (3.10)$$

Σε αυτή την περίπτωση το κέρδος του αναμεταδότη περιορίζεται όταν παρουσιάζονται βαθιές διαλείψεις στην επικοινωνία του με τον πομπό.
Το ισοδύναμο end-to-end SNR είναι:

$$\gamma_{end} = \frac{\frac{\varepsilon_1 a_1^2}{N_0}\frac{a_2^2}{N_0}}{\frac{a_2^2}{N_0} + \frac{1}{G^2 N_0}} = \frac{\gamma_1 \gamma_2}{\gamma_1 + \gamma_2 + 1} \qquad (3.11)$$

όπου $\gamma_i = \frac{\varepsilon_i a_i^2}{N_0}, (i = 1, 2,)$ το SNR για κάθε hop.

Τα παραπάνω αφορούσαν στην περίπτωση όπου το κέρδος του αναμεταδότη έπαιρνε μια τιμή με βάση την κατάσταση του διαύλου. Το πλάτος του fading ήταν αυτό που ρύθμιζε την τιμή που Gain. Αυτή η κατηγορία αναμεταδοτών ονομάζεται CSI-Relays (Non Regenerative - Channel State Information).

3.2.2 Fixed Gain Relays

Μια άλλη κατηγορία αναμεταδοτών, στην οποία το Gain παίρνει μια σταθερή τιμή είναι οι Fixed Gain Relays. Οι αναμεταδότες που ανήκουν σε αυτή την κατηγορία έχουν σταθερό κέρδος, δεν απαίτουν πολύπλοκα κυκλώματα και είναι αρκετά αποδοτικοί. Μάλιστα για χαμηλές τιμές μέσου SNR παρουσιάζουν καλύτερη απόδοση από τους CSI Relays. Οι Fixed Gain Relays χωρίζονται και αυτοί σε δύο υπό-κατηγορίες: τους “Blind” και “Semi-Blind” Relays.

3.2.3 Blind Relays

Το ολικό στιγμιαίο SNR είναι:

$$\gamma_{end} = \frac{\frac{\varepsilon_1 a_1^2}{N_0}\frac{a_2^2}{N_0}}{\frac{a_2^2}{N_0} + \frac{1}{G^2 N_0}} \qquad (3.12)$$

Αν θεωρήσουμε C, έναν σταθερό αριθμό για σταθερό Gain:

$$C = \varepsilon_2 / (G^2 N_0), \qquad \text{[Hasna \& Alouini, ICAS \& SP'03]} \qquad (3.13)$$

τότε το end-to-end SNR είναι:

$$\gamma_{end} = \frac{\gamma_1 \gamma_2}{C + \gamma_2} \qquad (3.14)$$

3.2.4 Semi-Blind Relays

Το Gain επιλέγεται ως εξής:

$$G^2 = E\left[\frac{\varepsilon_2}{\varepsilon_1 a_1^2 + N_{01}}\right] \quad (3.15)$$

Το στιγμιαίο SNR εξόδου υπολογίζεται πάλι από τη σχέση:

$$\gamma_{end} = \frac{\gamma_1\gamma_2}{C+\gamma_2} \quad (3.16)$$

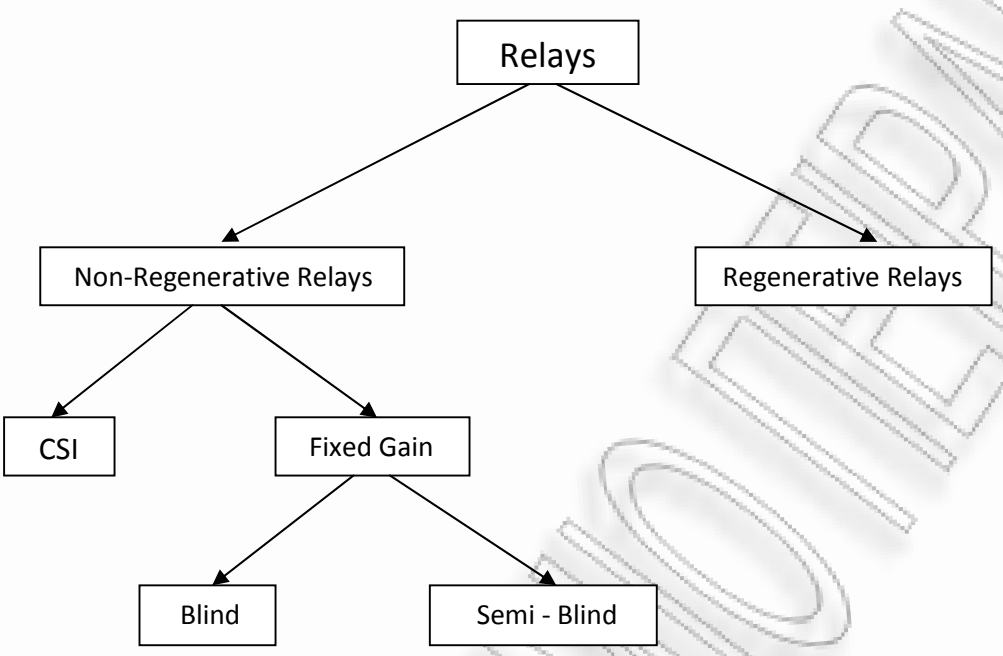
Εάν επιλέξουμε Gain:

$$G^2 = \frac{\varepsilon_2}{\mathrm{E}\left[|r_R|^2\right]}, \quad \text{[Nabar, et. al., 2003]} \quad (3.17)$$

με $\mathrm{E}\left[|r_R|^2\right]$ τη μέση ισχύ του σήματος εισόδου στον αναμεταδότη, τότε ο αναμεταδότης κανονικοποιεί το λαμβανόμενο σήμα στη μονάδα και το ολικό SNR στην έξοδο του δέκτη είναι ίσο με :

$$\gamma_{end} = \frac{\frac{\varepsilon_1 a_1^2}{N_0}\frac{a_2^2}{N_0}}{\frac{a_2^2}{N_0}+\frac{1}{G^2 N_0}} = \frac{\gamma_1\gamma_2}{\gamma_2+\gamma_1+1} \quad (3.18)$$

Στο παρακάτω σχήμα παρουσιάζονται οι διάφορες κατηγορίες στις οποίες χωρίζονται οι αναμεταδότες.



Σχήμα 3.3 *Κατηγοριοποίηση Relays*

3.3 Επίδοση Non Regenerative CSI Relays

Στα γραφήματα που ακολουθούν μελετάται η επίδοση των αναμεταδοτών όσον αφορά στους ρυθμούς σφαλμάτων BER και SER. Όλα τα αποτελέσματα αφορούν Dual-Hop σύστημα μετάδοσης, ενώ τα κανάλια του συστήματος είναι ανεξάρτητα μεταξύ τους και ανόμοια. Θεωρούμε στο μοντέλο μετάδοσης του σχήματος 3.2 την ύπαρξη απευθείας συνιστώσας επικοινωνίας (LoS) μεταξύ του κόμβου A και του κόμβου C.

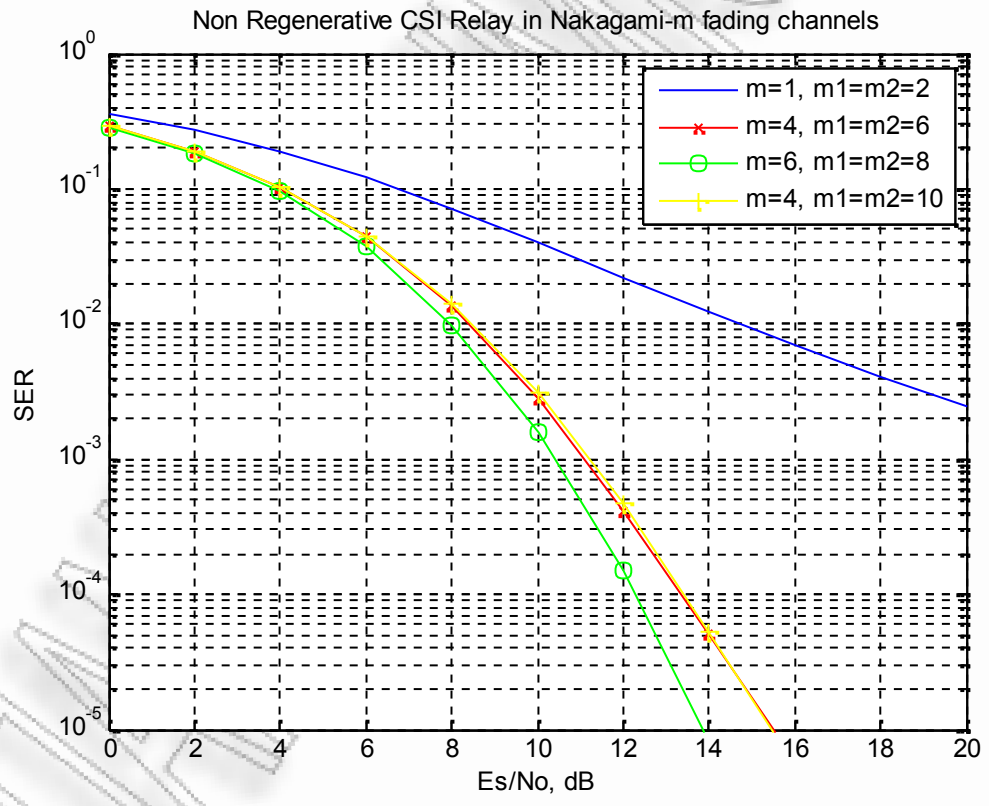
Τα μοντέλα διαλείψεων που χρησιμοποιήθηκαν για τις προσομοιώσεις είναι Nakagami-m και Generalized-K. Όπως είδαμε στο πρώτο κεφάλαιο οι συναρτήσεις πυκνότητας πιθανότητας των κατανομών αυτών είναι οι εξής:

- **Nakagami-m PDF:** $f(x|\mu,\omega)=\frac{2\mu^{\mu}}{\Gamma(\mu)\omega^{\mu}}x^{2\mu-1}e^{\left(\frac{-\mu}{\omega}x^2\right)}$ (3.16)

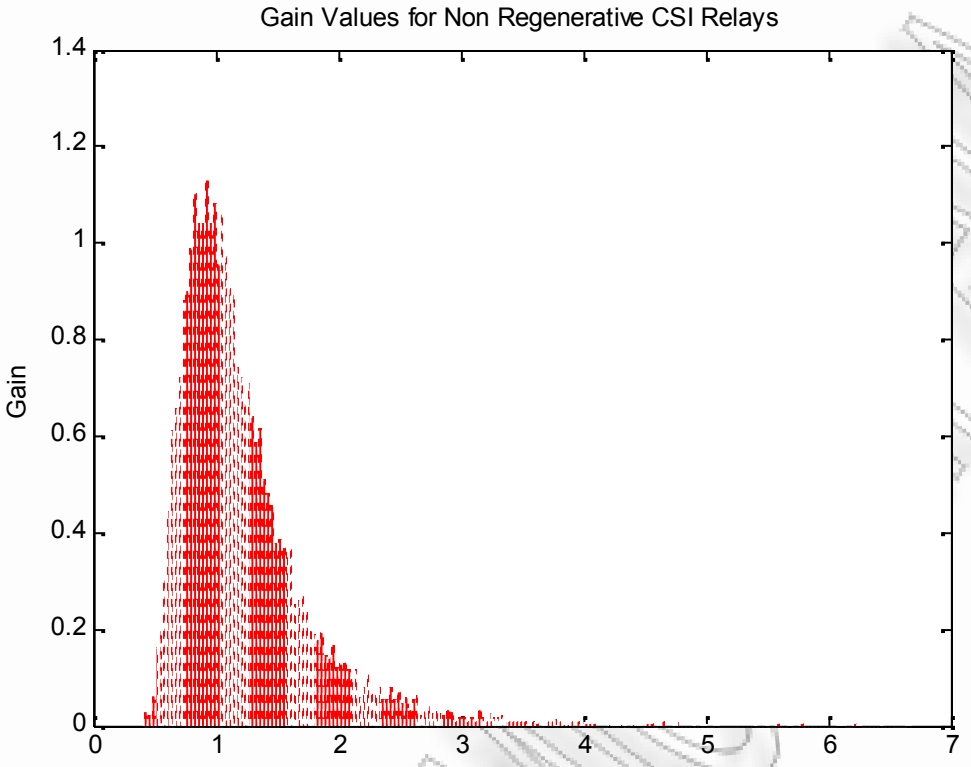
- **Generalized-K PDF:** $f_x(x)=\frac{4m^{(k+m)/2}}{\Gamma(m)\Gamma(k)\Omega^{(k+m)/2}}x^{k+m-1}K_{k-m}\left(2\left(\frac{m}{\Omega}\right)^{1/2}x\right)$ (3.17)

Η τιμή του Relay Gain δίνεται από τη σχέση: $G^2=\frac{1}{a_1^2+No}$, η οποία όπως είδαμε παραπάνω οδηγεί σε ισοδύναμο SNR: $\gamma_{eq1}=\frac{\gamma_1\gamma_2}{\gamma_1+\gamma_2+1}$.

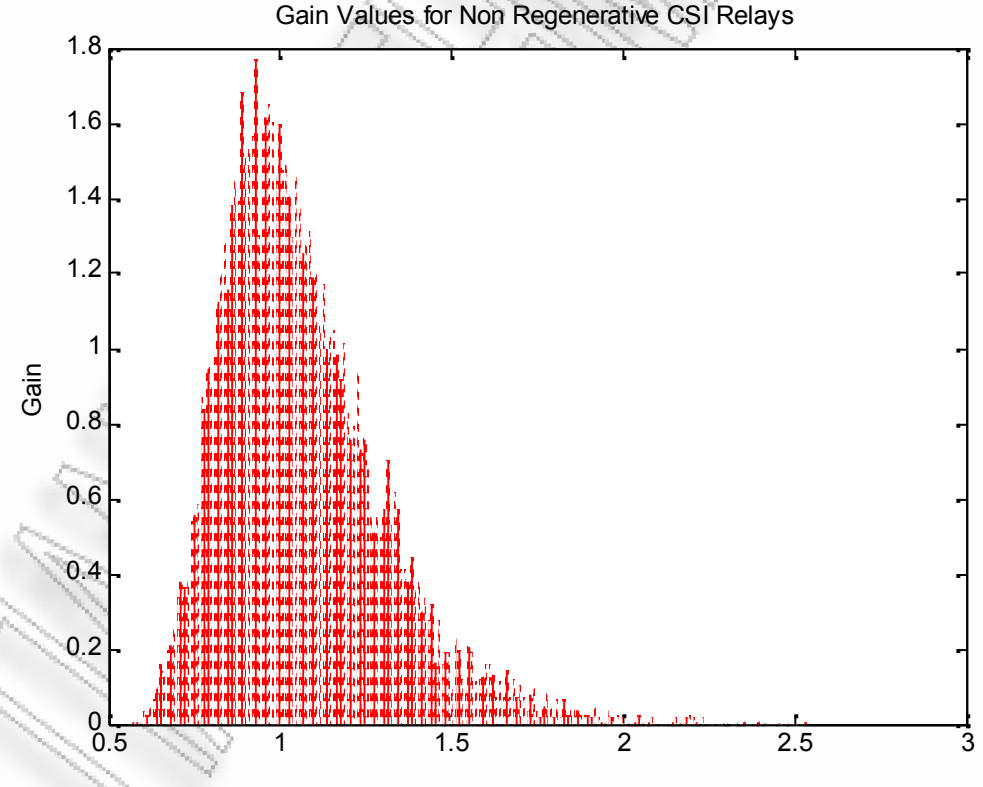
Για το πρώτο σχήμα παρατίθενται και τα αντίστοιχα ιστογράμματα για το κέρδος, ενώ για κάθε σχήμα ακολουθεί ο κώδικας matlab.



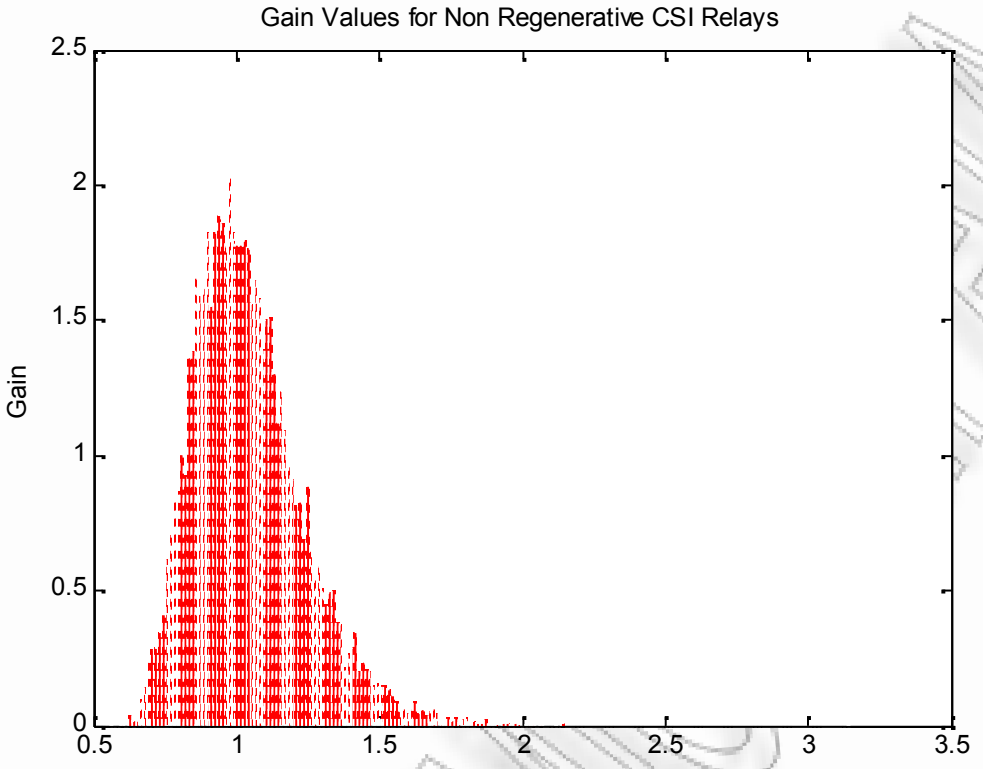
Σχήμα 3.4 *SER of CSI Relay for QPSK modulation in Nakagami-m fading channels*



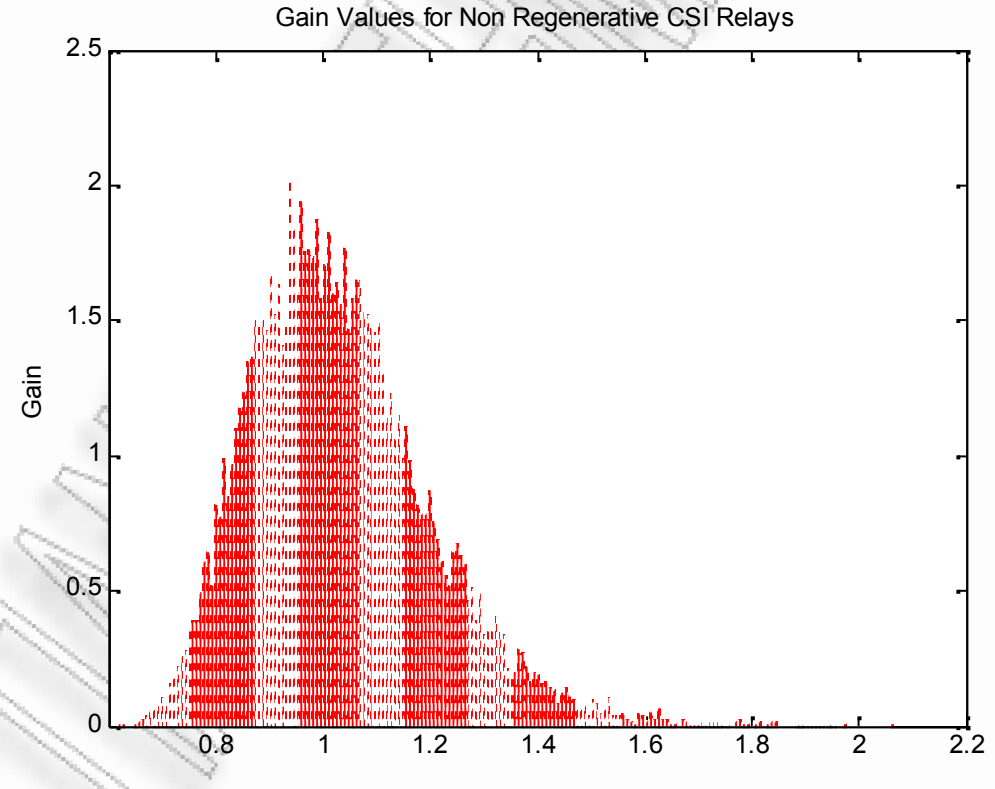
Σχήμα 3.5 *Gain values for m=1, m1=m2=2 in Nakagami-m fading channels*



