Linkedin
Soundcloud
GitHub
Academic CV
CV

A glimpse to my mind

Ilari Angervuori

Last major update 23.05.2022

Contents

1 About me

I am a Finnish mathematician interested in engineering. Read about my life, or jump straight to my professional records in the subsections below. Also, check out my blog.

I graduated in 2009 from Munkkiniemi high school. Mathematics was a subject I had naturally thrived in – so, after some bumps and turns, I found myself at the University of Helsinki studying mathematics. And yeah, indeed, I love mathematics – I love the apparent universality of it. This subject is without a doubt debatable, but, at least in some sense, I like to think that mathematical truths are universal in the truest sense of the word; they are eternal, they are the same everywhere, regardless of the physical universe we live in. Alien in another galaxy will end up in the same mathematical truths we do. Alien in another universe will end up in the same mathematical truths we do. Mathematics has the power to explain what we see in our everyday life. Mathematics is not only natural science but a form of art and poetry. Mathematics is music – music is mathematics.

While studying mathematics, physics and computer science, I took some courses on economics. That inspired me to write my bachelor's thesis on optimal control theory. I worked on the problem of how increasing public investments affects the GDP. I did not find any breakthrough, but it was an intriguing subject.

I proceeded with my graduate studies studying applied mathematics. I studied subjects like partial differential equations, functional analysis, dynamical systems, and – the University of Helsinki's proudness – complex analysis. (My thesis advisor said that, in a moral sense, you cannot graduate from the University of Helsinki without taking some courses on Complex analysis, because a lot of the discipline has been developed in the university.) In addition, as a more "practical" subject, I studied some inverse problems. Summa summarum, I studied a wide range of fields in mathematics.

During my graduate studies, I spent half a year in Utrecht, Netherlands, studying more applied analysis of varying subjects (searching periodic orbits in the Lorentz attractor as an example of a course – that I failed). At Utrecht University, I got the inspiration for a subject for my future master's thesis; the Finite Element Method. After I got back to Helsinki from the exchange, I had a chance to study more about the finite element method in Aalto University's courses. (Aalto University is a consortium of the Helsinki University of Technology, the Helsinki School of Economics and the University of Art and Design Helsinki.) While writing my thesis I also taught basic mathematics courses at the University of Helsinki and gained valuable experience in the pedagogical area.

In the binge of graduation, I started to look for future opportunities. I looked for coding jobs in Helsinki and Tallinn, jobs for mathematicians in the mapping industry, continuing at some universities to pursue a PhD etc. I am glad I had the chance to use my creativity and continue in Aalto University's Department of Signal Processing and Acoustics to research low earth orbit satellite communications. The research methodology was from a stochastic geometry perspective, that was well aligned with my mathematical background.

My professional ambitions are in improving the lives of people globally. Communications play an essential in the picture. (But contain some challenging problems also, as we have seen with the social media.) Through effective communication, we can share knowledge, control resources, discuss issues etc. – however, globally, the communication infrastructure is still not nearly complete. My interests contain, but are not limited to, communications, particularly wireless networks and signal processing.[4] My dream is to share my knowledge in the process toward a free and honest world. (Pardon me for the cliches.)

Figure 1: Me in my beloved home town Helsinki in the middle of the summer (well, technically I am in Espoo, Otaniemi – and it is a snowy and shiny day in February, frozen sea seen in the background)

I am keeping up a monthly blog that you can find in the blog posts section. It handles stuff encountered in my daily professional life (and I keep the right to write about other subjects). I hope you will find it interesting.

Btw, In the free time I love music, literature, long walks listening to podcasts, coffee and beer.

1.1 Education

- 2016 Bachelor of Science University of Helsinki

- 2018 Master of Philosophy, Department of Applied Analysis, University of Helsinki, grade: 4/5

1.2 Publications

- Ilari Angervuori and Risto Wichman, "A Closed-Form Approximation of the SIR Distribution in a LEO Uplink Channel", GLOBECOM 2022, Rio de Janeiro, Brazil, submitted for publication.

- Ilari Angervuori, Niloofar Okati, Taneli Riihonen, Risto Wichman, "Uplink Interference Characterization and Coverage Analysis of Modern LEO and VLEO Networks: A Stochastic Geometry Approach", IEEE Transactions on Communications, submitted for publication.

- Okati, N., Riihonen, T., Korpi, D., Angervuori, I. Wichman, R., Aug 2020, "Down-link Coverage and Rate Analysis of Low Earth Orbit Satellite Constellations Using Stochastic Geometry", IEEE Transactions on Communications. 68, 8, p. 5120-5134 9079921.