Σχήμα 3.6 *Gain values for m=4, m1=m2=6 in Nakagami-m fading channels*



Σχήμα 3.7 *Gain values for m=6, m1=m2=8 in Nakagami-m fading channels*

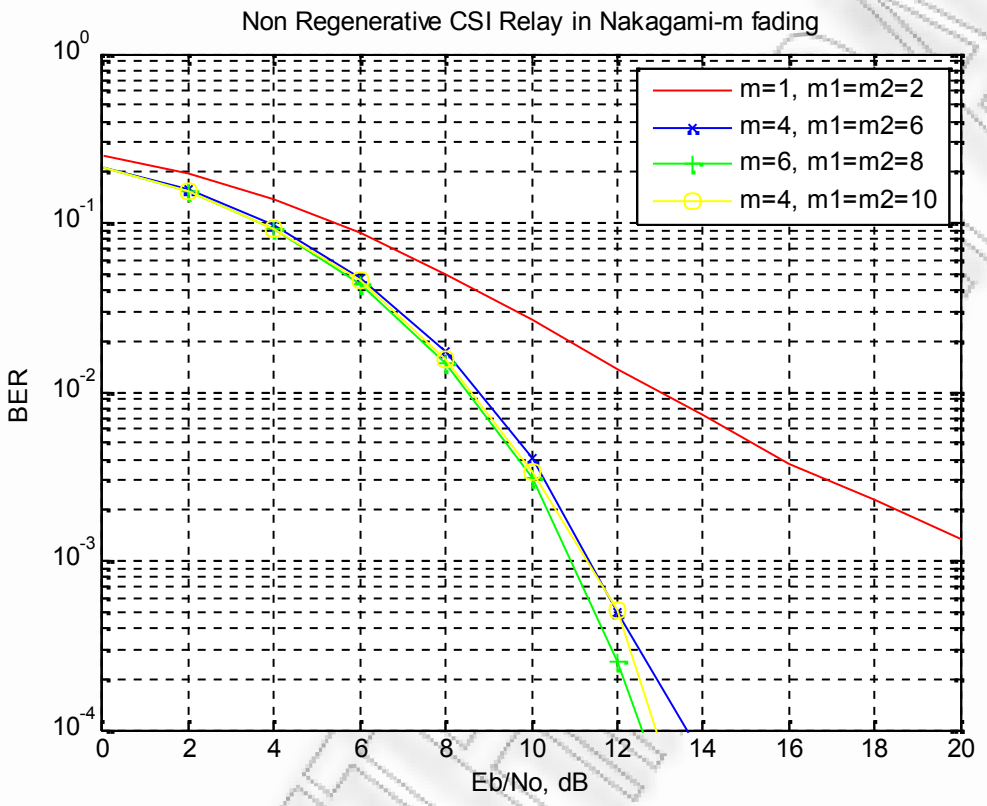


Σχήμα 3.8 *Gain values for m=4, m1=m2=10 in Nakagami-m fading channels*

3.3.1 Matlab script for SER of Non Regenerative CSI Relay in Nakagami-m

```
M = 4;
SNR_dB = 0:2:20;
omega_av = 1;
%1st case
m = 1;
k = 1;
m1 = 2;
k1 = 3;
m2 = 2;
k2 = 3;

N = 10000;
loops = 1000;

for i=1:length(SNR_dB)

    errors_1 = 0;
    SNR = 10^(SNR_dB(i)/10);

    for z=1:loops

        h = randint(1,N,M);
        u = pskmod(h,M);

        %%%%%%%%%%%%%%%%%%1st CASE %%%%%%%%%%%
        a = gamrnd(m,omega_av/m,1,N);
        b = 1;%gamrnd(k,1/k,1,N);
        r = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
        n = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a1 = gamrnd(m1,omega_av/m1,1,N);
        b1 = 1; %gamrnd(k1,1/k1,1,N);
        r1 = sqrt(a1.*b1).*exp(2.*j.*pi.*rand(1,N));
        n1 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a2 = gamrnd(m2,omega_av/m2,1,N);
        b2 = 1; %gamrnd(k2,1/k2,1,N);
        r2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
        n2 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));


        %%%%% Amplify & Forward %%%%%%% 1st CASE %%%%%%%%
        A = sqrt(1./((a1.*b1) + (1./(SNR))));
        y1 = conj(r).*((r.*u + n))./(r.*conj(r));
        y2 = conj(r1).*conj(r2).*(r2.*(A.*(r1.*u + n1) +
n2))./(r1.*r2.*conj(r1).*conj(r2));
        y = y1 + y2;

        %%%%%% Demodulation CASE 1 %%%%%%%
        demodmsg = pskdemod(y,M);
        [nes_amp,res_amp] = symerr(h,demodmsg);

        %Find total erros
        errors_1 = errors_1 + nes_amp;
    end
```

```
    ser_amp(i) = errors_1/(N*loops);
end
```



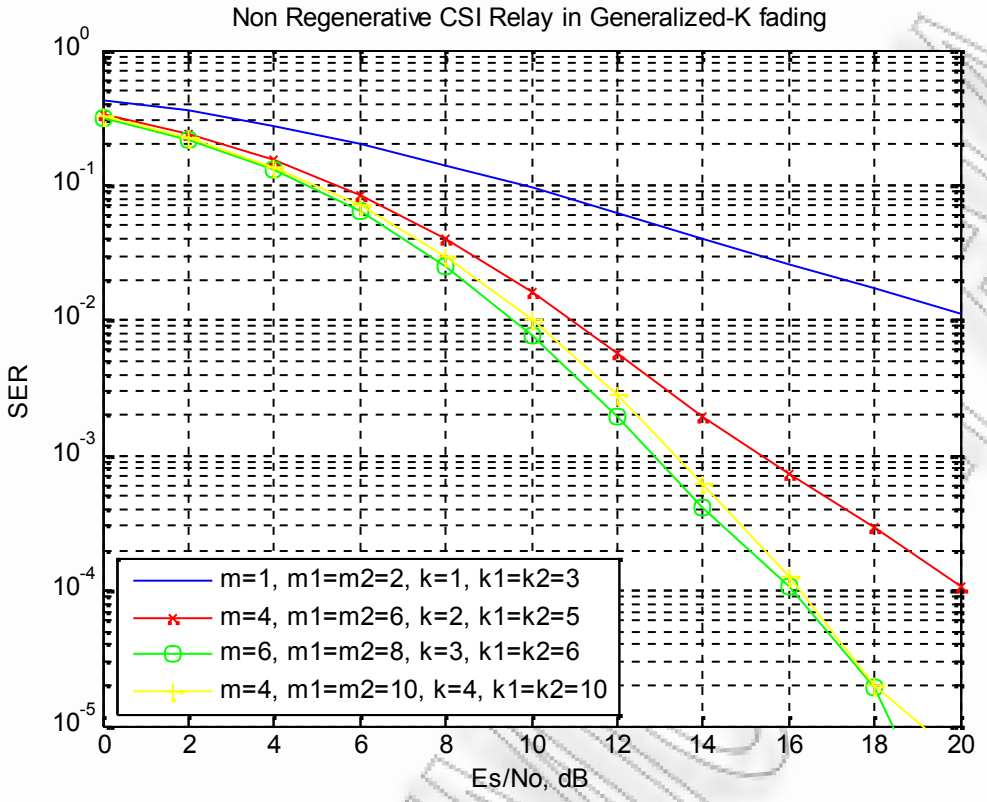
Σχήμα 3.9 *BER of CSI Relay for QPSK modulation in Nakagami-m fading channels*

3.3.2 Matlab script for BER of Non Regenerative CSI Relay in Nakagami-m

```
N = 10^5; % number of symbols
M = 4;    % constellation size
kb = log2(M); % bits per symbol
omega_av = 1;

%1st case
m = 1;
k = 1;
m1 = 2;
k1 = 3;
m2 = 2;
k2 = 3;

thetaMpsk = [0:M-1]*2*pi/M; % reference phase values

Eb_N0_dB  = [0:2:20]; % multiple Es/N0 values
Es_N0_dB  = Eb_N0_dB + 10*log10(kb);
```

```
% Mapping for binary <--> Gray code conversion
ref = [0:M-1];
map = bitxor(ref,floor(ref/2));
[tt ind] = sort(map);

ipPhaseHat = zeros(1,N);
for ii = 1:length(Eb_N0_dB)


    % symbol generation
    % ------------------
    ipBit = rand(1,N*kb,1)>0.5; % random 1's and 0's
    bin2DecMatrix = ones(N,1)*(2.^[(kb-1):-1:0]) ; % conversion from
binary to decimal
    ipBitReshape =  reshape(ipBit,kb,N).'; % grouping to N symbols
having kb bits each
    ipGray = [sum(ipBitReshape.*bin2DecMatrix,2)].'; % decimal to
binary

    % Gray coded constellation mapping
    ipDec = ind(ipGray+1)-1; % bit group to constellation point
    ipPhase = ipDec*2*pi/M; % conversion to phase
    ip = exp(j*ipPhase);  % modulation
    s = ip;

    u = s + 10^(-Es_N0_dB(ii)/20);
    SNR = 10^(Eb_N0_dB(ii)/10);

    %%%%%%%%%%%%%%%%%%1st CASE %%%%%%%%%%%%
        a = gamrnd(m,omega_av/m,1,N);
        b = 1;%gamrnd(k,1/k,1,N);
        r = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
        n = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a1 = gamrnd(m1,omega_av/m1,1,N);
        b1 = 1; %gamrnd(k1,1/k1,1,N);
        r1 = sqrt(a1.*b1).*exp(2.*j.*pi.*rand(1,N));
        n1 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a2 = gamrnd(m2,omega_av/m2,1,N);
        b2 = 1; %gamrnd(k2,1/k2,1,N);
        r2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
        n2 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));



        %%%%% Amplify & Forward %%%%%%% 1st CASE %%%%%%%%
        A = sqrt(1./((a1.*b1) + (1./(SNR))));
        y1 = conj(r).*((r.*u + n))./(r.*conj(r));
        y2 = conj(r1).*conj(r2).*(r2.*(A.*(r1.*u + n1) +
n2))./(r1.*r2.*conj(r1).*conj(r2));
        y = y1 + y2;


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION 1st CASE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    % ------------------------------------------------------------
--------------- %
    % finding the phase from [-pi to +pi]
    opPhase = angle(y);
    % unwrapping the phase i.e. phase less than 0 are
    % added 2pi
    opPhase(find(opPhase<0)) = opPhase(find(opPhase<0)) + 2*pi;

    % rounding the received phase to the closest constellation
    ipPhaseHat = 2*pi/M*round(opPhase/(2*pi/M)) ;
    % as there is phase ambiguity for phase = 0 and 2*pi,
    % changing all phases reported as 2*pi to 0.
    % this is to enable comparison with the transmitted phase
    ipPhaseHat(find(ipPhaseHat==2*pi)) = 0;
    ipDecHat = round(ipPhaseHat*M/(2*pi));

    % Decimal to Gray code conversion
    ipGrayHat = map(ipDecHat+1); % converting to decimal
    ipBinHat = dec2bin(ipGrayHat,kb) ; % decimal to binary

    % converting binary string to number
    ipBinHat = ipBinHat.';
    ipBinHat = ipBinHat(1:end).';
    ipBinHat = str2num(ipBinHat).' ;

    % counting errors
    nBitErr(ii) = size(find([ipBit- ipBinHat]),2); % counting the
number of errors
end
simBer = nBitErr/(N*kb);
```

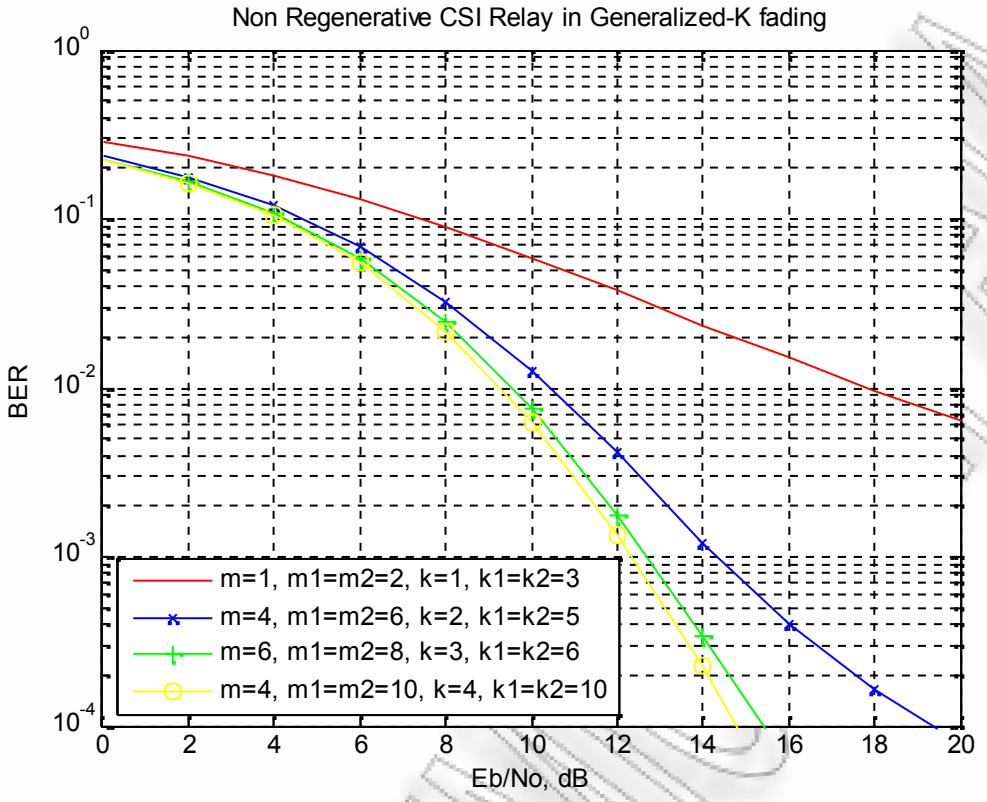


Σχήμα 3.10 *SER of CSI Relay for QPSK modulation in Generalized-K fading channels*

3.3.3 Matlab script for SER of Non Regenerative CSI Relay in Generalized-K

```
M = 4;
SNR_dB = 0:2:20;
omega_av = 1;
%1st case
m = 1;
k = 1;
m1 = 2;
k1 = 3;
m2 = 2;
k2 = 3;

N = 1000;
loops = 1000;
 for i=1:length(SNR_dB)

    errors_1 = 0;
    SNR = 10^(SNR_dB(i)/10);

    for z=1:loops

        h = randint(1,N,M);
        u = pskmod(h,M);

        %%%%%%%%%%%%%%%%%%1st CASE %%%%%%%%%%%%
```

```
        a = gamrnd(m,omega_av/m,1,N);
        b = gamrnd(k,1/k,1,N);
        r = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
        n = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a1 = gamrnd(m1,omega_av/m1,1,N);
        b1 = gamrnd(k1,1/k1,1,N);
        r1 = sqrt(a1.*b1).*exp(2.*j.*pi.*rand(1,N));
        n1 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a2 = gamrnd(m2,omega_av/m2,1,N);
        b2 = gamrnd(k2,1/k2,1,N);
        r2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
        n2 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        %%%%% Amplify & Forward %%%%%%% 1st CASE %%%%%%%%
        A = sqrt(1./((a1.*b1) + (1./(SNR))));
        y1 = conj(r).*((r.*u + n))./(r.*conj(r));
        y2 = conj(r1).*conj(r2).*(r2.*(A.*(r1.*u + n1) +
n2))./(r1.*r2.*conj(r1).*conj(r2));
        y = y1 + y2;

        %%%%%% Demodulation CASE 1 %%%%%%%
        demodmsg = pskdemod(y,M);
        [nes_amp,res_amp] = symerr(h,demodmsg);
        %Find total erros
        errors_1 = errors_1 + nes_amp;
    end

    ser_amp(i) = errors_1/(N*loops);
end
```



Σχήμα 3.11 *BER of CSI Relay for QPSK modulation in Generalized-K fading channels*

3.3.4 Matlab script for BER of Non Regenerative CSI Relay in Generalized-K

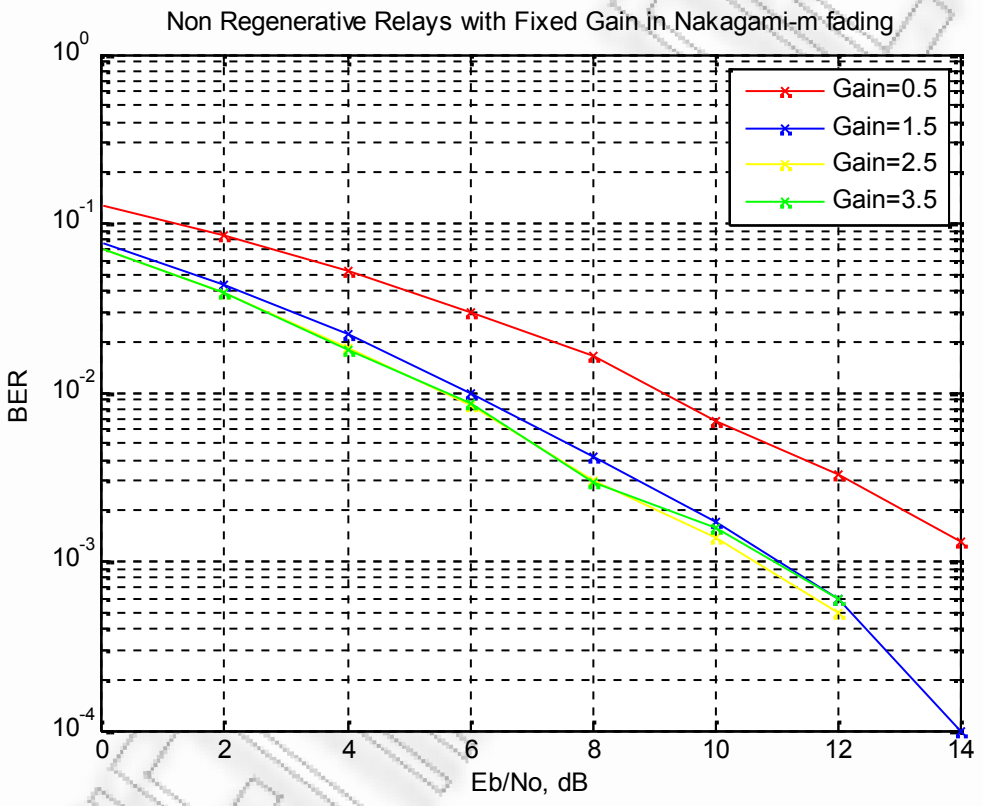
```
N = 10^5; % number of symbols
M = 4;    % constellation size
kb = log2(M); % bits per symbol
omega_av = 1;

m = 1;
k = 1;
m1 = 2;
k1 = 3;
m2 = 2;
k2 = 3;

thetaMpsk = [0:M-1]*2*pi/M; % reference phase values
Eb_N0_dB  = [0:2:20]; % multiple Es/N0 values
Es_N0_dB  = Eb_N0_dB + 10*log10(kb);

% Mapping for binary <--> Gray code conversion
ref = [0:M-1];
map = bitxor(ref,floor(ref/2));
[tt ind] = sort(map);

ipPhaseHat = zeros(1,N);
for ii = 1:length(Eb_N0_dB)
```

```
    % symbol generation
    % ------------------
    ipBit = rand(1,N*kb,1)>0.5; % random 1's and 0's
    bin2DecMatrix = ones(N,1)*(2.^[(kb-1):-1:0]) ; % conversion from
binary to decimal
    ipBitReshape =  reshape(ipBit,kb,N).'; % grouping to N symbols
having kb bits each
    ipGray = [sum(ipBitReshape.*bin2DecMatrix,2)].'; % decimal to
binary

    % Gray coded constellation mapping
    ipDec = ind(ipGray+1)-1; % bit group to constellation point
    ipPhase = ipDec*2*pi/M; % conversion to phase
    ip = exp(j*ipPhase);  % modulation
    s = ip;

    u = s + 10^(-Es_N0_dB(ii)/20);
    SNR = 10^(Eb_N0_dB(ii)/10);

    %%%%%%%%%%%%%%%%%1st CASE %%%%%%%%%%%
        a = gamrnd(m,omega_av/m,1,N);
        b = gamrnd(k,1/k,1,N);
        r = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
        n = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a1 = gamrnd(m1,omega_av/m1,1,N);
        b1 = gamrnd(k1,1/k1,1,N);
        r1 = sqrt(a1.*b1).*exp(2.*j.*pi.*rand(1,N));
        n1 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a2 = gamrnd(m2,omega_av/m2,1,N);
        b2 = gamrnd(k2,1/k2,1,N);
        r2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
        n2 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        %%%%% Amplify & Forward %%%%%%% 1st CASE %%%%%%%%
        A = sqrt(1./((a1.*b1) + (1./(SNR))));
        y1 = conj(r).*((r.*u + n))./(r.*conj(r));
        y2 = conj(r1).*conj(r2).*(r2.*(A.*(r1.*u + n1) +
n2))./(r1.*r2.*conj(r1).*conj(r2));
        y = y1 + y2;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION 1st CASE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % ---------------------------------------------------------------
--------------- %
    % finding the phase from [-pi to +pi]
    opPhase = angle(y);
    % unwrapping the phase i.e. phase less than 0 are
    % added 2pi
    opPhase(find(opPhase<0)) = opPhase(find(opPhase<0)) + 2*pi;

    % rounding the received phase to the closest constellation
    ipPhaseHat = 2*pi/M*round(opPhase/(2*pi/M)) ;
    % as there is phase ambiguity for phase = 0 and 2*pi,
    % changing all phases reported as 2*pi to 0.
    % this is to enable comparison with the transmitted phase
    ipPhaseHat(find(ipPhaseHat==2*pi)) = 0;
```

```
    ipDecHat = round(ipPhaseHat*M/(2*pi));

    % Decimal to Gray code conversion
    ipGrayHat = map(ipDecHat+1); % converting to decimal
    ipBinHat = dec2bin(ipGrayHat,kb) ; % decimal to binary

    % converting binary string to number
    ipBinHat = ipBinHat.';
    ipBinHat = ipBinHat(1:end).';
    ipBinHat = str2num(ipBinHat).' ;

    % counting errors
    nBitErr(ii) = size(find([ipBit- ipBinHat]),2); % couting the
number of errors

    % counting errors
    nBitErr4(ii) = size(find([ipBit- ipBinHat4]),2); % couting the
number of errors

end
simBer = nBitErr/(N*kb);
```

3.4 Επίδοση Non Regenerative with Fixed Gain Relays

Στη συνέχεια παρουσιάζονται οι καμπύλες SER και BER για Non Regenerative Relays με διάφορες τιμές σταθερού κέρδους. Τα κανάλια διαλείψεων είναι Nakagami-m και Generalized-K ενώ οι ψηφιακές διαμορφώσεις που χρησιμοποιήθηκαν είναι οι BPSK και QPSK. Παρουσιάζεται η επίδοση του αναμεταδότη για τιμές κέρδους από Gain=0.5 έως Gain=3.5.