- A. Yastrebova et al., "Theoretical and Simulation-based Analysis of Terrestrial Interference to LEO Satellite Uplinks," GLOBECOM 2020 - 2020 IEEE Global Communications Conference, Taipei, Taiwan, 2020, pp. 1-6, doi: 10.1109/GLOBECOM42002.2020.9347980.

- Niloofar Okati, Islam Tanash, Taneli Riihonen, Ilari Angervuori, Wichman Risto, "Performance Evaluation of Low Earth Orbit Communication Satellites", Proceedings of XXXV Finnish URSI Convention on Radio Science, 2019

- Ilari Angervuori, Wichman Risto, Niloofar Okati, Taneli Riihonen, "On routing protocols in inter-satellite communications", Proceedings of XXXV Finnish URSI Convention on Radio Science, 2019

1.3 Thesis's and other projects

- Master's thesis "Elementtimenetelmä (The Finite Element Method)", https://helda.helsinki.fi/handle/101 Department of Applied Analysis, University of Helsinki, 2018, grade: eximia cum laude (5/5)

- Bachelor's thesis "Optimiohjausteoriasta ja sen sovelluksista (On Optimal Control Theory and Its Applications)", https://www.overleaf.com/project/563b9dec737da16f65bd24f2, University of Helsinki, 2016, grade: 4/5

- A university project for a poster session, "X-ray tomography with sparse data", https://www.dropbox.com/s/88pr2da24dil460/projektitosaulijamina.jpg?dl=0, 2015

2 Blog posts 2021

General thoughts on mathematics and engineering, practical instructions, artistic non-sense. The themes of this blog are much inspired by the challenges encountered in my professional life. The blog serves also as my personal notes.

2.1 January – Poisson process on a sphere

Poisson point process can be generalized to general manifolds. Particularly, Poisson process on a three dimensional sphere surface is useful. Nicely enough, Poisson process on a unit sphere is equivalent to process in a two dimensional

area $A = [-\pi, \pi] \times [-1, 1]$ through the area preserving mapping from A to geographical coordinates

$$(x, y) \mapsto (1, x, \sin^{-1}(y)).$$

Resulting process interpreted in geographical coordinates (r, θ, φ) is a Poisson point process on a sphere of radius r. Following codes returns a scatter plot of Poisson points on the unit sphere.

GNU Octave or Matlab:

```
%Plot random points on a unit sphere. Returns the points in a vector ref in cartesian coordi
function refc = poissononsphere(density)
  yMin = -1; yMax = 1;
  xMin = -pi; xMax = pi;

  xDelta = xMax - xMin; yDelta = yMax - yMin; %Rectangle dimensions
  numbPoints = poissrnd(density);    %Number of points in the area is a Poisson variable of
  x = xDelta*(rand(numbPoints,1)) + xMin;    %Pick points from uniform distribution
  y = yDelta*(rand(numbPoints,1)) + yMin;    %Map referencepoints to geographical coordinate
  ref = [x y]';

  refs = [x'; asin(y)'];%Map geographical coordinates to Cartesian coordinates on a unit cir
  r = 1;
  refc = [r*sin(refs(2,:)+pi/2).*cos(refs(1,:)+pi);...
          r*sin(refs(2,:)+pi/2).*sin(refs(1,:)+pi);...
          r*cos(refs(2,:)+pi/2)];

  figure(1)    %Plot
  [X, Y, Z] = sphere;
  surf(X,Y,Z,'EdgeColor','none','FaceColor','black');
  hold on
  scatter3(refc(1,:),refc(2,:),refc(3,:),10,...
           'MarkerFaceColor','yellow',...
           'MarkerEdgeColor','red');
  axis equal
end
```

Python:

```
import numpy as np
import scipy.stats
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d

#Rectangle dimension
xMin = -np.pi; xMax = np.pi;
yMin = -1; yMax = 1;
```

```
xDelta = xMax - xMin; yDelta = yMax - yMin; #rectangle dimensions

#Density parameter of the Poisson point process. Mean number of points on the sphere
lambda0=1000;

#Simulate Poisson point process

#Number of point in the area is a Poisson variable of intensity lambda0
numbPoints = scipy.stats.poisson( lambda0 ).rvs()
x = xDelta*scipy.stats.uniform.rvs(0,1,((numbPoints,1)))+xMin
y = yDelta*scipy.stats.uniform.rvs(0,1,((numbPoints,1)))+yMin

#Transform to geographical coordinates
x = x
y = np.arcsin(y)
#Plotting
fig = plt.figure()
ax = plt.axes(projection="3d")
ax.scatter(np.sin(y+np.pi/2)*np.cos(x+np.pi),np.sin(y+np.pi/2)*np.sin(x+np.pi),np.cos(y+np.p
plt.show()
```

Wolfram Language:

```
(*lambda is the mean number of points on the unit sphere*)
  poissononsphere[lambda_] :=
  Module[{nrofpoints, phi, theta, radius, refc, polarp},
   nrofpoints = RandomVariate[PoissonDistribution[lambda]];
   polarp =
    Table[{RandomVariate[UniformDistribution[{-Pi, Pi}]],
      ArcSin[RandomVariate[UniformDistribution[{-1, 1}]]]},
     nrofpoints];
   radius = 1;
   refc =
    Table[{radius*Sin[polarp[[i]][[2]] + Pi/2]*
       Cos[polarp[[i]][[1]] + Pi],
      radius*Sin[polarp[[i]][[2]] + Pi/2]*Sin[polarp[[i]][[1]] + Pi],
      radius*Cos[polarp[[i]][[2]] + Pi/2]}, {i, nrofpoints}];
   refc
   ];
   ListPointPlot3D[poissononsphere[500], BoxRatios -> {1, 1, 1}]
```

References:

- D. J. Daley and D. Vere-Jones, The General Poisson Process in "An introduction to the theory of point processes". New York: Springer, 2003, pp. 39.

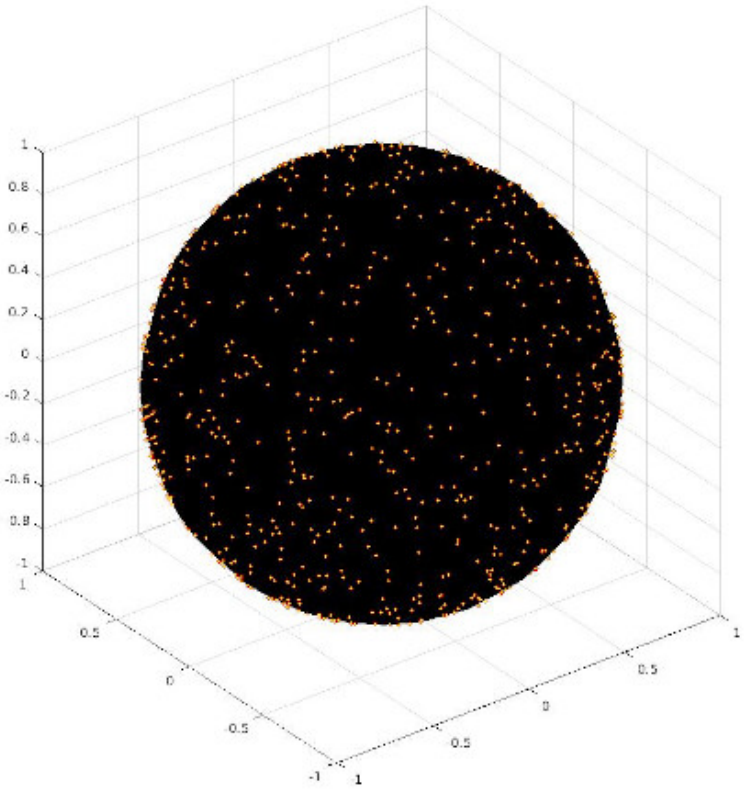


Figure 2: Are the stars Poisson distributed in the sky?