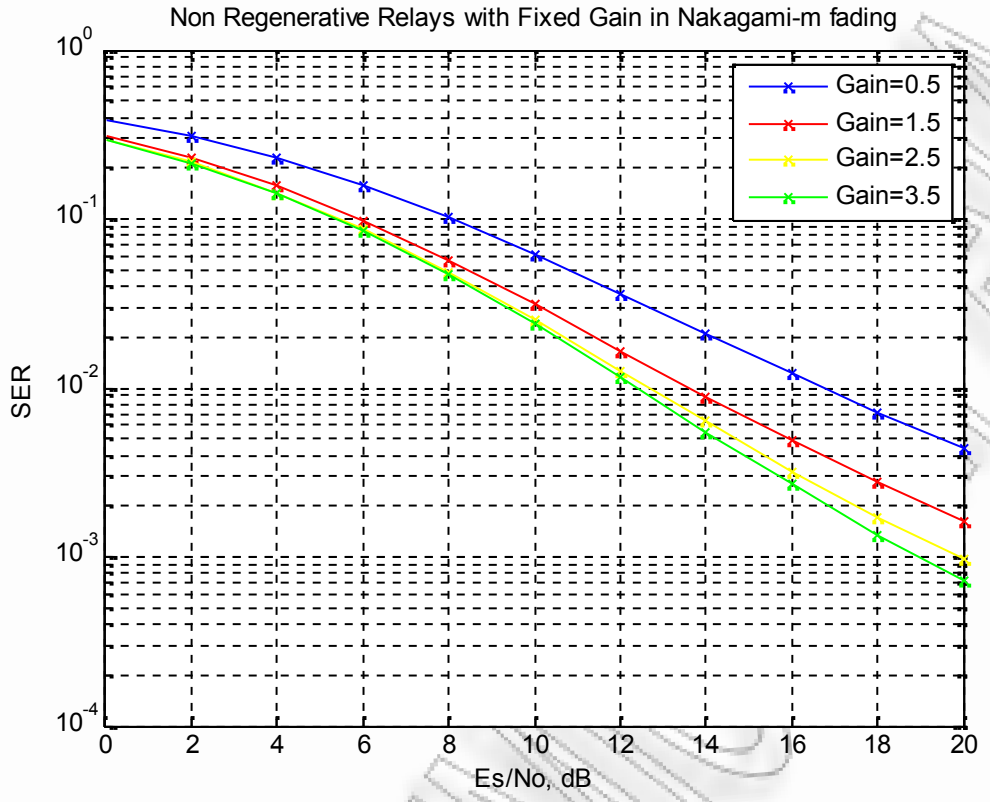


Σχήμα 3.12 *BER of Fixed Gain Relay for BPSK modulation in Nakagami-m fading channels*

3.4.1 Matlab script for BER of Fixed Gain Relay for BPSK in Nakagami-m

```
SNR_dB = 0:2:20;
omega_av = 1;
m=1;
k=1;

m1=2;
k1=3;
m2=2;
```

```
k2=3;

N=10^4;

for i=1:length(SNR_dB)

    errors1=0;
    SNR = 10^(SNR_dB(i)/10);

        for j=1:N
            h = randint;
            u = 2*h-1;

            a = gamrnd(m,omega_av/m);
            b = 1;%gamrnd(k,1/k);
            r = sqrt(a*b);
            n = sqrt((omega_av/(2*SNR)))*randn;

            a1 = gamrnd(m1,omega_av/m1);
            b1 = 1;%gamrnd(k1,1/k1);
            r1 = sqrt(a1*b1);
            n1 = sqrt((omega_av/(2*SNR)))*randn;

            a2 = gamrnd(m2,omega_av/m2);
            b2 = 1;%gamrnd(k2,1/k2);
            r2 = sqrt(a2*b2);
            n2 = sqrt((omega_av/(2*SNR)))*randn;

            A = 0.5; %sqrt(1/((a1*b1)+(1/(SNR))));
            y = r*(r*u + n) + (r2*r1)*(r2*(A*(r1*u + n1)) + n2);
            if y >= 0.0
                h_r = 1;
            elseif y < 0
                h_r = 0;
            end

            errors1 = errors1 + (h_r~=h);
      end

ber(i) = errors1/N;
end
```

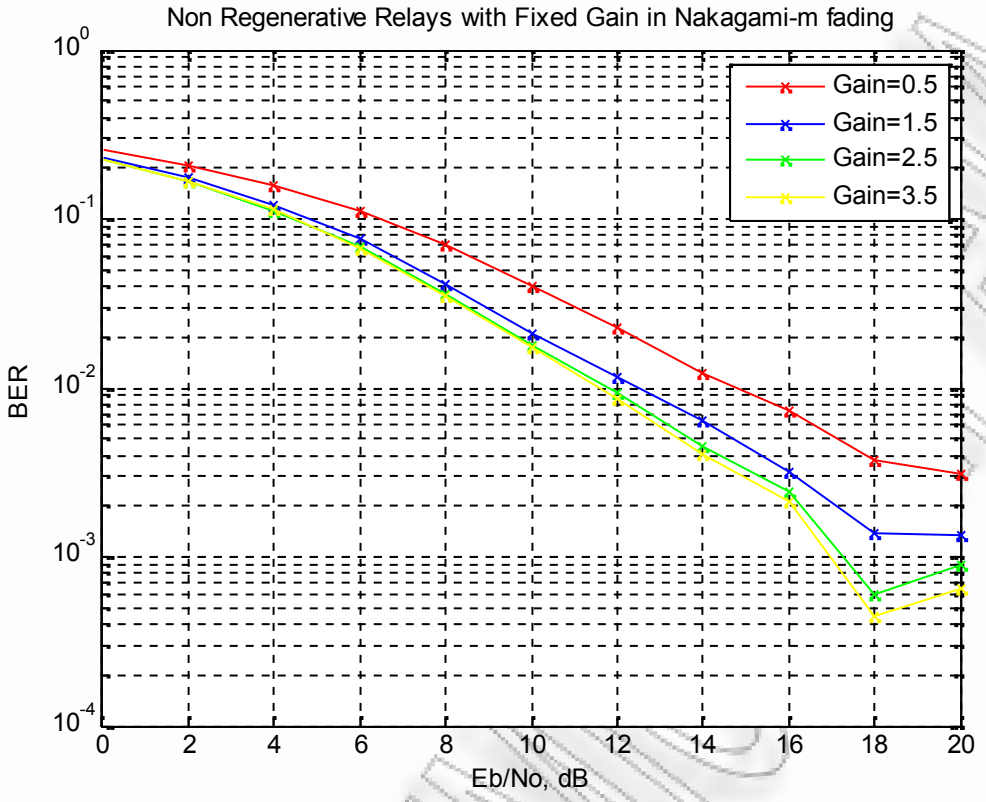


Σχήμα 3.13 *SER of Fixed Gain Relay for QPSK modulation in Nakagami-m fading channels*

3.4.2 Matlab script for SER of Fixed Gain Relay for QPSK in Nakagami-m

```
M = 4;
SNR_dB = 0:2:20;
omega_av = 1;
m = 1;
k = 1;

m1 = 2;
k1 = 3;
m2 = 2;
k2 = 3;

N = 1000;
loops = 1000;

for i=1:length(SNR_dB)

    errors_1 = 0;
    SNR = 10^(SNR_dB(i)/10);

    for z=1:loops

        h = randint(1,N,M);
        u = pskmod(h,M);
```

```
        a = gamrnd(m,omega_av/m,1,N);
        b = 1;%gamrnd(k,1/k,1,N);
        r = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
        n = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a1 = gamrnd(m1,omega_av/m1,1,N);
        b1 = 1;%gamrnd(k1,1/k1,1,N);
        r1 = sqrt(a1.*b1).*exp(2.*j.*pi.*rand(1,N));
        n1 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a2 = gamrnd(m2,omega_av/m2,1,N);
        b2 = 1;%gamrnd(k2,1/k2,1,N);
        r2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
        n2 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        %Amplify & Forward
        A = 0.5;%sqrt(1./((a1.*b1) + (1./(SNR))));
        y1 = conj(r).*((r.*u + n))./(r.*conj(r));
        y2 = conj(r1).*conj(r2).*(r2.*(A.*(r1.*u + n1) +
n2))./(r1.*r2.*conj(r1).*conj(r2));
        y = y1 + y2;

        %Demodulation
        demodmsg = pskdemod(y,M);
        [nes_amp,res_amp] = symerr(h,demodmsg);

        %Find total erros
        errors_1 = errors_1 + nes_amp;
    end

    ser_amp(i) = errors_1/(N*loops);
end
```



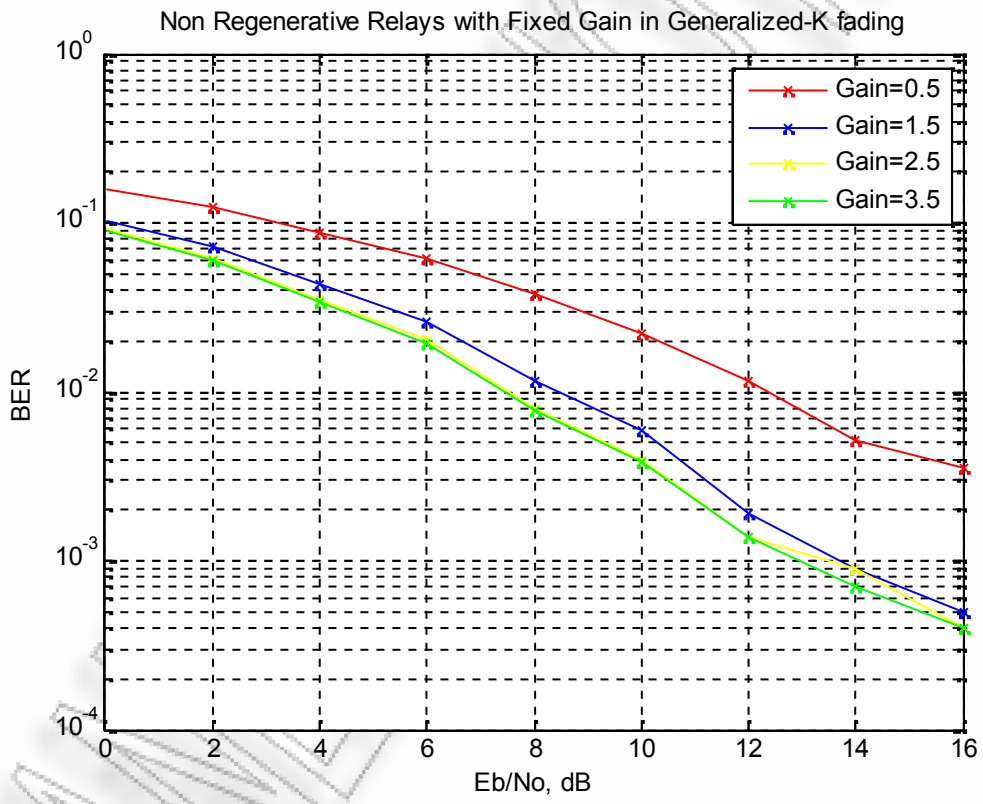
Σχήμα 3.14 *BER of Fixed Gain Relay for QPSK modulation in Nakagami-m fading channels*

3.4.3 Matlab script for BER of Fixed Gain Relay for QPSK in Nakagami-m

```
N = 10^4; % number of symbols
M = 4;    % constellation size
kb = log2(M); % bits per symbol

omega_av = 1;
m = 1;
k = 1;
m1 = 2;
k1 = 3;
m2 = 2;
k2 = 3;

thetaMpsk = [0:M-1]*2*pi/M; % reference phase values

Eb_N0_dB  = [0:2:20]; % multiple Es/N0 values
Es_N0_dB  = Eb_N0_dB + 10*log10(kb);

% Mapping for binary <--> Gray code conversion
ref = [0:M-1];
map = bitxor(ref,floor(ref/2));
[tt ind] = sort(map);

ipPhaseHat = zeros(1,N);
for ii = 1:length(Eb_N0_dB)
```

```
    % symbol generation
    % ------------------
    ipBit = rand(1,N*kb,1)>0.5; % random 1's and 0's
    bin2DecMatrix = ones(N,1)*(2.^[(kb-1):-1:0]) ; % conversion from
binary to decimal
    ipBitReshape =  reshape(ipBit,kb,N).'; % grouping to N symbols
having kb bits each
    ipGray = [sum(ipBitReshape.*bin2DecMatrix,2)].'; % decimal to
binary

    % Gray coded constellation mapping
    ipDec = ind(ipGray+1)-1; % bit group to constellation point
    ipPhase = ipDec*2*pi/M; % conversion to phase
    ip = exp(j*ipPhase);  % modulation
    s = ip;

    u = s + 10^(-Es_N0_dB(ii)/20);
    SNR = 10^(Eb_N0_dB(ii)/10);

    a = gamrnd(m,omega_av/m,1,N);
    b = 1;%gamrnd(k,1/k,1,N);
    r = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
    n = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

    a1 = gamrnd(m1,omega_av/m1,1,N);
    b1 = 1;%gamrnd(k,1/k,1,N);
    r1 = sqrt(a1.*b1).*exp(2.*j.*pi.*rand(1,N));
    n1 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

    a2 = gamrnd(m2,omega_av/m2,1,N);
    b2 = 1;%gamrnd(k,1/k);
    r2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
    n2 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

    A1 = 0.5; %sqrt(1./((a1.*b1)+(1./(SNR))));
    y1 = conj(r).*((r.*u + n))./(r.*conj(r));
    y2 = conj(r1).*conj(r2).*(r2.*(A1.*(r1.*u + n1)+
n2))./(r1.*r2.*conj(r1).*conj(r2));
    y = y1+y2;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % ------------------------------------------------------------------
--------------- %
    % finding the phase from [-pi to +pi]
    opPhase = angle(y);
    % unwrapping the phase i.e. phase less than 0 are
    % added 2pi
    opPhase(find(opPhase<0)) = opPhase(find(opPhase<0)) + 2*pi;

    % rounding the received phase to the closest constellation
    ipPhaseHat = 2*pi/M*round(opPhase/(2*pi/M)) ;
    % as there is phase ambiguity for phase = 0 and 2*pi,
    % changing all phases reported as 2*pi to 0.
    % this is to enable comparison with the transmitted phase
    ipPhaseHat(find(ipPhaseHat==2*pi)) = 0;
    ipDecHat = round(ipPhaseHat*M/(2*pi));
```

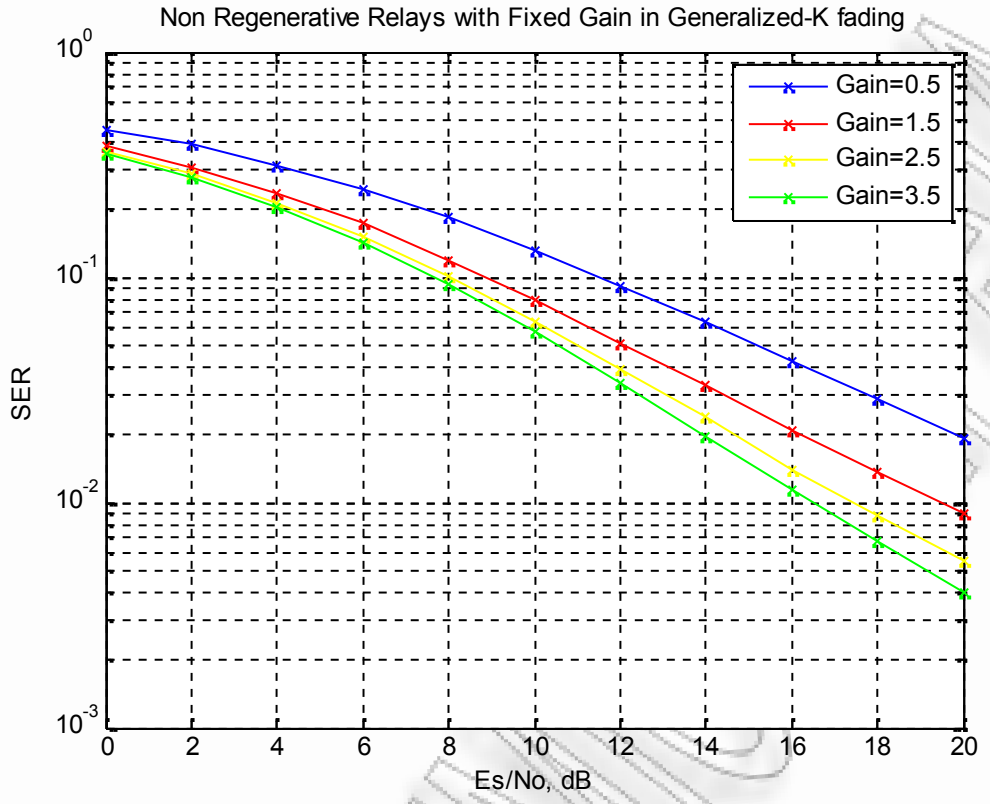
```
    % Decimal to Gray code conversion
    ipGrayHat = map(ipDecHat+1); % converting to decimal
    ipBinHat = dec2bin(ipGrayHat,kb) ; % decimal to binary

    % converting binary string to number
    ipBinHat = ipBinHat.';
    ipBinHat = ipBinHat(1:end).';
    ipBinHat = str2num(ipBinHat).' ;

    % counting errors
    nBitErr(ii) = size(find([ipBit- ipBinHat]),2); % couting the
number of errors
end
simBer = nBitErr/(N*kb);
```



Σχήμα 3.15 *BER of Fixed Gain Relay for BPSK modulation in Generalized-K fading channels*

3.4.4 Matlab script for BER of Fixed Gain Relay for BPSK in Generalized-K

```
SNR_dB = 0:2:20;
omega_av = 1;
m=1;
k=1;

m1=2;
k1=3;
m2=2;
k2=3;

N=10^4;

for i=1:length(SNR_dB)

    errors1=0;
    SNR = 10^(SNR_dB(i)/10);

        for j=1:N
            h = randint;
            u = 2*h-1;

            a = gamrnd(m,omega_av/m);
            b = gamrnd(k,1/k);
            r = sqrt(a*b);
            n = sqrt((omega_av/(2*SNR)))*randn;

            a1 = gamrnd(m1,omega_av/m1);
            b1 = gamrnd(k1,1/k1);
            r1 = sqrt(a1*b1);
            n1 = sqrt((omega_av/(2*SNR)))*randn;

            a2 = gamrnd(m2,omega_av/m2);
            b2 = gamrnd(k2,1/k2);
            r2 = sqrt(a2*b2);
            n2 = sqrt((omega_av/(2*SNR)))*randn;

            A = 0.5; %sqrt(1/((a1*b1)+(1/(SNR))));

            y = r*(r*u + n) + (r2*r1)*(r2*(A*(r1*u + n1)) + n2);
            if y >= 0.0
                h_r = 1;
            elseif y < 0
                h_r = 0;
            end

            errors1 = errors1 + (h_r~=h);
        end

ber(i) = errors1/N;
end
```

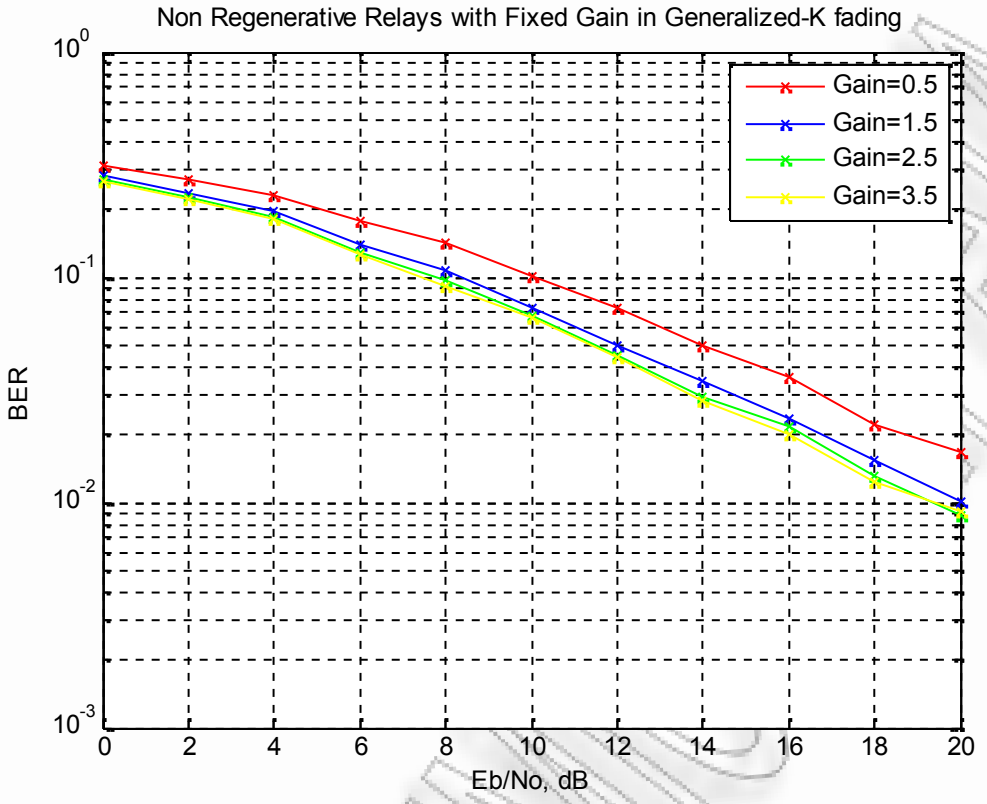


Σχήμα 3.16 *SER of Fixed Gain Relay for QPSK modulation in Generalized-K fading channels*

3.4.5 Matlab script for SER of Fixed Gain Relay for QPSK in Generalized-K

```
M = 4;
SNR_dB = 0:2:20;
omega_av = 1;
m = 1;
k = 1;

m1 = 2;
k1 = 3;
m2 = 2;
k2 = 3;

N = 1000;
loops = 1000;

for i=1:length(SNR_dB)

    errors_1 = 0;
    SNR = 10^(SNR_dB(i)/10);

    for z=1:loops

        h = randint(1,N,M);
        u = pskmod(h,M);
        a = gamrnd(m,omega_av/m,1,N);
```

```
        b = gamrnd(k,1/k,1,N);
        r = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
        n = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a1 = gamrnd(m1,omega_av/m1,1,N);
        b1 = gamrnd(k1,1/k1,1,N);
        r1 = sqrt(a1.*b1).*exp(2.*j.*pi.*rand(1,N));
        n1 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a2 = gamrnd(m2,omega_av/m2,1,N);
        b2 = gamrnd(k2,1/k2,1,N);
        r2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
        n2 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        %Amplify & Forward
        A = 0.5;%sqrt(1./((a1.*b1) + (1./(SNR))));

        y1 = conj(r).*((r.*u + n))./(r.*conj(r));
        y2 = conj(r1).*conj(r2).*(r2.*(A.*(r1.*u + n1) +
n2))./(r1.*r2.*conj(r1).*conj(r2));
        y = y1 + y2;
        %Demodulation
        demodmsg = pskdemod(y,M);
        [nes_amp,res_amp] = symerr(h,demodmsg);

        %Find total erros
        errors_1 = errors_1 + nes_amp;
    end

    ser_amp(i) = errors_1/(N*loops);
end
```



Σχήμα 3.17 *BER of Fixed Gain Relay for QPSK modulation in Generalized-K fading channels*

3.4.6 Matlab script for BER of Fixed Gain Relay for QPSK in Generalized-K

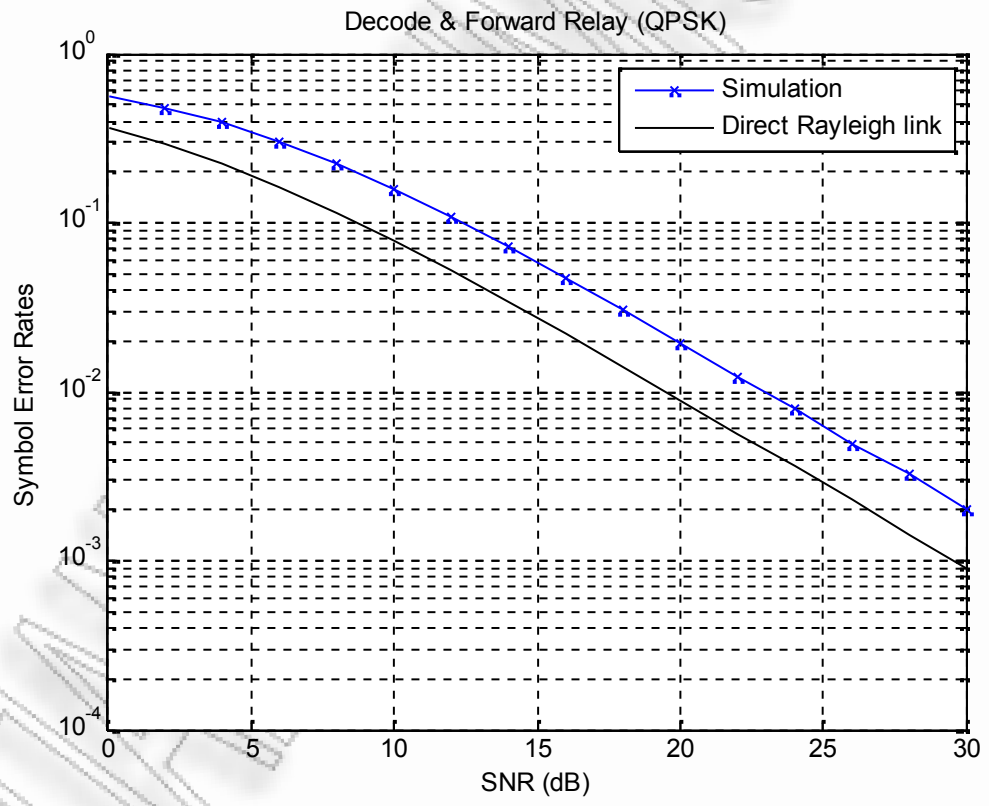
```
N = 10^4; % number of symbols
M = 4;    % constellation size
kb = log2(M); % bits per symbol

omega_av = 1;
m = 1;
k = 1;
m1 = 2;
k1 = 3;
m2 = 2;
k2 = 3;

thetaMpsk = [0:M-1]*2*pi/M; % reference phase values

Eb_N0_dB  = [0:2:20]; % multiple Es/N0 values
Es_N0_dB  = Eb_N0_dB + 10*log10(kb);

% Mapping for binary <--> Gray code conversion
ref = [0:M-1];
map = bitxor(ref,floor(ref/2));
[tt ind] = sort(map);

ipPhaseHat = zeros(1,N);
for ii = 1:length(Eb_N0_dB)
```

```
        % symbol generation
    % ------------------
    ipBit = rand(1,N*kb,1)>0.5; % random 1's and 0's
    bin2DecMatrix = ones(N,1)*(2.^[(kb-1):-1:0]) ; % conversion from
binary to decimal
    ipBitReshape =  reshape(ipBit,kb,N).'; % grouping to N symbols
having kb bits each
    ipGray = [sum(ipBitReshape.*bin2DecMatrix,2)].'; % decimal to
binary

    % Gray coded constellation mapping
    ipDec = ind(ipGray+1)-1; % bit group to constellation point
    ipPhase = ipDec*2*pi/M; % conversion to phase
    ip = exp(j*ipPhase);  % modulation
    s = ip;

    u = s + 10^(-Es_N0_dB(ii)/20);
    SNR = 10^(Eb_N0_dB(ii)/10);

    a = gamrnd(m,omega_av/m,1,N);
    b = gamrnd(k,1/k,1,N);
    r = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
    n = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

    a1 = gamrnd(m1,omega_av/m1,1,N);
    b1 = gamrnd(k,1/k,1,N);
    r1 = sqrt(a1.*b1).*exp(2.*j.*pi.*rand(1,N));
    n1 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

    a2 = gamrnd(m2,omega_av/m2,1,N);
    b2 = gamrnd(k,1/k);
    r2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
    n2 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

    A1 = 0.5; %sqrt(1./((a1.*b1)+(1./(SNR))));

    y1 = conj(r).*((r.*u + n))./(r.*conj(r));
    y2 = conj(r1).*conj(r2).*(r2.*(A1.*(r1.*u + n1)+
n2))./(r1.*r2.*conj(r1).*conj(r2));
    y = y1+y2;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % ------------------------------------------------------------------
---------------- %
    % finding the phase from [-pi to +pi]
    opPhase = angle(y);
    % unwrapping the phase i.e. phase less than 0 are
    % added 2pi
    opPhase(find(opPhase<0)) = opPhase(find(opPhase<0)) + 2*pi;

    % rounding the received phase to the closest constellation
    ipPhaseHat = 2*pi/M*round(opPhase/(2*pi/M)) ;
    % as there is phase ambiguity for phase = 0 and 2*pi,
    % changing all phases reported as 2*pi to 0.
    % this is to enable comparison with the transmitted phase
    ipPhaseHat(find(ipPhaseHat==2*pi)) = 0;
    ipDecHat = round(ipPhaseHat*M/(2*pi));
```

```
    % Decimal to Gray code conversion
    ipGrayHat = map(ipDecHat+1); % converting to decimal
    ipBinHat = dec2bin(ipGrayHat,kb) ; % decimal to binary

    % converting binary string to number
    ipBinHat = ipBinHat.';
    ipBinHat = ipBinHat(1:end).';
    ipBinHat = str2num(ipBinHat).' ;

    % counting errors
    nBitErr(ii) = size(find([ipBit- ipBinHat]),2); % couting the
number of errors
end
simBer = nBitErr/(N*kb);
```

3.5 Επίδοση Regenerative Relays (Decode and Forward)

Στα σχήματα που ακολουθούν μελετάται η επίδοση των Regenerative Relays, οι οποίοι αποκωδικοποιούν το εισερχόμενο σήμα και το κωδικοποιούν ξανά πριν το προωθήσουν στον επόμενο σταθμό. Αρχικά, παρουσιάζεται η επίδοση όσον αφορά στο Symbol Error Rate, του ψηφιακού αναμεταδότη σε σχέση με την απευθείας Rayleigh συνιστώσα, ενώ στη συνέχεια γίνονται συγκρίσεις των θεωρητικών με τα αποτελέσματα της προσομοίωσης, χρησιμοποιώντας τη μέθοδο Monte Carlo. Τέλος, γίνονται συγκρίσεις μεταξύ των ψηφιακών αναμεταδοτών και των αναλογικών (CSI). Η τιμή του κέρδους των αναλογικών αναμεταδοτών δίνεται από τη σχέση (3.10). Τα περιβάλλοντα διάδοσης χαρακτηρίζονται από διαλείψεις που ακολουθούν την κατανομή Generalized-K.

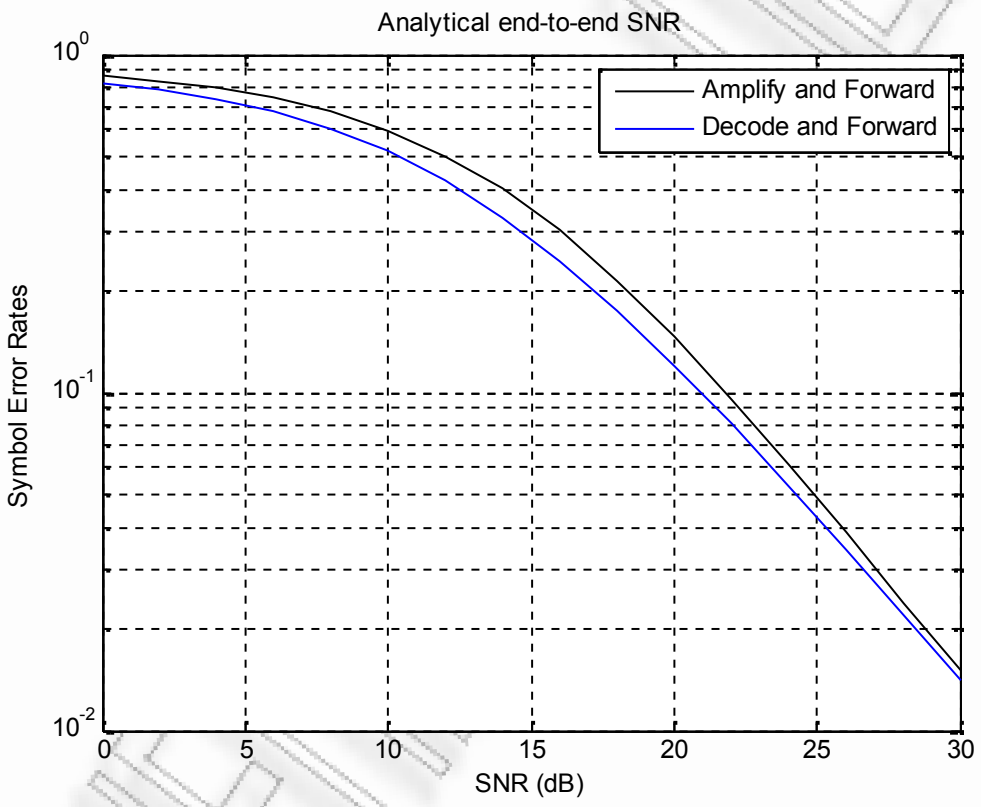


Σχήμα 3.18 *SER of Regenerative Relay for QPSK modulation in Generalized-K fading channel*

3.5.1. Matlab script for SER of Decode and Forward Relay for QPSK in Generalized-K

```
global g m m1 x L
M = 4;
g=(sin(pi/M))^2;
SNR_dB = 0:2:30; %SNR_dB = 0:5:40;
omega_av = 1;

L=1;
m = 1;
k = 10.5;
m1 = 1;
k1 = 10.5;
m2 = 1;
k2 = 10.5;
N = 500;
loops = 1000;

for i=1:length(SNR_dB)

        errors=0;
        errors1=0;
        SNR = 10^(SNR_dB(i)/10);
        x=SNR;
        for z=1:loops

            s = randint(1,N,M); % Random message
            u = pskmod(s,M,pi/M);     % Modulate using QPSK.

                a1 = gamrnd(m1,omega_av/m1, [1,N]);
                b1 = gamrnd(k1,1/k1, [1,N]);
                h1 = sqrt(a1.*b1).*exp(2.*j.*pi.*rand(1,N));
                n1 =
sqrt((omega_av./(2*SNR))).*(randn(1,N)+j*randn(1,N));

                a2 = gamrnd(m2,omega_av/m2, [1,N]);
                b2 = gamrnd(k2,1/k2, [1,N]);
                h2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
                n2 =
sqrt((omega_av./(2*SNR))).*(randn(1,N)+j*randn(1,N));

                %r0 = (h.*u + n);
                r1 = (h1.*u + n1);

                y1 =  conj(h1).*r1./(conj(h1).*h1);
                s1 = pskdemod(y1,M,pi/M);
                u1 = pskmod(s1,M, pi/M);
                [nes1,res1] = symerr(s,s1);
                errors1 = errors1 + nes1;

                r2 = h2.*u1 + n2;

                %y0 = conj(h).*r0./(conj(h).*h);

                y2 =  conj(h2).*r2./(conj(h2).*h2);

                %DEMODULATION using only the relay link
```

```
                s2 = pskdemod(y2,M,pi/M);
                [nes,res] = symerr(s,s2);

                % Find total errors from previous loops
                errors = errors + nes;

        end
ser1(i) = errors1/(N*loops);
ser(i) = errors/(N*loops);
ser_th(i) = (1/pi)*quadl(@mgf, 0, pi-pi/M, 10^(-6));
end
```

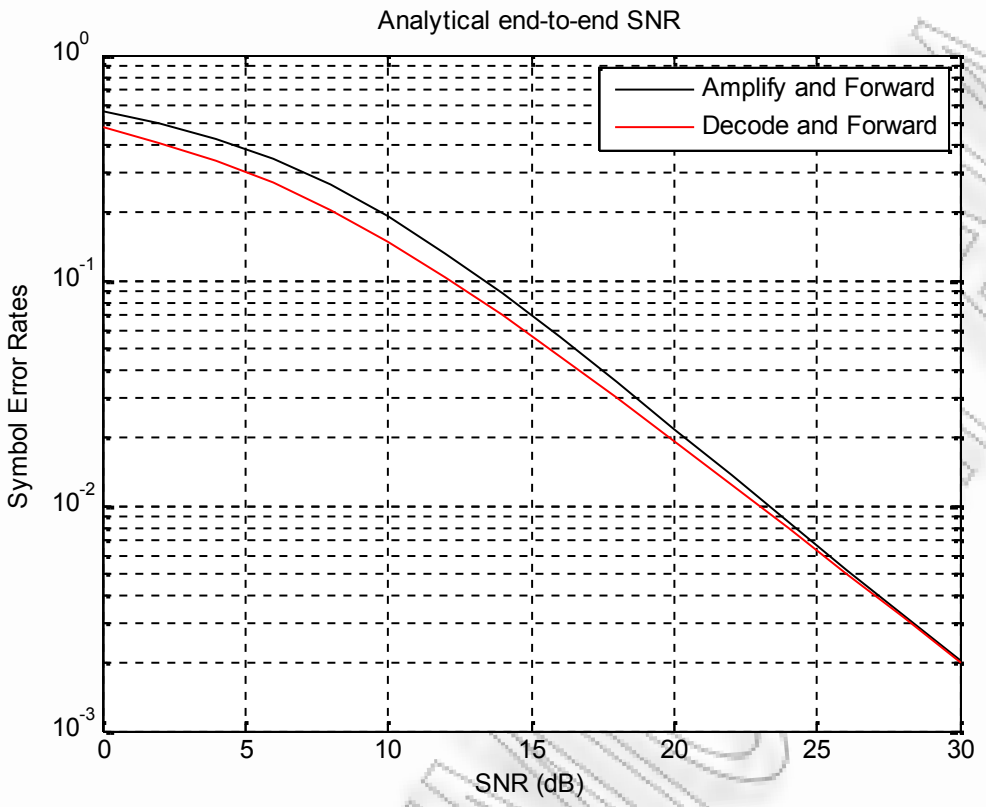


Σχήμα 3.19 *Theoretical results for 16QAM in Generalized-K fading channel*

3.5.2. Matlab script for 16QAM comparison in Generalized-K

```
SNRdB=0:2:30;
m1=1;
k1=10.5;

m2=1;
k2=10.5;

SNR=10.^(SNRdB/10);
```

```
N=1000;
itr=1000;
for step=1:itr

for kk= 1:1:length(SNRdB)

    h1=1;
    h2=1;

    h1=h1.*SNR(kk);
    h2=h2.*SNR(kk);

    R1m = gamrnd(m1,h1/m1, 1, N);
    R1k = gamrnd(k1,1/k1, 1, N);
    R1 = R1m.*R1k;

    R2m = gamrnd(m2,h2/m2, 1, N);
    R2k = gamrnd(k2,1/k2, 1, N);
    R2 = R2m.*R2k;

    eqv = 1./((1+1./R1).*(1+1./R2)-1);
    eqv_min = min(R1, R2);

     M=16;
     b=3/(M-1);
     EXACT_qam1=(4*(sqrt(M)-1)/sqrt(M))*qfunc(sqrt(b*eqv)) -
(4*(sqrt(M)-1).^2/M).*(qfunc(sqrt(b*eqv))).^2;
     Min_qam1=(4*(sqrt(M)-1)/sqrt(M))*qfunc(sqrt(b*eqv_min)) -
(4*(sqrt(M)-1).^2/M).*(qfunc(sqrt(b*eqv_min))).^2;
     ExactQAM1_N2(step,kk)=sum(EXACT_qam1)/N;
     MinQAM1_N2(step,kk)=sum(Min_qam1)/N;

end
end

ExactQAM1_N2=sum(ExactQAM1_N2)/itr
MinQAM1_N2=sum(MinQAM1_N2)/itr
```

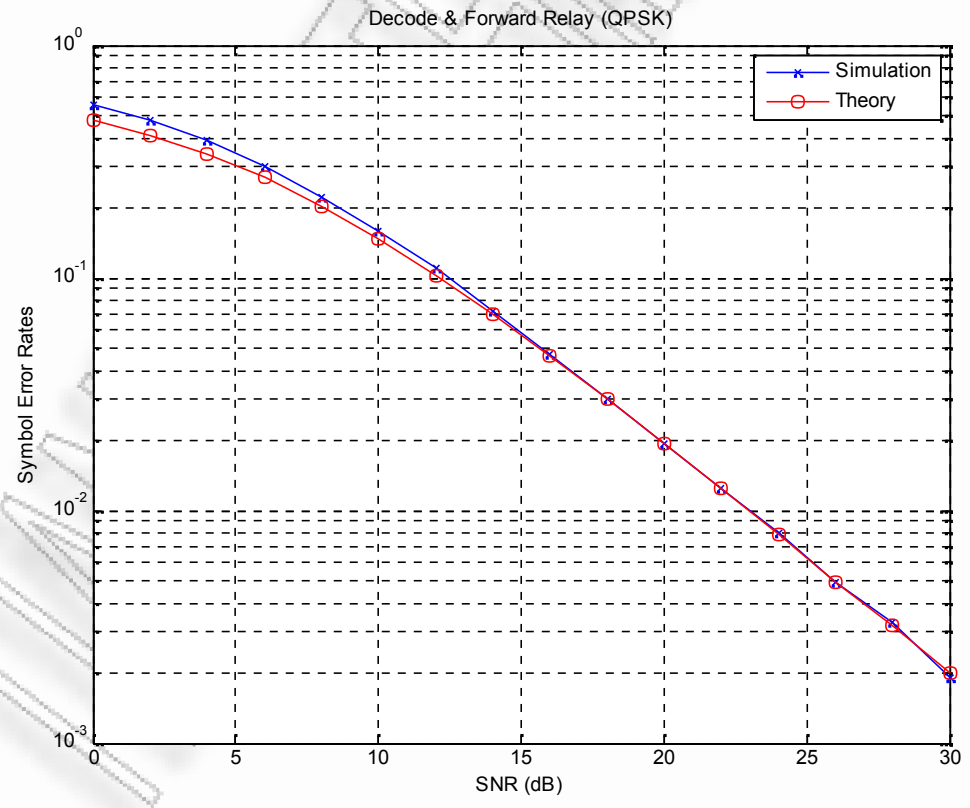


Σχήμα 3.20 *Theoretical results for QPSK in Generalized-K fading channel*

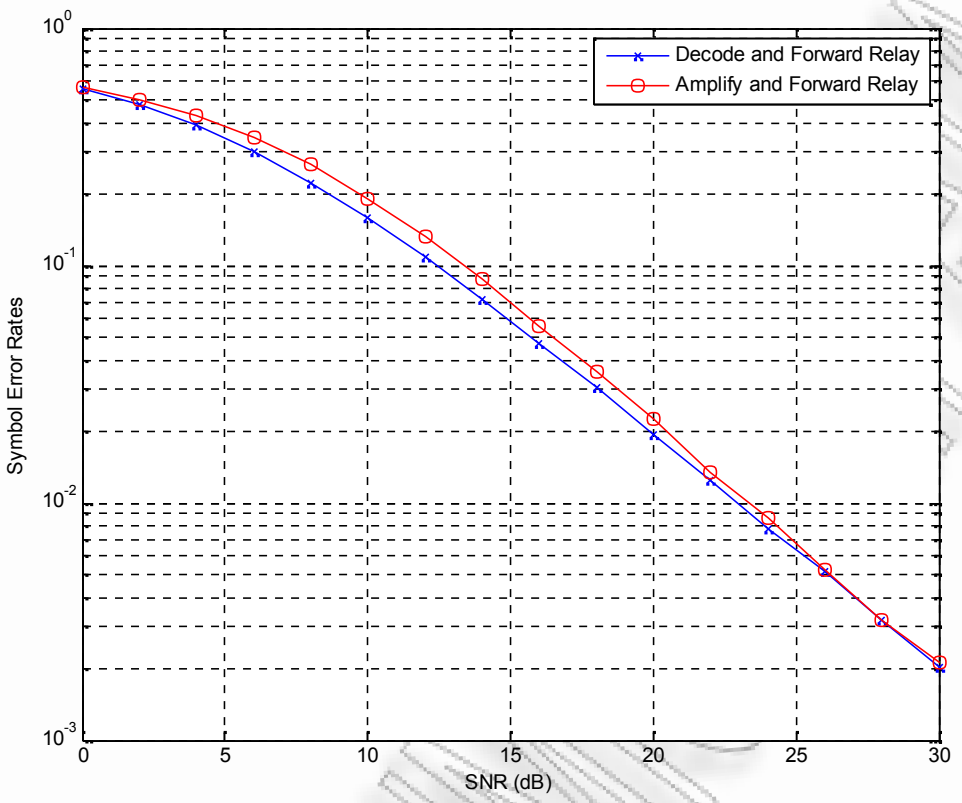
3.5.3. Matlab script for QPSK comparison in Generalized-K

```
%close all
clear all
%clc
SNRdB=0:2:30;

m1=1;
k1=10.5;

m2=1;
k2=10.5;

SNR=10.^(SNRdB/10);
N=1000;
itr=1000;
for step=1:itr

for kk= 1:1:length(SNRdB)

    h1=1;
    h2=1;

    h1=h1.*SNR(kk);
    h2=h2.*SNR(kk);
```

```
    R1m = gamrnd(m1,h1/m1, 1, N);
    R1k = gamrnd(k1,1/k1, 1, N);
    R1 = R1m.*R1k;

    R2m = gamrnd(m2,h2/m2, 1, N);
    R2k = gamrnd(k2,1/k2, 1, N);
    R2 = R2m.*R2k;

    eqv = 1./((1+1./R1).*(1+1./R2)-1);
    eqv_min = min(R1, R2);

    M=4;
    EXACT_qpsk1=2.*qfunc(sqrt(eqv))-(qfunc(sqrt(eqv))).^2;
    Min_qpsk1 = 2.*qfunc(sqrt(eqv_min))-(qfunc(sqrt(eqv_min))).^2;
    ExactQPSK1_N2(step,kk)=sum(EXACT_qpsk1)/N;
    MinQPSK1_N2(step,kk)=sum(Min_qpsk1)/N;

end
end

ExactQPSK1_N2=sum(ExactQPSK1_N2)/itr
MinQPSK1_N2=sum(MinQPSK1_N2)/itr
```