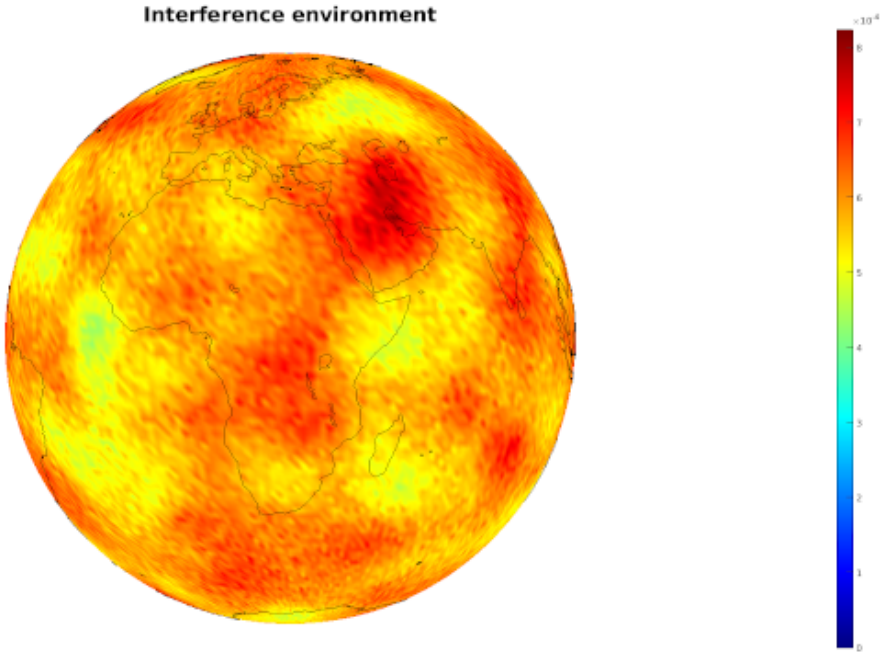


Figure 3: Aggregate interference power in a satellite receiver represented by a color – interfering sources are considered to be Poisson distributed on Earth

- Stoyan, Dietrich. et al. “Stochastic Geometry and Its Applications”. 3rd ed. Chichester: Wiley, 2013. Print.
- H. Paul Keeler’s Blog

2.2 February – Controlling your passwords with pass

Pass is a nice Unix style free and open source wallet for keeping your passwords safe. Here is a brief look how to set it up in Ubuntu.

- Install the application in the terminal

  ```
  sudo apt install pass
  ```

- Check for existing GPG keys

  ```
  gpg --list-keys
  ```

- If no keys were found generate a key pair

  ```
  gpg --generate-key
  ```

- Copy the name of the key and initialize pass

  ```
  pass init ABCDEFGHIJKLMNOPQRSTUV1234,
  ```

  where ABCDEFGHIJKLMNOPQRSTUV1234 is the name of the key.

- Generate a password with

  ```
  pass generate keyfolder/newkey
  ```

  List passwords

  ```
  pass
  ```

  Copy a password to clipboard

  ```
  pass keyfolder/newkey -c
  ```

  For more commands

```
man pass
```

Connect pass to git so it is easy to keep track of changes with multiple machines.

- Export your public and private key to a file with

  ```
  gpg --export --output public.key ABCDEFGHIJKLMNOPQRSTUV1234
  gpg --export-secret-key --output private.key ABCDEFGHIJKLMNOPQRSTUV1234,
  ```

  where ABCDEFGHIJKLMNOPQRSTUV1234 is your key name.

- Now we can initialize the git repository with these keys. Move public.key and private.key through a safe channel to a computer you wish to use pass in. Import the keys to the machine

  ```
  gpg --import public.key
  gpg --import private.key
  ```

- After importing keys to a new machine you can initialize pass

  ```
  pass init ABCDEFGHIJKLMNOPQRSTUV1234
  ```

- Initialize your git repository. Make a new repository named pass-store e.g. to GitHub if you are doing this first time before the following commands

  ```
  pass git init
  pass git remote add origin git@repo.com:myname/pass-store
  ```

- Get password data from the server (from a non-empty repository, otherwise skip)

  ```
  pass git pull origin master --allow-unrelated-histories
  pass git commit -am "firstcommit"
  ```

- Do some changes and pass will automatically commit them. Push and set upstream

  ```
  pass git push --set-upstream origin master
  ```

- From here on you can use the familiar git commands

Figure 4: The white boxes represents buildings, and the little white circle is the transmitter. You can see how reflections will cause the aggregate signal strength to fluctuate by location depending on phases of the incoming reflected waves.

```
pass git pull
pass git push
```

Stay safe :)

References:

- Password Store

2.3 March – Signal propagation in a city

Mobile telephone signal propagation in a city is characterized essentially by obstacles, such as buildings and cars, that attenuates, reflects, refracts and diffracts the signal. Should there be a pure line-of-sight from the transmitter to the receiver, the receiver will always receive a constant power from the transmitter conditioned that the transmitter stays at a constant distance from the receiver – this can be the case for example if you are transmitting to a near-by high base-station antenna. If there is no line-of-sight component, the signal strength will vary according to Rayleigh fading as the transmitter and/or the obstacles moves. In the case that there is only a limited line-of-sight element present, the signal strength will follow Rician fading statistics.

I animated a couple of GIFs to help us to perceive what is going on. The first figure demonstrates how a simple sine signal propagates between buildings. You can observe how the multi-path components sum up to a signal that is amplifying in some locations and degenerating in others. You can look for both Rician conditions (lower right) and Rayleigh conditions (lower left) in the image and see how the signal behaves under those conditions. The figure was obtained by solving the Helmholtz equation by finite element method.