Σχήμα 3.21 *Simulation vs. Theoretical results in Generalized-K*



Σχήμα 3.22 *Decode and Forward vs. Amplify and Forward in Generalized-K*

Βιβλιογραφικές Αναφορές

- Hasna, O. M, 2003. *End-to-end Performance of Transmission Systems With Relays Over Rayleigh Fading Channels*. IEEE Transactions on Wireless Communications, Vol.2, No. 6
- Hasna, O. M, 2004. *A Performance Study of Dual-Hop Transmissions With Fixed-Gain Relays*. IEEE Transactions on Wireless Communications, Vol.3, No.6
- Efthymoglou, G., Bissias, N., Aalo, V., 2010. *On the Error Rate Analysis of Dual-Hop Amplify-and-Forward Relaying in Generalized-K Fading Channels*. Journal of Electrical and Computer Engineering, Volume 2010, Article ID 584594, Hindawi Publishing Corporation
- Tsiftis, T.A., Karagiannidis, G.K., Kotsopoulos, S.A., 2005. Dual-Hop wireless communications with combined gain relays. IEE Proceedings online no. 20045268, doi:10.1049/ip-com:20045268

Παράρτημα Α: Παράθεση Λογισμικού

1.1 Rayleigh Fading

```
% Matlab script for plotting the probability density
% function of Rayleigh random variable

close all
clear all
N = 10^6;
x = randn(1,N); % gaussian random variable, mean 0, variance 1
y = randn(1,N); % gaussian random variable, mean 0, variance 1
z = (x + j*y);  % complex random variable

%probability density function of abs(z)
zBin = [0:0.01:7];
sigma2 = 1;
pzTheory = (zBin/sigma2).*exp(-(zBin.^2)/(2*sigma2)); % theory
[nzSim zBinSim] = hist(abs(z),zBin); % simulation

% probability density of theta
%thetaBin = [-pi:0.01:pi];
%pThetaTheory = 1/(2*pi)*ones(size(thetaBin));
%[nThetaSim thetaBinSim] = hist(angle(z),thetaBin); % simulation

%figure
plot(zBinSim,nzSim/(N*0.01),'m');
hold on
plot(zBin,pzTheory,'b-')
xlabel('z');
ylabel('probability density, p(z)');
legend('simulation','theory');
title('Probability density function of abs(z)' )
axis([0 7 0 0.7]);
grid on

%figure
%plot(thetaBinSim,nThetaSim/(N*0.01),'m');
%hold on
%plot(thetaBin,pThetaTheory,'b-')
%xlabel('θ');
%ylabel('probability density, p(θ)');
%legend('simulation','theory');
%title('Probability density function of θ')
%axis([-pi pi 0 0.2])
%grid on
```

1.2 Rician Fading

```
function y = ricepdf(x, v, s)
% RICEPDF Rice/Rician probability density function (pdf).
% y = ricepdf(x, v, s) returns the pdf of the Rice (aka Rician)
% distribution with parameters v and s, evaluated at the values in x.

s2 = s.^2; % (neater below)

try
    y = (x ./ s2) .*...
        exp(-0.5 * (x.^2 + v.^2) ./ s2) .*...
        besseli(0, x .* v ./ s2);
        % besseli(0, ...) is the zeroth order modified Bessel
function of
        % the first kind. (see help bessel)
    y(x <= 0) = 0;
catch
    error('ricepdf:InputSizeMismatch',...
        'Non-scalar arguments must match in size.');
end
```

```
%% The Rician PDF
x = linspace(0, 8, 100);
close;
%subplot(3, 1, 1)
figure;
plot(x, ricepdf(x, 0, 1), x, ricepdf(x, 1, 1),...
     x, ricepdf(x, 2, 1), x, ricepdf(x, 4, 1))
%title('Rice PDF with σ = 1')
legend('v=0', 'v=1', 'v=2', 'v=4')
grid on;
%subplot(3,1,2)
figure;
plot(x, ricepdf(x, 1, 0.25), x, ricepdf(x, 1, 0.50),...
     x, ricepdf(x, 1, 1.00), x, ricepdf(x, 1, 2.00))
%title('Rice PDF with v = 1')
legend('σ=0.25', 'σ=0.50', 'σ=1.00', 'σ=2.00')
%subplot(3,1,3)
grid on;
```

1.3 Nakagami-m Fading

```
clear;
x=0:0.1:3;

m1=1;
m2=2;
m3=3;
m4=4;
m5=5;
omega1=1;
```

```
omega2=2;
omega3=3;

%y1=((2.*(m1.^m1))./(1.*(omega1.^m1))).*(x.^(2.*(m1-1))).*(exp(-
(m1.*(x.^2)./omega1)));
y2=((2.*(m2.^m2))./(1.*(omega2.^m2))).*(x.^(2.*(m2-1))).*(exp(-
(m2.*(x.^2)./omega2)));
y3=((2.*(m3.^m3))./(2.*(omega1.^m3))).*(x.^(2.*(m3-1))).*(exp(-
(m3.*(x.^2)./omega1)));
y4=((2.*(m4.^m4))./(6.*(omega2.^m4))).*(x.^(2.*(m4-1))).*(exp(-
(m4.*(x.^2)./omega2)));
y5=((2.*(m1.^m1))./(1.*(omega3.^m1))).*(x.^(2.*(m1-1))).*(exp(-
(m1.*(x.^2)./omega3)));

%plot(x,y1,'b-');
%hold on;
plot(x,y2,'r-');
hold on;
plot(x,y3,'g-');
hold on;
plot(x,y4,'y-');
hold on;
plot(x,y5,'c-');
legend('m=2, omega=2', 'm=3, omega=1','m=4, omega=2','m=1, omega=3')
%legend('m=2, omega=2', 'm=3, omega=1','m=4, omega=2')
ylabel('Probability Density Function')
grid on;
```

1.4 Composite Fading

```
clear;
x=0:0.1:7;
m=2;
k=2;
omega=1;
m1=3;
k1=3;
omega1=2;
m2=2;
k2=4;
omega2=1.5;
m3=2;
k3=3;
omega3=1.5;

nu=k-m;
z=2.*((m./omega).^(1/2)).*x;
Kei=besselk(nu,z);
nu1=k1-m1;
z1=2.*((m1./omega1).^(1/2)).*x;
Kei1=besselk(nu1,z1);
nu2=k2-m2;
z2=2.*((m2./omega2).^(1/2)).*x;
Kei2=besselk(nu2,z2);
```

```
nu3=k3-m3;
z3=2.*((m3./omega3).^(1/2)).*x;
Kei3=besselk(nu3,z3);

y=(((4.*m.^((k+m)./2))./(1.*1.*omega.^((k+m)./2))).*(x.^(k+m-
1))).*Kei;
y1=(((4.*m1.^((k1+m1)./2))./(2.*2.*omega1.^((k1+m1)./2))).*(x.^(k1+m1
-1))).*Kei1;
y2=(((4.*m2.^((k2+m2)./2))./(1.*6.*omega2.^((k2+m2)./2))).*(x.^(k2+m2
-1))).*Kei2;
y3=(((4.*m3.^((k3+m3)./2))./(1.*2.*omega3.^((k3+m3)./2))).*(x.^(k3+m3
-1))).*Kei3;

plot(x,y);
hold on;
plot(x,y1,'r-');
hold on;
plot(x,y2,'g-');
hold on;
plot(x,y3,'c-');
legend('m=2, k=2, omega=1','m=3, k=3, omega=2','m=2, k=4, omega=1.5',
'm=2, k=3, omega=1.5')
ylabel('Probability Density Function');
grid on;
```

2.1 Gamma PDF

```
clear;
x = 0:0.1:10;
omega = 1;
m1 = 1;
m2 = 2;
m3 = 3;
y = gampdf(x,m1,omega);
y2 = gampdf(x,m2,omega);
y3 = gampdf(x,m3,omega);

figure;
plot(x,y,'b-',x,y2,'r-',x,y3,'y-');
xlabel('y values');
ylabel('pdf');
legend('m=1 (exponential)','m=2','m=3');
grid on;
```

2.2 MGF method

```
function y=mgf(phi);
global gM m x;
tmp = sin(phi).*sin(phi);
y = (1 + (gM.*x./(m.*tmp))).^(-m);
```

- **2.1.3 Matlab script for SER with MGF method**

```
clear;
global gM m x;
m=4;
M=4;         %  QPSK
gM=sin(pi/M)*sin(pi/M);
aM=2;
for i=0:30       % i= SNR in dB
    x=10^(i/10);     % x=SNR in real number inside equations
ser(i+1) = (aM/pi)*quadl(@mgf,0,(pi/2),10^(-6));
snr(i+1)=i;
end
semilogy(snr, ser, '-')
xlabel('average SNR per symbol (dB)')
ylabel('Average SER')
axis([0, 30, 10^(-6), 10^(-0)])
```

- **2.2.1 Matlab script for MGF method vs. Simulation**

```
% Theoritical vs Simulation Results - MGF Method vs Simulation
% Rayleigh Fading (m=1)
% Any other value for m-> Nakagami-m
clear;
global g m m2 x L

M = 4;  % M=2 for BPSK
g = (sin(pi/M))^2; % g=1 for BPSK
SNR_dB = 0:2:30;
omega_av = 1;
L=2;

m = 1;
k = 10.5;
m2 = 1;
k2 = 10.5;

N = 1000;
loops = 800;

for i = 1:length(SNR_dB)

    errors = 0;
    SNR = 10^(SNR_dB(i)/10);
    x = SNR;

    for z = 1:loops
```

```
        s = randint(1,N,M); %Random message
        u = pskmod(s,M);    %Modulate using PSK

        a = gamrnd(m,omega_av/m, [1,N]);    % Direct channel
        b = gamrnd(k,1/k, [1,N]);
        h = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
        n = sqrt((omega_av./(2*SNR))).*(randn(1,N)+j*randn(1,N));

        a2 = gamrnd(m2,omega_av/m2, [1,N]); % Relay channel
        b2 = gamrnd(k2,1/k2, [1,N]);
        h2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
        n2 = sqrt((omega_av./(2*SNR))).*(randn(1,N)+j*randn(1,N));

        r0 = (h.*u + n);    % Direct
        r2 = (h2.*u + n2);  % Relay

        y0 = conj(h).*r0;
        y2 = conj(h2).*r2;
        y = ((y0 + y2)./(conj(h).*h + conj(h2).*h2))/L;

        %DEMODULATION
        demodmsg = pskdemod(y,M);
        [nes,res] = symerr(s,demodmsg);

        %FIND TOTAL ERRORS FROM PREVIOUS LOOPS
        errors = errors + nes;
    end

ser(i) = errors/(N*loops);  % Simulation
ser_th(i) = (1/pi)*quadl(@mgf, 0, pi-pi/M, 10^(-6));    % MGF
end

semilogy(SNR_dB, ser, 'r-+', SNR_dB, ser_th, 'k-');
legend('Simulation', 'Theory');
xlabel('SNR (dB)'); ylabel('SER');
axis([0 20 10^(-4.5) 10^(-0)]);
grid on;
```

3.1 Relays

- **3.3.1 Matlab script for SER of Non Regenerative CSI Relay in Nakagami-m**

```
clear;

M = 4;
SNR_dB = 0:2:20;
omega_av = 1;
%1st case
m = 1;
k = 1;
m1 = 2;
k1 = 3;
m2 = 2;
k2 = 3;
%2nd case
m3 = 4;
k3 = 2;
m4 = 6;
k4 = 5;
m5 = 6;
k5 = 5;
%3rd case
m6 = 6;
k6 = 3;
m7 = 8;
k7 = 6;
m8 = 8;
k8 = 6;
%4th case
m9 = 4;
k9 = 4;
m10 = 10;
k10 = 10;
m11 = 10;
k11 = 10;

N = 10000;
loops = 1000;

for i=1:length(SNR_dB)

    errors_1 = 0;
    errors_2 = 0;
    errors_3 = 0;
    errors_4 = 0;

    SNR = 10^(SNR_dB(i)/10);

    for z=1:loops

        h = randint(1,N,M);
        u = pskmod(h,M);

        %%%%%%%%%%%%%%%%%%1st CASE %%%%%%%%%%%%
```

```
a = gamrnd(m,omega_av/m,1,N);
b = 1;%gamrnd(k,1/k,1,N);
r = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
n = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

a1 = gamrnd(m1,omega_av/m1,1,N);
b1 = 1; %gamrnd(k1,1/k1,1,N);
r1 = sqrt(a1.*b1).*exp(2.*j.*pi.*rand(1,N));
n1 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

a2 = gamrnd(m2,omega_av/m2,1,N);
b2 = 1; %gamrnd(k2,1/k2,1,N);
r2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
n2 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

%%%%%%%%%%%% 2nd CASE %%%%%%%%%%%%%%%%%%
a3 = gamrnd(m3,omega_av/m3,1,N);
b3 = 1;%gamrnd(k,1/k,1,N);
r3 = sqrt(a3.*b3).*exp(2.*j.*pi.*rand(1,N));
n3 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

a4 = gamrnd(m4,omega_av/m4,1,N);
b4 = 1; %gamrnd(k1,1/k1,1,N);
r4 = sqrt(a4.*b4).*exp(2.*j.*pi.*rand(1,N));
n4 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

a5 = gamrnd(m5,omega_av/m5,1,N);
b5 = 1; %gamrnd(k2,1/k2,1,N);
r5 = sqrt(a5.*b5).*exp(2.*j.*pi.*rand(1,N));
n5 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

%%%%%%%%%%%%%%%%%%%%%%%% 3rd CASE%%%%%%%%%%%%%%
a6 = gamrnd(m6,omega_av/m6,1,N);
b6 = 1;%gamrnd(k,1/k,1,N);
r6 = sqrt(a6.*b6).*exp(2.*j.*pi.*rand(1,N));
n6 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

a7 = gamrnd(m7,omega_av/m7,1,N);
b7 = 1; %gamrnd(k1,1/k1,1,N);
r7 = sqrt(a7.*b7).*exp(2.*j.*pi.*rand(1,N));
n7 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

a8 = gamrnd(m8,omega_av/m8,1,N);
b8 = 1; %gamrnd(k2,1/k2,1,N);
r8 = sqrt(a8.*b8).*exp(2.*j.*pi.*rand(1,N));
n8 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

%%%%%%%%%%%%%%%% 4th CASE %%%%%%%%%%%%%%%
a9 = gamrnd(m9,omega_av/m9,1,N);
b9 = 1;%gamrnd(k,1/k,1,N);
r9 = sqrt(a9.*b9).*exp(2.*j.*pi.*rand(1,N));
n9 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

a10 = gamrnd(m10,omega_av/m10,1,N);
b10 = 1; %gamrnd(k1,1/k1,1,N);
r10 = sqrt(a4.*b4).*exp(2.*j.*pi.*rand(1,N));
n10 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));
```

```
        a11 = gamrnd(m11,omega_av/m11,1,N);
        b11 = 1; %gamrnd(k2,1/k2,1,N);
        r11 = sqrt(a11.*b11).*exp(2.*j.*pi.*rand(1,N));
        n11 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        %%%%% Amplify & Forward %%%%%%% 1st CASE %%%%%%%
        A = sqrt(1./((a1.*b1) + (1./(SNR))));
        y1 = conj(r).*((r.*u + n))./(r.*conj(r));
        y2 = conj(r1).*conj(r2).*(r2.*(A.*(r1.*u + n1) +
n2))./(r1.*r2.*conj(r1).*conj(r2));
        y = y1 + y2;

        %%%%% Amplify & Forward %%%%%%% 2nd CASE %%%%%%%
        A2 = sqrt(1./((a4.*b4) + (1./(SNR))));
        y3 = conj(r3).*((r3.*u + n))./(r3.*conj(r3));
        y4 = conj(r4).*conj(r5).*(r5.*(A2.*(r4.*u + n4) +
n5))./(r4.*r5.*conj(r4).*conj(r5));
        y5 = y3 + y4;

        %%%%% Amplify & Forward %%%%%%% 3rd CASE %%%%%%%
        A3 = sqrt(1./((a7.*b7) + (1./(SNR))));
        y6 = conj(r6).*((r6.*u + n))./(r6.*conj(r6));
        y7 = conj(r7).*conj(r8).*(r8.*(A3.*(r7.*u + n7) +
n8))./(r7.*r8.*conj(r7).*conj(r8));
        y8 = y6 + y7;

        %%%%% Amplify & Forward %%%%%%% 4th CASE %%%%%%%
        A4 = sqrt(1./((a11.*b11) + (1./(SNR))));
        y9 = conj(r9).*((r9.*u + n))./(r9.*conj(r9));
        y10 = conj(r10).*conj(r11).*(r11.*(A4.*(r10.*u + n10) +
n11))./(r10.*r11.*conj(r10).*conj(r11));
        y11 = y9 + y10;

        %%%%%% Demodulation CASE 1 %%%%%%%
        demodmsg = pskdemod(y,M);
        [nes_amp,res_amp] = symerr(h,demodmsg);

        %%%%%% Demodulation CASE 2 %%%%%%%
        demodmsg2 = pskdemod(y5,M);
        [nes_amp2,res_amp2] = symerr(h,demodmsg2);

        %%%%%% Demodulation CASE 3 %%%%%%%
        demodmsg3 = pskdemod(y8,M);
        [nes_amp3,res_amp3] = symerr(h,demodmsg3);

        %%%%%% Demodulation CASE 4 %%%%%%%
        demodmsg4 = pskdemod(y11,M);
        [nes_amp4,res_amp4] = symerr(h,demodmsg4);


        %Find total erros
        errors_1 = errors_1 + nes_amp;
        errors_2 = errors_2 + nes_amp2;
        errors_3 = errors_3 + nes_amp3;
        errors_4 = errors_4 + nes_amp4;
    end

    ser_amp(i) = errors_1/(N*loops);
```

```
    ser_amp2(i) = errors_2/(N*loops);
    ser_amp3(i) = errors_3/(N*loops);
    ser_amp4(i) = errors_4/(N*loops);
end

figure;
semilogy(SNR_dB, ser_amp, 'b-',SNR_dB, ser_amp2, 'rx-',SNR_dB,
ser_amp3, 'go-',SNR_dB, ser_amp4, 'y+-');
axis([0 20 10^(-5) 1])
grid on;
legend ('m=1, m1=m2=2','m=4, m1=m2=6','m=6, m1=m2=8','m=4,
m1=m2=10');
xlabel('Es/No, dB'); ylabel('SER');
title('Non Regenerative CSI Relay in Nakagami-m fading channels');

%figure;
%bins=200;
%[j1 xout]=hist(A,bins);
%bar(xout, j1/(N*max(xout)/bins))
%h=findobj(gca,'Type','patch');
%set(h,'Facecolor','r','Linestyle',':','Edgecolor','w')
%plot(A);
%title('Gain Values for Non Regenerative CSI Relays');
%xlabel('Number of Samples');
%ylabel('Gain');

%figure;
%bins=200;
%[j2 xout2]=hist(A2,bins);
%bar(xout2, j2/(N*max(xout2)/bins))
%h2=findobj(gca,'Type','patch');
%set(h2,'Facecolor','r','Linestyle',':','Edgecolor','w')
%plot(A2);
%title('Gain Values for Non Regenerative CSI Relays');
%xlabel('Number of Samples');
%ylabel('Gain');

%figure;
%bins=200;
%[j3 xout3]=hist(A3,bins);
%bar(xout3, j3/(N*max(xout3)/bins))
%h3=findobj(gca,'Type','patch');
%set(h3,'Facecolor','r','Linestyle',':','Edgecolor','w')
%plot(A3);
%title('Gain Values for Non Regenerative CSI Relays');
%xlabel('Number of Samples');
%ylabel('Gain');

%figure;
%bins=200;
%[j4 xout4]=hist(A4,bins);
%bar(xout4, j4/(N*max(xout4)/bins))
%h4=findobj(gca,'Type','patch');
%set(h4,'Facecolor','r','Linestyle',':','Edgecolor','w')
%plot(A);
%title('Gain Values for Non Regenerative CSI Relays');
%xlabel('Number of Samples');
%ylabel('Gain');
```

- **3.3.2 Matlab script for BER of Non Regenerative CSI Relay in Nakagami-m**

```
clear
N = 10^5; % number of symbols
M = 4;   % constellation size
kb = log2(M); % bits per symbol
omega_av = 1;

%1st case
m = 1;
k = 1;
m1 = 2;
k1 = 3;
m2 = 2;
k2 = 3;
%2nd case
m3 = 4;
k3 = 2;
m4 = 6;
k4 = 5;
m5 = 6;
k5 = 5;
%3rd case
m6 = 6;
k6 = 3;
m7 = 8;
k7 = 6;
m8 = 8;
k8 = 6;
%4th case
m9 = 4;
k9 = 4;
m10 = 10;
k10 = 10;
m11 = 10;
k11 = 10;

thetaMpsk = [0:M-1]*2*pi/M; % reference phase values

Eb_N0_dB  = [0:2:20]; % multiple Es/N0 values
Es_N0_dB  = Eb_N0_dB + 10*log10(kb);

% Mapping for binary <--> Gray code conversion
ref = [0:M-1];
map = bitxor(ref,floor(ref/2));
[tt ind] = sort(map);

ipPhaseHat = zeros(1,N);
for ii = 1:length(Eb_N0_dB)


    % symbol generation
    % -----------------
    ipBit = rand(1,N*kb,1)>0.5; % random 1's and 0's
    bin2DecMatrix = ones(N,1)*(2.^[(kb-1):-1:0]) ; % conversion from
binary to decimal
    ipBitReshape =  reshape(ipBit,kb,N).'; % grouping to N symbols
having kb bits each
```

```
    ipGray = [sum(ipBitReshape.*bin2DecMatrix,2)].'; % decimal to
binary

    % Gray coded constellation mapping
    ipDec = ind(ipGray+1)-1; % bit group to constellation point
    ipPhase = ipDec*2*pi/M; % conversion to phase
    ip = exp(j*ipPhase);  % modulation
    s = ip;

    u = s + 10^(-Es_N0_dB(ii)/20);
    SNR = 10^(Eb_N0_dB(ii)/10);

    %%%%%%%%%%%%%%%%%%1st CASE %%%%%%%%%%%%
        a = gamrnd(m,omega_av/m,1,N);
        b = 1;%gamrnd(k,1/k,1,N);
        r = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
        n = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a1 = gamrnd(m1,omega_av/m1,1,N);
        b1 = 1; %gamrnd(k1,1/k1,1,N);
        r1 = sqrt(a1.*b1).*exp(2.*j.*pi.*rand(1,N));
        n1 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a2 = gamrnd(m2,omega_av/m2,1,N);
        b2 = 1; %gamrnd(k2,1/k2,1,N);
        r2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
        n2 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        %%%%%%%%%%%% 2nd CASE %%%%%%%%%%%%%%%%%%
        a3 = gamrnd(m3,omega_av/m3,1,N);
        b3 = 1;%gamrnd(k,1/k,1,N);
        r3 = sqrt(a3.*b3).*exp(2.*j.*pi.*rand(1,N));
        n3 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a4 = gamrnd(m4,omega_av/m4,1,N);
        b4 = 1; %gamrnd(k1,1/k1,1,N);
        r4 = sqrt(a4.*b4).*exp(2.*j.*pi.*rand(1,N));
        n4 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a5 = gamrnd(m5,omega_av/m5,1,N);
        b5 = 1; %gamrnd(k2,1/k2,1,N);
        r5 = sqrt(a5.*b5).*exp(2.*j.*pi.*rand(1,N));
        n5 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        %%%%%%%%%%%%%%%%%%%%%%%% 3rd CASE%%%%%%%%%%%%%%
        a6 = gamrnd(m6,omega_av/m6,1,N);
        b6 = 1;%gamrnd(k,1/k,1,N);
        r6 = sqrt(a6.*b6).*exp(2.*j.*pi.*rand(1,N));
        n6 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a7 = gamrnd(m7,omega_av/m7,1,N);
        b7 = 1; %gamrnd(k1,1/k1,1,N);
        r7 = sqrt(a7.*b7).*exp(2.*j.*pi.*rand(1,N));
        n7 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a8 = gamrnd(m8,omega_av/m8,1,N);
        b8 = 1; %gamrnd(k2,1/k2,1,N);
        r8 = sqrt(a8.*b8).*exp(2.*j.*pi.*rand(1,N));
```

```
        n8 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        %%%%%%%%%%%%%%%% 4th CASE %%%%%%%%%%%%%%%
        a9 = gamrnd(m9,omega_av/m9,1,N);
        b9 = 1;%gamrnd(k,1/k,1,N);
        r9 = sqrt(a9.*b9).*exp(2.*j.*pi.*rand(1,N));
        n9 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a10 = gamrnd(m10,omega_av/m10,1,N);
        b10 = 1; %gamrnd(k1,1/k1,1,N);
        r10 = sqrt(a10.*b10).*exp(2.*j.*pi.*rand(1,N));
        n10 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a11 = gamrnd(m11,omega_av/m11,1,N);
        b11 = 1; %gamrnd(k2,1/k2,1,N);
        r11 = sqrt(a11.*b11).*exp(2.*j.*pi.*rand(1,N));
        n11 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));



        %%%%% Amplify & Forward %%%%%%% 1st CASE %%%%%%%%
        A = sqrt(1./((a1.*b1) + (1./(SNR))));
        y1 = conj(r).*((r.*u + n))./(r.*conj(r));
        y2 = conj(r1).*conj(r2).*(r2.*(A.*(r1.*u + n1) +
n2))./(r1.*r2.*conj(r1).*conj(r2));
        y = y1 + y2;

        %%%%% Amplify & Forward %%%%%%% 2nd CASE %%%%%%%%
        A2 = sqrt(1./((a4.*b4) + (1./(SNR))));
        y3 = conj(r3).*((r3.*u + n))./(r3.*conj(r3));
        y4 = conj(r4).*conj(r5).*(r5.*(A2.*(r4.*u + n4) +
n5))./(r4.*r5.*conj(r4).*conj(r5));
        y5 = y3 + y4;

        %%%%% Amplify & Forward %%%%%%% 3rd CASE %%%%%%%%
        A3 = sqrt(1./((a7.*b7) + (1./(SNR))));
        y6 = conj(r6).*((r6.*u + n))./(r6.*conj(r6));
        y7 = conj(r7).*conj(r8).*(r8.*(A3.*(r7.*u + n7) +
n8))./(r7.*r8.*conj(r7).*conj(r8));
        y8 = y6 + y7;

        %%%%% Amplify & Forward %%%%%%% 4th CASE %%%%%%%%
        A4 = sqrt(1./((a11.*b11) + (1./(SNR))));
        y9 = conj(r9).*((r9.*u + n))./(r9.*conj(r9));
        y10 = conj(r10).*conj(r11).*(r11.*(A4.*(r10.*u + n10) +
n11))./(r10.*r11.*conj(r10).*conj(r11));
        y11 = y9 + y10;


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION 1st CASE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % ------------------------------------------------------------
--------------- %
    % finding the phase from [-pi to +pi]
    opPhase = angle(y);
    % unwrapping the phase i.e. phase less than 0 are
    % added 2pi
    opPhase(find(opPhase<0)) = opPhase(find(opPhase<0)) + 2*pi;
```

```
    % rounding the received phase to the closest constellation
    ipPhaseHat = 2*pi/M*round(opPhase/(2*pi/M)) ;
    % as there is phase ambiguity for phase = 0 and 2*pi,
    % changing all phases reported as 2*pi to 0.
    % this is to enable comparison with the transmitted phase
    ipPhaseHat(find(ipPhaseHat==2*pi)) = 0;
    ipDecHat = round(ipPhaseHat*M/(2*pi));

    % Decimal to Gray code conversion
    ipGrayHat = map(ipDecHat+1); % converting to decimal
    ipBinHat = dec2bin(ipGrayHat,kb) ; % decimal to binary

    % converting binary string to number
    ipBinHat = ipBinHat.';
    ipBinHat = ipBinHat(1:end).';
    ipBinHat = str2num(ipBinHat).' ;

    % counting errors
    nBitErr(ii) = size(find([ipBit- ipBinHat]),2); % couting the
number of errors

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION 2nd CASE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % ---------------------------------------------------------------
--------------- %
    % finding the phase from [-pi to +pi]
    opPhase2 = angle(y5);
    % unwrapping the phase i.e. phase less than 0 are
    % added 2pi
    opPhase2(find(opPhase2<0)) = opPhase2(find(opPhase2<0)) + 2*pi;

    % rounding the received phase to the closest constellation
    ipPhaseHat2 = 2*pi/M*round(opPhase2/(2*pi/M)) ;
    % as there is phase ambiguity for phase = 0 and 2*pi,
    % changing all phases reported as 2*pi to 0.
    % this is to enable comparison with the transmitted phase
    ipPhaseHat2(find(ipPhaseHat2==2*pi)) = 0;
    ipDecHat2 = round(ipPhaseHat2*M/(2*pi));

    % Decimal to Gray code conversion
    ipGrayHat2 = map(ipDecHat2+1); % converting to decimal
    ipBinHat2 = dec2bin(ipGrayHat2,kb) ; % decimal to binary

    % converting binary string to number
    ipBinHat2 = ipBinHat2.';
    ipBinHat2 = ipBinHat2(1:end).';
    ipBinHat2 = str2num(ipBinHat2).' ;

    % counting errors
    nBitErr2(ii) = size(find([ipBit- ipBinHat2]),2); % couting the
number of errors

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION 3rd CASE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % ---------------------------------------------------------------
--------------- %
    % finding the phase from [-pi to +pi]
```

```
    opPhase3 = angle(y8);
    % unwrapping the phase i.e. phase less than 0 are
    % added 2pi
    opPhase3(find(opPhase3<0)) = opPhase3(find(opPhase3<0)) + 2*pi;

    % rounding the received phase to the closest constellation
    ipPhaseHat3 = 2*pi/M*round(opPhase3/(2*pi/M)) ;
    % as there is phase ambiguity for phase = 0 and 2*pi,
    % changing all phases reported as 2*pi to 0.
    % this is to enable comparison with the transmitted phase
    ipPhaseHat3(find(ipPhaseHat3==2*pi)) = 0;
    ipDecHat3 = round(ipPhaseHat3*M/(2*pi));

    % Decimal to Gray code conversion
    ipGrayHat3 = map(ipDecHat3+1); % converting to decimal
    ipBinHat3 = dec2bin(ipGrayHat3,kb) ; % decimal to binary

    % converting binary string to number
    ipBinHat3 = ipBinHat3.';
    ipBinHat3 = ipBinHat3(1:end).';
    ipBinHat3 = str2num(ipBinHat3).' ;

    % counting errors
    nBitErr3(ii) = size(find([ipBit- ipBinHat3]),2); % couting the
number of errors

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION 4th CASE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % ---------------------------------------------------------------
--------------- %
    % finding the phase from [-pi to +pi]
    opPhase4 = angle(y11);
    % unwrapping the phase i.e. phase less than 0 are
    % added 2pi
    opPhase4(find(opPhase4<0)) = opPhase4(find(opPhase4<0)) + 2*pi;

    % rounding the received phase to the closest constellation
    ipPhaseHat4 = 2*pi/M*round(opPhase4/(2*pi/M)) ;
    % as there is phase ambiguity for phase = 0 and 2*pi,
    % changing all phases reported as 2*pi to 0.
    % this is to enable comparison with the transmitted phase
    ipPhaseHat4(find(ipPhaseHat4==2*pi)) = 0;
    ipDecHat4 = round(ipPhaseHat4*M/(2*pi));

    % Decimal to Gray code conversion
    ipGrayHat4 = map(ipDecHat4+1); % converting to decimal
    ipBinHat4 = dec2bin(ipGrayHat4,kb) ; % decimal to binary

    % converting binary string to number
    ipBinHat4 = ipBinHat4.';
    ipBinHat4 = ipBinHat4(1:end).';
    ipBinHat4 = str2num(ipBinHat4).' ;

    % counting errors
    nBitErr4(ii) = size(find([ipBit- ipBinHat4]),2); % couting the
number of errors

end
```

```
simBer = nBitErr/(N*kb);
simBer2 = nBitErr2/(N*kb);
simBer3 = nBitErr3/(N*kb);
simBer4 = nBitErr4/(N*kb);

semilogy(Eb_N0_dB,simBer,'r-',Eb_N0_dB,simBer2,'bx-',
Eb_N0_dB,simBer3,'g+-', Eb_N0_dB,simBer4,'yo-');
grid on;
axis([0 20 10^(-4) 1]);
legend ('m=1, m1=m2=2','m=4, m1=m2=6','m=6, m1=m2=8','m=4,
m1=m2=10');
xlabel('Eb/No, dB'); ylabel('BER');
title('Non Regenerative CSI Relay in Nakagami-m fading');
```

- **3.3.3 Matlab script for SER of Non Regenerative CSI Relay in Generalized-K**

```
clear;

M = 4;
SNR_dB = 0:2:20;
omega_av = 1;
%1st case
m = 1;
k = 1;
m1 = 2;
k1 = 3;
m2 = 2;
k2 = 3;
%2nd case
m3 = 4;
k3 = 2;
m4 = 6;
k4 = 5;
m5 = 6;
k5 = 5;
%3rd case
m6 = 6;
k6 = 3;
m7 = 8;
k7 = 6;
m8 = 8;
k8 = 6;
%4th case
m9 = 4;
k9 = 4;
m10 = 10;
k10 = 10;
m11 = 10;
k11 = 10;

N = 1000;
loops = 1000;
```

```
for i=1:length(SNR_dB)

    errors_1 = 0;
    errors_2 = 0;
    errors_3 = 0;
    errors_4 = 0;

    SNR = 10^(SNR_dB(i)/10);

    for z=1:loops

        h = randint(1,N,M);
        u = pskmod(h,M);

        %%%%%%%%%%%%%%%%%%1st CASE %%%%%%%%%%%
        a = gamrnd(m,omega_av/m,1,N);
        b = gamrnd(k,1/k,1,N);
        r = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
        n = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a1 = gamrnd(m1,omega_av/m1,1,N);
        b1 = gamrnd(k1,1/k1,1,N);
        r1 = sqrt(a1.*b1).*exp(2.*j.*pi.*rand(1,N));
        n1 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a2 = gamrnd(m2,omega_av/m2,1,N);
        b2 = gamrnd(k2,1/k2,1,N);
        r2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
        n2 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        %%%%%%%%%%%% 2nd CASE %%%%%%%%%%%%%%%%%
        a3 = gamrnd(m3,omega_av/m3,1,N);
        b3 = gamrnd(k3,1/k3,1,N);
        r3 = sqrt(a3.*b3).*exp(2.*j.*pi.*rand(1,N));
        n3 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a4 = gamrnd(m4,omega_av/m4,1,N);
        b4 = gamrnd(k4,1/k4,1,N);
        r4 = sqrt(a4.*b4).*exp(2.*j.*pi.*rand(1,N));
        n4 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a5 = gamrnd(m5,omega_av/m5,1,N);
        b5 = gamrnd(k5,1/k5,1,N);
        r5 = sqrt(a5.*b5).*exp(2.*j.*pi.*rand(1,N));
        n5 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        %%%%%%%%%%%%%%%%%%%%%%%% 3rd CASE%%%%%%%%%%%%%%
        a6 = gamrnd(m6,omega_av/m6,1,N);
        b6 = gamrnd(k6,1/k6,1,N);
        r6 = sqrt(a6.*b6).*exp(2.*j.*pi.*rand(1,N));
        n6 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a7 = gamrnd(m7,omega_av/m7,1,N);
        b7 = gamrnd(k7,1/k7,1,N);
        r7 = sqrt(a7.*b7).*exp(2.*j.*pi.*rand(1,N));
        n7 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));
```

```
        a8 = gamrnd(m8,omega_av/m8,1,N);
        b8 = gamrnd(k8,1/k8,1,N);
        r8 = sqrt(a8.*b8).*exp(2.*j.*pi.*rand(1,N));
        n8 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        %%%%%%%%%%%%%%%% 4th CASE %%%%%%%%%%%%%%%
        a9 = gamrnd(m9,omega_av/m9,1,N);
        b9 = gamrnd(k9,1/k9,1,N);
        r9 = sqrt(a9.*b9).*exp(2.*j.*pi.*rand(1,N));
        n9 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a10 = gamrnd(m10,omega_av/m10,1,N);
        b10 = gamrnd(k10,1/k10,1,N);
        r10 = sqrt(a4.*b4).*exp(2.*j.*pi.*rand(1,N));
        n10 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a11 = gamrnd(m11,omega_av/m11,1,N);
        b11 = gamrnd(k11,1/k11,1,N);
        r11 = sqrt(a11.*b11).*exp(2.*j.*pi.*rand(1,N));
        n11 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        %%%%% Amplify & Forward %%%%%%% 1st CASE %%%%%%%%
        A = sqrt(1./((a1.*b1) + (1./(SNR))));
        y1 = conj(r).*((r.*u + n))./(r.*conj(r));
        y2 = conj(r1).*conj(r2).*(r2.*(A.*(r1.*u + n1) +
n2))./(r1.*r2.*conj(r1).*conj(r2));
        y = y1 + y2;

        %%%%% Amplify & Forward %%%%%%% 2nd CASE %%%%%%%%
        A2 = sqrt(1./((a4.*b4) + (1./(SNR))));
        y3 = conj(r3).*((r3.*u + n))./(r3.*conj(r3));
        y4 = conj(r4).*conj(r5).*(r5.*(A2.*(r4.*u + n4) +
n5))./(r4.*r5.*conj(r4).*conj(r5));
        y5 = y3 + y4;

        %%%%% Amplify & Forward %%%%%%% 3rd CASE %%%%%%%%
        A3 = sqrt(1./((a7.*b7) + (1./(SNR))));
        y6 = conj(r6).*((r6.*u + n))./(r6.*conj(r6));
        y7 = conj(r7).*conj(r8).*(r8.*(A3.*(r7.*u + n7) +
n8))./(r7.*r8.*conj(r7).*conj(r8));
        y8 = y6 + y7;

        %%%%% Amplify & Forward %%%%%%% 4th CASE %%%%%%%%
        A4 = sqrt(1./((a11.*b11) + (1./(SNR))));
        y9 = conj(r9).*((r9.*u + n))./(r9.*conj(r9));
        y10 = conj(r10).*conj(r11).*(r11.*(A4.*(r10.*u + n10) +
n11))./(r10.*r11.*conj(r10).*conj(r11));
        y11 = y9 + y10;

        %%%%%% Demodulation CASE 1 %%%%%%%
        demodmsg = pskdemod(y,M);
        [nes_amp,res_amp] = symerr(h,demodmsg);

        %%%%%% Demodulation CASE 2 %%%%%%%
        demodmsg2 = pskdemod(y5,M);
        [nes_amp2,res_amp2] = symerr(h,demodmsg2);

        %%%%%% Demodulation CASE 3 %%%%%%%
```

```
        demodmsg3 = pskdemod(y8,M);
        [nes_amp3,res_amp3] = symerr(h,demodmsg3);

        %%%%%% Demodulation CASE 4 %%%%%%%
        demodmsg4 = pskdemod(y11,M);
        [nes_amp4,res_amp4] = symerr(h,demodmsg4);


        %Find total erros
        errors_1 = errors_1 + nes_amp;
        errors_2 = errors_2 + nes_amp2;
        errors_3 = errors_3 + nes_amp3;
        errors_4 = errors_4 + nes_amp4;
    end

    ser_amp(i) = errors_1/(N*loops);
    ser_amp2(i) = errors_2/(N*loops);
    ser_amp3(i) = errors_3/(N*loops);
    ser_amp4(i) = errors_4/(N*loops);
end

figure;
semilogy(SNR_dB, ser_amp, 'b-',SNR_dB, ser_amp2, 'rx-',SNR_dB,
ser_amp3, 'go-',SNR_dB, ser_amp4, 'y+-');
axis([0 20 10^(-5) 1]);
grid on;
legend ('m=1, m1=m2=2, k=1, k1=k2=3','m=4, m1=m2=6, k=2,
k1=k2=5','m=6, m1=m2=8, k=3, k1=k2=6','m=4, m1=m2=10, k=4,
k1=k2=10','Location','SouthWest');
xlabel('Es/No, dB'); ylabel('SER');
title('Non Regenerative CSI Relay in Generalized-K fading');
```

- **3.3.4 Matlab script for BER of Non Regenerative CSI Relay in Generalized-K**

```
clear
N = 10^5; % number of symbols
M = 4;   % constellation size
kb = log2(M); % bits per symbol
omega_av = 1;

%1st case
m = 1;
k = 1;
m1 = 2;
k1 = 3;
m2 = 2;
k2 = 3;
%2nd case
m3 = 4;
k3 = 2;
m4 = 6;
k4 = 5;
m5 = 6;
```

```
k5 = 5;
%3rd case
m6 = 6;
k6 = 3;
m7 = 8;
k7 = 6;
m8 = 8;
k8 = 6;
%4th case
m9 = 4;
k9 = 4;
m10 = 10;
k10 = 10;
m11 = 10;
k11 = 10;

thetaMpsk = [0:M-1]*2*pi/M; % reference phase values

Eb_N0_dB  = [0:2:20]; % multiple Es/N0 values
Es_N0_dB  = Eb_N0_dB + 10*log10(kb);

% Mapping for binary <--> Gray code conversion
ref = [0:M-1];
map = bitxor(ref,floor(ref/2));
[tt ind] = sort(map);

ipPhaseHat = zeros(1,N);
for ii = 1:length(Eb_N0_dB)


    % symbol generation
    % -----------------
    ipBit = rand(1,N*kb,1)>0.5; % random 1's and 0's
    bin2DecMatrix = ones(N,1)*(2.^[(kb-1):-1:0]) ; % conversion from
binary to decimal
    ipBitReshape =  reshape(ipBit,kb,N).'; % grouping to N symbols
having kb bits each
    ipGray = [sum(ipBitReshape.*bin2DecMatrix,2)].'; % decimal to
binary

    % Gray coded constellation mapping
    ipDec = ind(ipGray+1)-1; % bit group to constellation point
    ipPhase = ipDec*2*pi/M; % conversion to phase
    ip = exp(j*ipPhase);  % modulation
    s = ip;

    u = s + 10^(-Es_N0_dB(ii)/20);
    SNR = 10^(Eb_N0_dB(ii)/10);

    %%%%%%%%%%%%%%%%%1st CASE %%%%%%%%%%%%
        a = gamrnd(m,omega_av/m,1,N);
        b = gamrnd(k,1/k,1,N);
        r = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
        n = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a1 = gamrnd(m1,omega_av/m1,1,N);
        b1 = gamrnd(k1,1/k1,1,N);
        r1 = sqrt(a1.*b1).*exp(2.*j.*pi.*rand(1,N));
```

```
n1 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

a2 = gamrnd(m2,omega_av/m2,1,N);
b2 = gamrnd(k2,1/k2,1,N);
r2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
n2 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

%%%%%%%%%%%% 2nd CASE %%%%%%%%%%%%%%%%%%
a3 = gamrnd(m3,omega_av/m3,1,N);
b3 = gamrnd(k3,1/k3,1,N);
r3 = sqrt(a3.*b3).*exp(2.*j.*pi.*rand(1,N));
n3 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

a4 = gamrnd(m4,omega_av/m4,1,N);
b4 = gamrnd(k4,1/k4,1,N);
r4 = sqrt(a4.*b4).*exp(2.*j.*pi.*rand(1,N));
n4 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

a5 = gamrnd(m5,omega_av/m5,1,N);
b5 = gamrnd(k5,1/k5,1,N);
r5 = sqrt(a5.*b5).*exp(2.*j.*pi.*rand(1,N));
n5 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

%%%%%%%%%%%%%%%%%%%%%%%% 3rd CASE%%%%%%%%%%%%%
a6 = gamrnd(m6,omega_av/m6,1,N);
b6 = gamrnd(k6,1/k6,1,N);
r6 = sqrt(a6.*b6).*exp(2.*j.*pi.*rand(1,N));
n6 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

a7 = gamrnd(m7,omega_av/m7,1,N);
b7 = gamrnd(k7,1/k7,1,N);
r7 = sqrt(a7.*b7).*exp(2.*j.*pi.*rand(1,N));
n7 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

a8 = gamrnd(m8,omega_av/m8,1,N);
b8 = gamrnd(k8,1/k8,1,N);
r8 = sqrt(a8.*b8).*exp(2.*j.*pi.*rand(1,N));
n8 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

%%%%%%%%%%%%%%%% 4th CASE %%%%%%%%%%%%%%%
a9 = gamrnd(m9,omega_av/m9,1,N);
b9 = gamrnd(k9,1/k9,1,N);
r9 = sqrt(a9.*b9).*exp(2.*j.*pi.*rand(1,N));
n9 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

a10 = gamrnd(m10,omega_av/m10,1,N);
b10 = gamrnd(k10,1/k10,1,N);
r10 = sqrt(a10.*b10).*exp(2.*j.*pi.*rand(1,N));
n10 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

a11 = gamrnd(m11,omega_av/m11,1,N);
b11 = gamrnd(k11,1/k11,1,N);
r11 = sqrt(a11.*b11).*exp(2.*j.*pi.*rand(1,N));
n11 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));
```

```
        %%%%% Amplify & Forward %%%%%%% 1st CASE %%%%%%%%
        A = sqrt(1./((a1.*b1) + (1./(SNR))));
        y1 = conj(r).*((r.*u + n))./(r.*conj(r));
        y2 = conj(r1).*conj(r2).*(r2.*(A.*(r1.*u + n1) +
n2))./(r1.*r2.*conj(r1).*conj(r2));
        y = y1 + y2;

        %%%%% Amplify & Forward %%%%%%% 2nd CASE %%%%%%%%
        A2 = sqrt(1./((a4.*b4) + (1./(SNR))));
        y3 = conj(r3).*((r3.*u + n))./(r3.*conj(r3));
        y4 = conj(r4).*conj(r5).*(r5.*(A2.*(r4.*u + n4) +
n5))./(r4.*r5.*conj(r4).*conj(r5));
        y5 = y3 + y4;

        %%%%% Amplify & Forward %%%%%%% 3rd CASE %%%%%%%%
        A3 = sqrt(1./((a7.*b7) + (1./(SNR))));
        y6 = conj(r6).*((r6.*u + n))./(r6.*conj(r6));
        y7 = conj(r7).*conj(r8).*(r8.*(A3.*(r7.*u + n7) +
n8))./(r7.*r8.*conj(r7).*conj(r8));
        y8 = y6 + y7;

        %%%%% Amplify & Forward %%%%%%% 4th CASE %%%%%%%%
        A4 = sqrt(1./((a11.*b11) + (1./(SNR))));
        y9 = conj(r9).*((r9.*u + n))./(r9.*conj(r9));
        y10 = conj(r10).*conj(r11).*(r11.*(A4.*(r10.*u + n10) +
n11))./(r10.*r11.*conj(r10).*conj(r11));
        y11 = y9 + y10;


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION 1st CASE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % ------------------------------------------------------------------
--------------- %
    % finding the phase from [-pi to +pi]
    opPhase = angle(y);
    % unwrapping the phase i.e. phase less than 0 are
    % added 2pi
    opPhase(find(opPhase<0)) = opPhase(find(opPhase<0)) + 2*pi;

    % rounding the received phase to the closest constellation
    ipPhaseHat = 2*pi/M*round(opPhase/(2*pi/M)) ;
    % as there is phase ambiguity for phase = 0 and 2*pi,
    % changing all phases reported as 2*pi to 0.
    % this is to enable comparison with the transmitted phase
    ipPhaseHat(find(ipPhaseHat==2*pi)) = 0;
    ipDecHat = round(ipPhaseHat*M/(2*pi));

    % Decimal to Gray code conversion
    ipGrayHat = map(ipDecHat+1); % converting to decimal
    ipBinHat = dec2bin(ipGrayHat,kb) ; % decimal to binary

    % converting binary string to number
    ipBinHat = ipBinHat.';
    ipBinHat = ipBinHat(1:end).';
    ipBinHat = str2num(ipBinHat).' ;

    % counting errors
```

```
    nBitErr(ii) = size(find([ipBit- ipBinHat]),2); % couting the
number of errors

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION 2nd CASE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % ------------------------------------------------------------------
--------------- %
    % finding the phase from [-pi to +pi]
    opPhase2 = angle(y5);
    % unwrapping the phase i.e. phase less than 0 are
    % added 2pi
    opPhase2(find(opPhase2<0)) = opPhase2(find(opPhase2<0)) + 2*pi;

    % rounding the received phase to the closest constellation
    ipPhaseHat2 = 2*pi/M*round(opPhase2/(2*pi/M)) ;
    % as there is phase ambiguity for phase = 0 and 2*pi,
    % changing all phases reported as 2*pi to 0.
    % this is to enable comparison with the transmitted phase
    ipPhaseHat2(find(ipPhaseHat2==2*pi)) = 0;
    ipDecHat2 = round(ipPhaseHat2*M/(2*pi));

    % Decimal to Gray code conversion
    ipGrayHat2 = map(ipDecHat2+1); % converting to decimal
    ipBinHat2 = dec2bin(ipGrayHat2,kb) ; % decimal to binary

    % converting binary string to number
    ipBinHat2 = ipBinHat2.';
    ipBinHat2 = ipBinHat2(1:end).';
    ipBinHat2 = str2num(ipBinHat2).' ;

    % counting errors
    nBitErr2(ii) = size(find([ipBit- ipBinHat2]),2); % couting the
number of errors

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION 3rd CASE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % ------------------------------------------------------------------
--------------- %
    % finding the phase from [-pi to +pi]
    opPhase3 = angle(y8);
    % unwrapping the phase i.e. phase less than 0 are
    % added 2pi
    opPhase3(find(opPhase3<0)) = opPhase3(find(opPhase3<0)) + 2*pi;

    % rounding the received phase to the closest constellation
    ipPhaseHat3 = 2*pi/M*round(opPhase3/(2*pi/M)) ;
    % as there is phase ambiguity for phase = 0 and 2*pi,
    % changing all phases reported as 2*pi to 0.
    % this is to enable comparison with the transmitted phase
    ipPhaseHat3(find(ipPhaseHat3==2*pi)) = 0;
    ipDecHat3 = round(ipPhaseHat3*M/(2*pi));

    % Decimal to Gray code conversion
    ipGrayHat3 = map(ipDecHat3+1); % converting to decimal
    ipBinHat3 = dec2bin(ipGrayHat3,kb) ; % decimal to binary

    % converting binary string to number
    ipBinHat3 = ipBinHat3.';
```

```
    ipBinHat3 = ipBinHat3(1:end).';
    ipBinHat3 = str2num(ipBinHat3).' ;

    % counting errors
    nBitErr3(ii) = size(find([ipBit- ipBinHat3]),2); % couting the
number of errors

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION 4th CASE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % ---------------------------------------------------------------
--------------- %
    % finding the phase from [-pi to +pi]
    opPhase4 = angle(y11);
    % unwrapping the phase i.e. phase less than 0 are
    % added 2pi
    opPhase4(find(opPhase4<0)) = opPhase4(find(opPhase4<0)) + 2*pi;

    % rounding the received phase to the closest constellation
    ipPhaseHat4 = 2*pi/M*round(opPhase4/(2*pi/M)) ;
    % as there is phase ambiguity for phase = 0 and 2*pi,
    % changing all phases reported as 2*pi to 0.
    % this is to enable comparison with the transmitted phase
    ipPhaseHat4(find(ipPhaseHat4==2*pi)) = 0;
    ipDecHat4 = round(ipPhaseHat4*M/(2*pi));

    % Decimal to Gray code conversion
    ipGrayHat4 = map(ipDecHat4+1); % converting to decimal
    ipBinHat4 = dec2bin(ipGrayHat4,kb) ; % decimal to binary

    % converting binary string to number
    ipBinHat4 = ipBinHat4.';
    ipBinHat4 = ipBinHat4(1:end).';
    ipBinHat4 = str2num(ipBinHat4).' ;

    % counting errors
    nBitErr4(ii) = size(find([ipBit- ipBinHat4]),2); % couting the
number of errors

end
simBer = nBitErr/(N*kb);
simBer2 = nBitErr2/(N*kb);
simBer3 = nBitErr3/(N*kb);
simBer4 = nBitErr4/(N*kb);

semilogy(Eb_N0_dB,simBer,'r-',Eb_N0_dB,simBer2,'bx-',
Eb_N0_dB,simBer3,'g+-', Eb_N0_dB,simBer4,'yo-');
grid on;
axis([0 20 10^(-4) 1]);
legend ('m=1, m1=m2=2, k=1, k1=k2=3','m=4, m1=m2=6, k=2,
k1=k2=5','m=6, m1=m2=8, k=3, k1=k2=6','m=4, m1=m2=10, k=4,
k1=k2=10','Location','SouthWest');
xlabel('Eb/No, dB'); ylabel('BER');
title('Non Regenerative CSI Relay in Generalized-K fading');
```

- **3.4.1 Matlab script for BER of Fixed Gain Relay for BPSK in Nakagami-m**

```
clear;

SNR_dB = 0:2:20;
omega_av = 1;
m=1;
k=1;

m1=2;
k1=3;
m2=2;
k2=3;

N=10^4;

for i=1:length(SNR_dB)

    errors1=0;
    errors2=0;
    errors3=0;
    errors4=0;

    SNR = 10^(SNR_dB(i)/10);

        for j=1:N
            h = randint;
            u = 2*h-1;

            a = gamrnd(m,omega_av/m);
            b = 1;%gamrnd(k,1/k);
            r = sqrt(a*b);
            n = sqrt((omega_av/(2*SNR)))*randn;

            a1 = gamrnd(m1,omega_av/m1);
            b1 = 1;%gamrnd(k1,1/k1);
            r1 = sqrt(a1*b1);
            n1 = sqrt((omega_av/(2*SNR)))*randn;

            a2 = gamrnd(m2,omega_av/m2);
            b2 = 1;%gamrnd(k2,1/k2);
            r2 = sqrt(a2*b2);
            n2 = sqrt((omega_av/(2*SNR)))*randn;

            A = 0.5; %sqrt(1/((a1*b1)+(1/(SNR))));
            A2 = 1.5;
            A3 = 2.5;
            A4 = 3.5;

            y = r*(r*u + n) + (r2*r1)*(r2*(A*(r1*u + n1)) + n2);
            y2 = r*(r*u + n) + (r2*r1)*(r2*(A2*(r1*u + n1)) + n2);
            y3 = r*(r*u + n) + (r2*r1)*(r2*(A3*(r1*u + n1)) + n2);
            y4 = r*(r*u + n) + (r2*r1)*(r2*(A4*(r1*u + n1)) + n2);

            if y >= 0.0
```

```
            h_r = 1;
        elseif y < 0
            h_r = 0;
        end

        errors1 = errors1 + (h_r~=h);

        if y2 >= 0.0
            h_r2 = 1;
        elseif y2 < 0
            h_r2 = 0;
        end

        errors2 = errors2 + (h_r2~=h);

        if y3 >= 0.0
            h_r3 = 1;
        elseif y3 < 0
            h_r3 = 0;
        end

        errors3 = errors3 + (h_r3~=h);

        if y4 >= 0.0
            h_r4 = 1;
        elseif y4 < 0
            h_r4 = 0;
        end

        errors4 = errors4 + (h_r4~=h);
    end

ber(i) = errors1/N;
ber2(i) = errors2/N;
ber3(i) = errors3/N;
ber4(i) = errors4/N;

end

semilogy(SNR_dB, ber,'rx-',SNR_dB, ber2,'bx-',SNR_dB, ber3,'yx-
',SNR_dB, ber4,'gx-');
axis([0 14 10^-4 10^0])
grid on;
legend ('Gain=0.5','Gain=1.5','Gain=2.5','Gain=3.5');
xlabel('Eb/No, dB'); ylabel('SER');
title('Non Regenerative Relays with Fixed Gain in Nakagami-m
fading');
```

- **3.4.2 Matlab script for SER of Fixed Gain Relay for QPSK in Nakagami-m**

```
clear;

M = 4;
SNR_dB = 0:2:20;
omega_av = 1;
m = 1;
k = 1;

m1 = 2;
k1 = 3;
m2 = 2;
k2 = 3;

N = 1000;
loops = 1000;

for i=1:length(SNR_dB)

    errors_1 = 0;
    errors_2 = 0;
    errors_3 = 0;
    errors_4 = 0;

    SNR = 10^(SNR_dB(i)/10);

    for z=1:loops

        h = randint(1,N,M);
        u = pskmod(h,M);

        a = gamrnd(m,omega_av/m,1,N);
        b = 1;%gamrnd(k,1/k,1,N);
        r = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
        n = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a1 = gamrnd(m1,omega_av/m1,1,N);
        b1 = 1;%gamrnd(k1,1/k1,1,N);
        r1 = sqrt(a1.*b1).*exp(2.*j.*pi.*rand(1,N));
        n1 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a2 = gamrnd(m2,omega_av/m2,1,N);
        b2 = 1;%gamrnd(k2,1/k2,1,N);
        r2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
        n2 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        %Amplify & Forward
        A = 0.5;%sqrt(1./((a1.*b1) + (1./(SNR))));
        A2 = 1.5;
        A3 = 2.5;
        A4 = 3.5;

        y1 = conj(r).*((r.*u + n))./(r.*conj(r));
        y2 = conj(r1).*conj(r2).*(r2.*(A.*(r1.*u + n1) +
n2))./(r1.*r2.*conj(r1).*conj(r2));
```

```
        y3 = conj(r1).*conj(r2).*(r2.*(A2.*(r1.*u + n1) +
n2))./(r1.*r2.*conj(r1).*conj(r2));
        y4 = conj(r1).*conj(r2).*(r2.*(A3.*(r1.*u + n1) +
n2))./(r1.*r2.*conj(r1).*conj(r2));
        y5 = conj(r1).*conj(r2).*(r2.*(A4.*(r1.*u + n1) +
n2))./(r1.*r2.*conj(r1).*conj(r2));

        y = y1 + y2;
        y6 = y1 + y3;
        y7 = y1 + y4;
        y8 = y1 + y5;

        %Demodulation
        demodmsg = pskdemod(y,M);
        [nes_amp,res_amp] = symerr(h,demodmsg);

        %Find total erros
        errors_1 = errors_1 + nes_amp;

        %Demodulation2
        demodmsg2 = pskdemod(y6,M);
        [nes_amp2,res_amp2] = symerr(h,demodmsg2);

        %Find total erros
        errors_2 = errors_2 + nes_amp2;

        %Demodulation3
        demodmsg3 = pskdemod(y7,M);
        [nes_amp3,res_amp3] = symerr(h,demodmsg3);

        %Find total erros
        errors_3 = errors_3 + nes_amp3;

        %Demodulation4
        demodmsg4 = pskdemod(y8,M);
        [nes_amp4,res_amp4] = symerr(h,demodmsg4);

        %Find total erros
        errors_4 = errors_4 + nes_amp4;
    end

    ser_amp(i) = errors_1/(N*loops);
    ser_amp2(i) = errors_2/(N*loops);
    ser_amp3(i) = errors_3/(N*loops);
    ser_amp4(i) = errors_4/(N*loops);
end

figure;
semilogy(SNR_dB, ser_amp, 'bx-',SNR_dB, ser_amp2, 'rx-',SNR_dB,
ser_amp3, 'yx-',SNR_dB, ser_amp4, 'gx-');
grid on;
legend ('Gain=0.5','Gain=1.5','Gain=2.5','Gain=3.5');
xlabel('Es/No, dB'); ylabel('SER');
title('Non Regenerative Relays with Fixed Gain in Nakagami-m
fading');
```

- **3.4.3 Matlab script for BER of Fixed Gain Relay for QPSK in Nakagami-m**

```
    % Bit Error Rate for QPSK modulation using Gray modulation
mapping

    clear
    N = 10^4; % number of symbols
    M = 4;   % constellation size
    kb = log2(M); % bits per symbol

    omega_av = 1;
    m = 1;
    k = 1;
    m1 = 2;
    k1 = 3;
    m2 = 2;
    k2 = 3;

    thetaMpsk = [0:M-1]*2*pi/M; % reference phase values

    Eb_N0_dB  = [0:2:20]; % multiple Es/N0 values
    Es_N0_dB  = Eb_N0_dB + 10*log10(kb);

    % Mapping for binary <--> Gray code conversion
    ref = [0:M-1];
    map = bitxor(ref,floor(ref/2));
    [tt ind] = sort(map);

    ipPhaseHat = zeros(1,N);
    for ii = 1:length(Eb_N0_dB)


        % symbol generation
        % -----------------
        ipBit = rand(1,N*kb,1)>0.5; % random 1's and 0's
        bin2DecMatrix = ones(N,1)*(2.^[(kb-1):-1:0]) ; % conversion
from binary to decimal
        ipBitReshape =  reshape(ipBit,kb,N).'; % grouping to N
symbols having kb bits each
        ipGray = [sum(ipBitReshape.*bin2DecMatrix,2)].'; % decimal to
binary

        % Gray coded constellation mapping
        ipDec = ind(ipGray+1)-1; % bit group to constellation point
        ipPhase = ipDec*2*pi/M; % conversion to phase
        ip = exp(j*ipPhase);  % modulation
        s = ip;

        u = s + 10^(-Es_N0_dB(ii)/20);
        SNR = 10^(Eb_N0_dB(ii)/10);

        a = gamrnd(m,omega_av/m,1,N);
        b = 1;%gamrnd(k,1/k,1,N);
        r = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
        n = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a1 = gamrnd(m1,omega_av/m1,1,N);
```

```
        b1 = 1;%gamrnd(k,1/k,1,N);
        r1 = sqrt(a1.*b1).*exp(2.*j.*pi.*rand(1,N));
        n1 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a2 = gamrnd(m2,omega_av/m2,1,N);
        b2 = 1;%gamrnd(k,1/k);
        r2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
        n2 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        A1 = 0.5; %sqrt(1./((a1.*b1)+(1./(SNR))));
        A2 = 1.5;
        A3 = 2.5;
        A4 = 3.5;

        y1 = conj(r).*((r.*u + n))./(r.*conj(r));
        y2 = conj(r1).*conj(r2).*(r2.*(A1.*(r1.*u + n1)+
n2))./(r1.*r2.*conj(r1).*conj(r2));
        y3 = conj(r1).*conj(r2).*(r2.*(A2.*(r1.*u + n1)+
n2))./(r1.*r2.*conj(r1).*conj(r2));
        y4 = conj(r1).*conj(r2).*(r2.*(A3.*(r1.*u + n1)+
n2))./(r1.*r2.*conj(r1).*conj(r2));
        y5 = conj(r1).*conj(r2).*(r2.*(A4.*(r1.*u + n1)+
n2))./(r1.*r2.*conj(r1).*conj(r2));

        y = y1+y2;
        y6 = y1+y3;
        y7 = y1+y4;
        y8 = y1+y5;


        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % ------------------------------------------------------------
------------------- %
        % finding the phase from [-pi to +pi]
        opPhase = angle(y);
        % unwrapping the phase i.e. phase less than 0 are
        % added 2pi
        opPhase(find(opPhase<0)) = opPhase(find(opPhase<0)) + 2*pi;

        % rounding the received phase to the closest constellation
        ipPhaseHat = 2*pi/M*round(opPhase/(2*pi/M)) ;
        % as there is phase ambiguity for phase = 0 and 2*pi,
        % changing all phases reported as 2*pi to 0.
        % this is to enable comparison with the transmitted phase
        ipPhaseHat(find(ipPhaseHat==2*pi)) = 0;
        ipDecHat = round(ipPhaseHat*M/(2*pi));

        % Decimal to Gray code conversion
        ipGrayHat = map(ipDecHat+1); % converting to decimal
        ipBinHat = dec2bin(ipGrayHat,kb) ; % decimal to binary

        % converting binary string to number
        ipBinHat = ipBinHat.';
        ipBinHat = ipBinHat(1:end).';
        ipBinHat = str2num(ipBinHat).' ;

        % counting errors
```

```
        nBitErr(ii) = size(find([ipBit- ipBinHat]),2); % couting the
number of errors

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION
2%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % ---------------------------------------------------------------
------------------- %
        % finding the phase from [-pi to +pi]
        opPhase2 = angle(y6);
        % unwrapping the phase i.e. phase less than 0 are
        % added 2pi
        opPhase2(find(opPhase2<0)) = opPhase2(find(opPhase2<0)) +
2*pi;

        % rounding the received phase to the closest constellation
        ipPhaseHat2 = 2*pi/M*round(opPhase2/(2*pi/M)) ;
        % as there is phase ambiguity for phase = 0 and 2*pi,
        % changing all phases reported as 2*pi to 0.
        % this is to enable comparison with the transmitted phase
        ipPhaseHat2(find(ipPhaseHat2==2*pi)) = 0;
        ipDecHat2 = round(ipPhaseHat2*M/(2*pi));

        % Decimal to Gray code conversion
        ipGrayHat2 = map(ipDecHat2+1); % converting to decimal
        ipBinHat2 = dec2bin(ipGrayHat2,kb) ; % decimal to binary

        % converting binary string to number
        ipBinHat2 = ipBinHat2.';
        ipBinHat2 = ipBinHat2(1:end).';
        ipBinHat2 = str2num(ipBinHat2).' ;

        % counting errors
        nBitErr2(ii) = size(find([ipBit- ipBinHat2]),2); % couting
the number of errors

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION 3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % ---------------------------------------------------------------
------------------- %
        % finding the phase from [-pi to +pi]
        opPhase3 = angle(y7);
        % unwrapping the phase i.e. phase less than 0 are
        % added 2pi
        opPhase3(find(opPhase3<0)) = opPhase3(find(opPhase3<0)) +
2*pi;

        % rounding the received phase to the closest constellation
        ipPhaseHat3 = 2*pi/M*round(opPhase3/(2*pi/M)) ;
        % as there is phase ambiguity for phase = 0 and 2*pi,
        % changing all phases reported as 2*pi to 0.
        % this is to enable comparison with the transmitted phase
        ipPhaseHat3(find(ipPhaseHat3==2*pi)) = 0;
        ipDecHat3 = round(ipPhaseHat3*M/(2*pi));

        % Decimal to Gray code conversion
        ipGrayHat3 = map(ipDecHat3+1); % converting to decimal
        ipBinHat3 = dec2bin(ipGrayHat3,kb) ; % decimal to binary
```

```
        % converting binary string to number
        ipBinHat3 = ipBinHat3.';
        ipBinHat3 = ipBinHat3(1:end).';
        ipBinHat3 = str2num(ipBinHat3).' ;

        % counting errors
        nBitErr3(ii) = size(find([ipBit- ipBinHat3]),2); % couting
the number of errors

       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION 3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % ---------------------------------------------------------------
------------------- %
        % finding the phase from [-pi to +pi]
        opPhase4 = angle(y8);
        % unwrapping the phase i.e. phase less than 0 are
        % added 2pi
        opPhase4(find(opPhase4<0)) = opPhase4(find(opPhase4<0)) +
2*pi;

        % rounding the received phase to the closest constellation
        ipPhaseHat4 = 2*pi/M*round(opPhase4/(2*pi/M)) ;
        % as there is phase ambiguity for phase = 0 and 2*pi,
        % changing all phases reported as 2*pi to 0.
        % this is to enable comparison with the transmitted phase
        ipPhaseHat4(find(ipPhaseHat4==2*pi)) = 0;
        ipDecHat4 = round(ipPhaseHat4*M/(2*pi));

        % Decimal to Gray code conversion
        ipGrayHat4 = map(ipDecHat4+1); % converting to decimal
        ipBinHat4 = dec2bin(ipGrayHat4,kb) ; % decimal to binary

        % converting binary string to number
        ipBinHat4 = ipBinHat4.';
        ipBinHat4 = ipBinHat4(1:end).';
        ipBinHat4 = str2num(ipBinHat4).' ;

        % counting errors
        nBitErr4(ii) = size(find([ipBit- ipBinHat4]),2); % couting
the number of errors

    end
    simBer = nBitErr/(N*kb);
    simBer2 = nBitErr2/(N*kb);
    simBer3 = nBitErr3/(N*kb);
    simBer4 = nBitErr4/(N*kb);

    semilogy(Eb_N0_dB,simBer,'rx-',Eb_N0_dB,simBer2,'bx-
',Eb_N0_dB,simBer3,'gx-',Eb_N0_dB,simBer4,'yx-');
    axis([0 20 10^-4 1])
    grid on;
    legend('Gain=0.5','Gain=1.5','Gain=2.5','Gain=3.5');
    xlabel('Eb/No, dB')
    ylabel('BER')
    title('Non Regenerative Relays with Fixed Gain in Nakagami-m
fading')
```

- **3.4.4 Matlab script for BER of Fixed Gain Relay for BPSK in Generalized-K**

```
clear;

SNR_dB = 0:2:20;
omega_av = 1;
m=1;
k=1;

m1=2;
k1=3;
m2=2;
k2=3;

N=10^4;

for i=1:length(SNR_dB)

    errors1=0;
    errors2=0;
    errors3=0;
    errors4=0;

    SNR = 10^(SNR_dB(i)/10);

        for j=1:N
            h = randint;
            u = 2*h-1;

            a = gamrnd(m,omega_av/m);
            b = gamrnd(k,1/k);
            r = sqrt(a*b);
            n = sqrt((omega_av/(2*SNR)))*randn;

            a1 = gamrnd(m1,omega_av/m1);
            b1 = gamrnd(k1,1/k1);
            r1 = sqrt(a1*b1);
            n1 = sqrt((omega_av/(2*SNR)))*randn;

            a2 = gamrnd(m2,omega_av/m2);
            b2 = gamrnd(k2,1/k2);
            r2 = sqrt(a2*b2);
            n2 = sqrt((omega_av/(2*SNR)))*randn;

            A = 0.5; %sqrt(1/((a1*b1)+(1/(SNR))));
            A2 = 1.5;
            A3 = 2.5;
            A4 = 3.5;

            y = r*(r*u + n) + (r2*r1)*(r2*(A*(r1*u + n1)) + n2);
            y2 = r*(r*u + n) + (r2*r1)*(r2*(A2*(r1*u + n1)) + n2);
            y3 = r*(r*u + n) + (r2*r1)*(r2*(A3*(r1*u + n1)) + n2);
            y4 = r*(r*u + n) + (r2*r1)*(r2*(A4*(r1*u + n1)) + n2);

            if y >= 0.0
                h_r = 1;
```

```
            elseif y < 0
                h_r = 0;
            end

            errors1 = errors1 + (h_r~=h);

            if y2 >= 0.0
                h_r2 = 1;
            elseif y2 < 0
                h_r2 = 0;
            end

            errors2 = errors2 + (h_r2~=h);

            if y3 >= 0.0
                h_r3 = 1;
            elseif y3 < 0
                h_r3 = 0;
            end

            errors3 = errors3 + (h_r3~=h);

            if y4 >= 0.0
                h_r4 = 1;
            elseif y4 < 0
                h_r4 = 0;
            end

            errors4 = errors4 + (h_r4~=h);
        end

ber(i) = errors1/N;
ber2(i) = errors2/N;
ber3(i) = errors3/N;
ber4(i) = errors4/N;

end

semilogy(SNR_dB, ber,'rx-',SNR_dB, ber2,'bx-',SNR_dB, ber3,'yx-
',SNR_dB, ber4,'gx-');
axis([0 16 10^-4 10^0])
grid on;
legend ('Gain=0.5','Gain=1.5','Gain=2.5','Gain=3.5');
xlabel('Eb/No, dB'); ylabel('BER');
title('Non Regenerative Relays with Fixed Gain in Generalized-K
fading');
```

- **3.4.5 Matlab script for SER of Fixed Gain Relay for QPSK in Generalized-K**

```
clear;

M = 4;
SNR_dB = 0:2:20;
omega_av = 1;
m = 1;
k = 1;

m1 = 2;
k1 = 3;
m2 = 2;
k2 = 3;

N = 1000;
loops = 1000;

for i=1:length(SNR_dB)

    errors_1 = 0;
    errors_2 = 0;
    errors_3 = 0;
    errors_4 = 0;

    SNR = 10^(SNR_dB(i)/10);

    for z=1:loops

        h = randint(1,N,M);
        u = pskmod(h,M);

        a = gamrnd(m,omega_av/m,1,N);
        b = gamrnd(k,1/k,1,N);
        r = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
        n = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a1 = gamrnd(m1,omega_av/m1,1,N);
        b1 = gamrnd(k1,1/k1,1,N);
        r1 = sqrt(a1.*b1).*exp(2.*j.*pi.*rand(1,N));
        n1 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        a2 = gamrnd(m2,omega_av/m2,1,N);
        b2 = gamrnd(k2,1/k2,1,N);
        r2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
        n2 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

        %Amplify & Forward
        A = 0.5;%sqrt(1./((a1.*b1) + (1./(SNR))));
        A2 = 1.5;
        A3 = 2.5;
        A4 = 3.5;

        y1 = conj(r).*((r.*u + n))./(r.*conj(r));
        y2 = conj(r1).*conj(r2).*(r2.*(A.*(r1.*u + n1) +
n2))./(r1.*r2.*conj(r1).*conj(r2));
```

```
        y3 = conj(r1).*conj(r2).*(r2.*(A2.*(r1.*u + n1) +
n2))./(r1.*r2.*conj(r1).*conj(r2));
        y4 = conj(r1).*conj(r2).*(r2.*(A3.*(r1.*u + n1) +
n2))./(r1.*r2.*conj(r1).*conj(r2));
        y5 = conj(r1).*conj(r2).*(r2.*(A4.*(r1.*u + n1) +
n2))./(r1.*r2.*conj(r1).*conj(r2));

        y = y1 + y2;
        y6 = y1 + y3;
        y7 = y1 + y4;
        y8 = y1 + y5;

        %Demodulation
        demodmsg = pskdemod(y,M);
        [nes_amp,res_amp] = symerr(h,demodmsg);

        %Find total erros
        errors_1 = errors_1 + nes_amp;

        %Demodulation2
        demodmsg2 = pskdemod(y6,M);
        [nes_amp2,res_amp2] = symerr(h,demodmsg2);

        %Find total erros
        errors_2 = errors_2 + nes_amp2;

        %Demodulation3
        demodmsg3 = pskdemod(y7,M);
        [nes_amp3,res_amp3] = symerr(h,demodmsg3);

        %Find total erros
        errors_3 = errors_3 + nes_amp3;

        %Demodulation4
        demodmsg4 = pskdemod(y8,M);
        [nes_amp4,res_amp4] = symerr(h,demodmsg4);

        %Find total erros
        errors_4 = errors_4 + nes_amp4;
    end

    ser_amp(i) = errors_1/(N*loops);
    ser_amp2(i) = errors_2/(N*loops);
    ser_amp3(i) = errors_3/(N*loops);
    ser_amp4(i) = errors_4/(N*loops);
end

figure;
semilogy(SNR_dB, ser_amp, 'bx-',SNR_dB, ser_amp2, 'rx-',SNR_dB,
ser_amp3, 'yx-',SNR_dB, ser_amp4, 'gx-');
grid on;
legend ('Gain=0.5','Gain=1.5','Gain=2.5','Gain=3.5');
xlabel('Es/No, dB'); ylabel('SER');
title('Non Regenerative Relays with Fixed Gain in Generalized-K
fading');
```

- **3.4.6 Matlab script for BER of Fixed Gain Relay for QPSK in Generalized-K**

```
% Bit Error Rate for QPSK modulation using Gray modulation mapping

clear
N = 10^4; % number of symbols
M = 4;   % constellation size
kb = log2(M); % bits per symbol

omega_av = 1;
m = 1;
k = 1;
m1 = 2;
k1 = 3;
m2 = 2;
k2 = 3;

thetaMpsk = [0:M-1]*2*pi/M; % reference phase values

Eb_N0_dB  = [0:2:20]; % multiple Es/N0 values
Es_N0_dB  = Eb_N0_dB + 10*log10(kb);

% Mapping for binary <--> Gray code conversion
ref = [0:M-1];
map = bitxor(ref,floor(ref/2));
[tt ind] = sort(map);

ipPhaseHat = zeros(1,N);
for ii = 1:length(Eb_N0_dB)


    % symbol generation
    % -----------------
    ipBit = rand(1,N*kb,1)>0.5; % random 1's and 0's
    bin2DecMatrix = ones(N,1)*(2.^[(kb-1):-1:0]) ; % conversion from
binary to decimal
    ipBitReshape =  reshape(ipBit,kb,N).'; % grouping to N symbols
having kb bits each
    ipGray = [sum(ipBitReshape.*bin2DecMatrix,2)].'; % decimal to
binary

    % Gray coded constellation mapping
    ipDec = ind(ipGray+1)-1; % bit group to constellation point
    ipPhase = ipDec*2*pi/M; % conversion to phase
    ip = exp(j*ipPhase);  % modulation
    s = ip;

    u = s + 10^(-Es_N0_dB(ii)/20);
    SNR = 10^(Eb_N0_dB(ii)/10);

    a = gamrnd(m,omega_av/m,1,N);
    b = gamrnd(k,1/k,1,N);
    r = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
    n = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

    a1 = gamrnd(m1,omega_av/m1,1,N);
    b1 = gamrnd(k,1/k,1,N);
```

```
    r1 = sqrt(a1.*b1).*exp(2.*j.*pi.*rand(1,N));
    n1 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

    a2 = gamrnd(m2,omega_av/m2,1,N);
    b2 = gamrnd(k,1/k);
    r2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
    n2 = sqrt((omega_av/(2*SNR))).*(randn(1,N) + j*randn(1,N));

    A1 = 0.5; %sqrt(1./((a1.*b1)+(1./(SNR))));
    A2 = 1.5;
    A3 = 2.5;
    A4 = 3.5;

    y1 = conj(r).*((r.*u + n))./(r.*conj(r));
    y2 = conj(r1).*conj(r2).*(r2.*(A1.*(r1.*u + n1)+
n2))./(r1.*r2.*conj(r1).*conj(r2));
    y3 = conj(r1).*conj(r2).*(r2.*(A2.*(r1.*u + n1)+
n2))./(r1.*r2.*conj(r1).*conj(r2));
    y4 = conj(r1).*conj(r2).*(r2.*(A3.*(r1.*u + n1)+
n2))./(r1.*r2.*conj(r1).*conj(r2));
    y5 = conj(r1).*conj(r2).*(r2.*(A4.*(r1.*u + n1)+
n2))./(r1.*r2.*conj(r1).*conj(r2));

    y = y1+y2;
    y6 = y1+y3;
    y7 = y1+y4;
    y8 = y1+y5;


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % ---------------------------------------------------------------
--------------- %
    % finding the phase from [-pi to +pi]
    opPhase = angle(y);
    % unwrapping the phase i.e. phase less than 0 are
    % added 2pi
    opPhase(find(opPhase<0)) = opPhase(find(opPhase<0)) + 2*pi;

    % rounding the received phase to the closest constellation
    ipPhaseHat = 2*pi/M*round(opPhase/(2*pi/M)) ;
    % as there is phase ambiguity for phase = 0 and 2*pi,
    % changing all phases reported as 2*pi to 0.
    % this is to enable comparison with the transmitted phase
    ipPhaseHat(find(ipPhaseHat==2*pi)) = 0;
    ipDecHat = round(ipPhaseHat*M/(2*pi));

    % Decimal to Gray code conversion
    ipGrayHat = map(ipDecHat+1); % converting to decimal
    ipBinHat = dec2bin(ipGrayHat,kb) ; % decimal to binary

    % converting binary string to number
    ipBinHat = ipBinHat.';
    ipBinHat = ipBinHat(1:end).';
    ipBinHat = str2num(ipBinHat).' ;

    % counting errors
```

```
    nBitErr(ii) = size(find([ipBit- ipBinHat]),2); % couting the
number of errors

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION
2%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % ------------------------------------------------------------------
--------------- %
    % finding the phase from [-pi to +pi]
    opPhase2 = angle(y6);
    % unwrapping the phase i.e. phase less than 0 are
    % added 2pi
    opPhase2(find(opPhase2<0)) = opPhase2(find(opPhase2<0)) + 2*pi;

    % rounding the received phase to the closest constellation
    ipPhaseHat2 = 2*pi/M*round(opPhase2/(2*pi/M)) ;
    % as there is phase ambiguity for phase = 0 and 2*pi,
    % changing all phases reported as 2*pi to 0.
    % this is to enable comparison with the transmitted phase
    ipPhaseHat2(find(ipPhaseHat2==2*pi)) = 0;
    ipDecHat2 = round(ipPhaseHat2*M/(2*pi));

    % Decimal to Gray code conversion
    ipGrayHat2 = map(ipDecHat2+1); % converting to decimal
    ipBinHat2 = dec2bin(ipGrayHat2,kb) ; % decimal to binary

    % converting binary string to number
    ipBinHat2 = ipBinHat2.';
    ipBinHat2 = ipBinHat2(1:end).';
    ipBinHat2 = str2num(ipBinHat2).' ;

    % counting errors
    nBitErr2(ii) = size(find([ipBit- ipBinHat2]),2); % couting the
number of errors

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION 3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % ------------------------------------------------------------------
--------------- %
    % finding the phase from [-pi to +pi]
    opPhase3 = angle(y7);
    % unwrapping the phase i.e. phase less than 0 are
    % added 2pi
    opPhase3(find(opPhase3<0)) = opPhase3(find(opPhase3<0)) + 2*pi;

    % rounding the received phase to the closest constellation
    ipPhaseHat3 = 2*pi/M*round(opPhase3/(2*pi/M)) ;
    % as there is phase ambiguity for phase = 0 and 2*pi,
    % changing all phases reported as 2*pi to 0.
    % this is to enable comparison with the transmitted phase
    ipPhaseHat3(find(ipPhaseHat3==2*pi)) = 0;
    ipDecHat3 = round(ipPhaseHat3*M/(2*pi));

    % Decimal to Gray code conversion
    ipGrayHat3 = map(ipDecHat3+1); % converting to decimal
    ipBinHat3 = dec2bin(ipGrayHat3,kb) ; % decimal to binary

    % converting binary string to number
    ipBinHat3 = ipBinHat3.';
```

```
    ipBinHat3 = ipBinHat3(1:end).';
    ipBinHat3 = str2num(ipBinHat3).' ;

    % counting errors
    nBitErr3(ii) = size(find([ipBit- ipBinHat3]),2); % couting the
number of errors

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULATION 3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % ---------------------------------------------------------------
--------------- %
    % finding the phase from [-pi to +pi]
    opPhase4 = angle(y8);
    % unwrapping the phase i.e. phase less than 0 are
    % added 2pi
    opPhase4(find(opPhase4<0)) = opPhase4(find(opPhase4<0)) + 2*pi;

    % rounding the received phase to the closest constellation
    ipPhaseHat4 = 2*pi/M*round(opPhase4/(2*pi/M)) ;
    % as there is phase ambiguity for phase = 0 and 2*pi,
    % changing all phases reported as 2*pi to 0.
    % this is to enable comparison with the transmitted phase
    ipPhaseHat4(find(ipPhaseHat4==2*pi)) = 0;
    ipDecHat4 = round(ipPhaseHat4*M/(2*pi));

    % Decimal to Gray code conversion
    ipGrayHat4 = map(ipDecHat4+1); % converting to decimal
    ipBinHat4 = dec2bin(ipGrayHat4,kb) ; % decimal to binary

    % converting binary string to number
    ipBinHat4 = ipBinHat4.';
    ipBinHat4 = ipBinHat4(1:end).';
    ipBinHat4 = str2num(ipBinHat4).' ;

    % counting errors
    nBitErr4(ii) = size(find([ipBit- ipBinHat4]),2); % couting the
number of errors

end
simBer = nBitErr/(N*kb);
simBer2 = nBitErr2/(N*kb);
simBer3 = nBitErr3/(N*kb);
simBer4 = nBitErr4/(N*kb);

semilogy(Eb_N0_dB,simBer,'rx-',Eb_N0_dB,simBer2,'bx-
',Eb_N0_dB,simBer3,'gx-',Eb_N0_dB,simBer4,'yx-');
axis([0 20 10^-3 1])
grid on;
legend('Gain=0.5','Gain=1.5','Gain=2.5','Gain=3.5');
xlabel('Eb/No, dB')
ylabel('BER')
title('Non Regenerative Relays with Fixed Gain in Generalized-K
fading')
```

- **3.5.1. Matlab script for SER of Decode and Forward Relay for QPSK in Generalized-K**

```
clear;
global g m m1 x L
M = 4;
g=(sin(pi/M))^2;
SNR_dB = 0:2:30; %SNR_dB = 0:5:40;
omega_av = 1;

L=1;
m = 1;
k = 10.5;

m1 = 1;
k1 = 10.5;

m2 = 1;
k2 = 10.5;

N = 500;
loops = 1000;

for i=1:length(SNR_dB)

        errors=0;
        errors1=0;
        SNR = 10^(SNR_dB(i)/10);
        x=SNR;
        for z=1:loops

            s = randint(1,N,M); % Random message
            u = pskmod(s,M,pi/M);     % Modulate using QPSK.

                %a = gamrnd(m,omega_av/m, [1,N]);
                %b = gamrnd(k,1/k, [1,N]);
                %h = sqrt(a.*b).*exp(2.*j.*pi.*rand(1,N));
                %n =
sqrt((omega_av./(2*SNR))).*(randn(1,N)+j*randn(1,N));

                a1 = gamrnd(m1,omega_av/m1, [1,N]);
                b1 = gamrnd(k1,1/k1, [1,N]);
                h1 = sqrt(a1.*b1).*exp(2.*j.*pi.*rand(1,N));
                n1 =
sqrt((omega_av./(2*SNR))).*(randn(1,N)+j*randn(1,N));

                a2 = gamrnd(m2,omega_av/m2, [1,N]);
                b2 = gamrnd(k2,1/k2, [1,N]);
                h2 = sqrt(a2.*b2).*exp(2.*j.*pi.*rand(1,N));
                n2 =
sqrt((omega_av./(2*SNR))).*(randn(1,N)+j*randn(1,N));

                %r0 = (h.*u + n);
                r1 = (h1.*u + n1);

                y1 =  conj(h1).*r1./(conj(h1).*h1);
```

```
                s1 = pskdemod(y1,M,pi/M);
                u1 = pskmod(s1,M, pi/M);
                [nes1,res1] = symerr(s,s1);
                errors1 = errors1 + nes1;

                r2 = h2.*u1 + n2;

                %y0 = conj(h).*r0./(conj(h).*h);

                y2 =  conj(h2).*r2./(conj(h2).*h2);

                %y = (y2+y0)/2;
                %DEMODULATION using only the relay link
                s2 = pskdemod(y2,M,pi/M);
                [nes,res] = symerr(s,s2);

                % Find total errors from previous loops
                errors = errors + nes;

        end
ser1(i) = errors1/(N*loops);
ser(i) = errors/(N*loops);
ser_th(i) = (1/pi)*quadl(@mgf, 0, pi-pi/M, 10^(-6));
end

semilogy(SNR_dB,ser,'b-x')%,SNR_dB, ser_th, 'k-');
title('Decode & Forward Relay (QPSK)');
%legend('Simulation', 'Direct Rayleigh link')
xlabel('SNR (dB)'); ylabel('Symbol Error Rates');
%axis([0 40 10^(-6) 10^0]);
grid on;
```

- **3.5.2. Matlab script for 16QAM comparison in Generalized-K**

```
%close all
clear all
%clc
SNRdB=0:2:30;

m1=1;
k1=10.5;

m2=1;
k2=10.5;

SNR=10.^(SNRdB/10);
N=1000;
itr=1000;
for step=1:itr

for kk= 1:1:length(SNRdB)
```

```
    h1=1;
    h2=1;

    h1=h1.*SNR(kk);
    h2=h2.*SNR(kk);

    R1m = gamrnd(m1,h1/m1, 1, N);
    R1k = gamrnd(k1,1/k1, 1, N);
    R1 = R1m.*R1k;

    R2m = gamrnd(m2,h2/m2, 1, N);
    R2k = gamrnd(k2,1/k2, 1, N);
    R2 = R2m.*R2k;

    eqv = 1./((1+1./R1).*(1+1./R2)-1);
    eqv_min = min(R1, R2);

     M=16;
     b=3/(M-1);
     EXACT_qam1=(4*(sqrt(M)-1)/sqrt(M))*qfunc(sqrt(b*eqv)) -
(4*(sqrt(M)-1).^2/M).*(qfunc(sqrt(b*eqv))).^2;
     Min_qam1=(4*(sqrt(M)-1)/sqrt(M))*qfunc(sqrt(b*eqv_min)) -
(4*(sqrt(M)-1).^2/M).*(qfunc(sqrt(b*eqv_min))).^2;
     ExactQAM1_N2(step,kk)=sum(EXACT_qam1)/N;
     inQAM1_N2(step,kk)=sum(Min_qam1)/N;

end
end

ExactQAM1_N2=sum(ExactQAM1_N2)/itr
MinQAM1_N2=sum(MinQAM1_N2)/itr

figure(2)
semilogy(SNRdB,ExactQAM1_N2,'-k',SNRdB,MinQAM1_N2,'-b')
legend('Amplify and Forward','Decode and Forward')
xlabel('SNR (dB)'); ylabel('Symbol Error Rates')
title('Analytical end-to-end SNR')
grid on
```

- **3.5.3. Matlab script for QPSK comparison in Generalized-K**

```
%close all
clear all
%clc
SNRdB=0:2:30;

m1=1;
k1=10.5;

m2=1;
k2=10.5;

SNR=10.^(SNRdB/10);
```

```
N=1000;
itr=1000;
for step=1:itr

for kk= 1:1:length(SNRdB)

    h1=1;
    h2=1;

    h1=h1.*SNR(kk);
    h2=h2.*SNR(kk);

    R1m = gamrnd(m1,h1/m1, 1, N);
    R1k = gamrnd(k1,1/k1, 1, N);
    R1 = R1m.*R1k;

    R2m = gamrnd(m2,h2/m2, 1, N);
    R2k = gamrnd(k2,1/k2, 1, N);
    R2 = R2m.*R2k;

    eqv = 1./((1+1./R1).*(1+1./R2)-1);
    eqv_min = min(R1, R2);

    M=4;
    EXACT_qpsk1=2.*qfunc(sqrt(eqv))-(qfunc(sqrt(eqv))).^2;
    Min_qpsk1 = 2.*qfunc(sqrt(eqv_min))-(qfunc(sqrt(eqv_min))).^2;
    ExactQPSK1_N2(step,kk)=sum(EXACT_qpsk1)/N;
    MinQPSK1_N2(step,kk)=sum(Min_qpsk1)/N;

end
end

ExactQPSK1_N2=sum(ExactQPSK1_N2)/itr
MinQPSK1_N2=sum(MinQPSK1_N2)/itr

figure(1)
semilogy(SNRdB,ExactQPSK1_N2,'-k',SNRdB,MinQPSK1_N2,'-r')
legend('Amplify and Forward','Decode and Forward')
xlabel('SNR (dB)'); ylabel('Symbol Error Rates')
title('Analytical end-to-end SNR')
grid on
```