The second figure shows how to aggregate signal power from many mobile transmitters develops in time under Rician fading conditions. The aggregate power can be considered as interference in a receiver. You can see how the high peaks of interference power emerge at random locations. In real life, these kinds of peaks can cause decreased data rates. The figure was obtained by simulating a random walk of points in a realization of the Poisson point process on a plane.

References:

- François Baccelli, Bartlomiej Blaszczyszyn Stochastic Geometry and Wireless Networks, Volume I -Theory
- Nicolae Cindea (2021). Movie to GIF Converter, MATLAB Central File Exchange. Retrieved June 14, 2021.

Figure 5: Interference power field developing in time. Transmitters are Poisson distributed at random locations and moving to random directions, and the aggregate signal strength after fading is plotted. Red color represents the highest mean power of interference and deep blue represents absence of any interference. There is a line-of-sight component is present – that is, we assume Rician fading. Red occurrences will cause remarkable disturbance in communication as interference from other transmitters gets large. Time can't be considered to be in a human perspective scale here.

2.4 April – Frequentist inference in Mathematica

Mathematica provides a package called "Hypothesis Testing" for analyzing random data. The package includes handy objects like "Around", representing a quantity and the uncertainty around it. The following example code returns an Around object representing the confidence interval of the estimated bias of a coin after a repeated trial. As shown in the code, the ListPlot plots Around objects as such.

```
(*Load the Hypothesis Testing Package.*)
Needs["HypothesisTesting`"];

coin[flips_, bias_] :=
 (*Make the experiment and collect the data*)
 Module[{index, realizations, mean, conf, around},
  realizations = {};
  For[index = 1, index <= flips, index++,
   realizations =
     Append[realizations,
      RandomVariate[BernoulliDistribution[bias]]];
   ];

  (*Mean*)
  mean = Mean[realizations[[All]]];
  (*Confidence interval*)
  conf = MeanCI[realizations[[All]]];
  (*Around object*)
  around = Around[mean, Abs[conf[[1]] - conf[[2]]]]
  ]
around = coin[1000, 0.5]
ListPlot[{around}]
```

References:

- reference.wolfram.com

Figure 6: Multi-path faded sine signal

2.5 May – Rayleigh fading audiolized

When a signal is propagating through multiple paths each signal component in each path will be in different phase and of different strength when received. Should there be no line-of-sight component present, the additive signal will fade according to the Rayleigh fading.

For example, a simple sine
wave (links to soundcloud.com – VOLUME ALERT)
can after some multi-path propagation sound something like
this –

assuming that the receiver is moving at a constant speed, so that variation of Doppler shifts in different signal paths will cause the aggregate signal to vary randomly in time.

If we add some Gaussian white noise we notice that the original signal is somehow recognizable from the
noised signal
but the
faded signal
will sometimes get buried under the noise – these events are referred to as **deep fades**.

Here is a GNU Octave or Matlab code for the used Rayleigh simulator which outputs a the signal as audio:

```
%Rayleigh simulator. Jakes model. In Octave remember to load the statistics package.

close all
clear all

tic

N = 20; %Number multipaths.
T = linspace(0,10000,100000); %Time.
v = 0.001; %Speed of the receiver.
randan = pi*rand(1,N); %Random angles w.r.t. receiver.
rI = 1000*rand(1,N); %Random distances of the sources.

%Geometrical stuff. Check for the Jakes model in the reference.
An = @(t) (atan(sin(randan).*rI./(cos(randan).*rI-v*t))).*(cos(randan).*rI>v*t)+...
(pi-atan(sin(randan).*rI./(v*t-cos(randan).*rI))).*(cos(randan).*rI<=v*t);
phis = 2*pi.*(rand(1,N));
wc =pi/2; %Frequency of the signal.
Beta = 2*pi/(1/(wc/(2*pi)));
```

```
theta = @(t) cos(An(t)).*Beta*v.*t+phis;

powers = rand(1,N); %Random powers of the signals in the multipaths.
powers = powers./sum(powers); %Normalize the powers.
Ez = @(t) sum(powers.*cos(wc*t+theta(t)));
EZ = []; %Faded signal.
REF = []; %Original signal.
NOISE = []; %Additional Gaussian noise.
for t =T
EZ = [EZ Ez(t)];
REF = [REF cos(wc*t)];
NOISE = [NOISE 0.9*stdnormal_rnd(1)];
end

%Write the audio files at sampling rate 8000.
audiowrite('EZ.wav',EZ,8000)
audiowrite('EZNOISE.wav',1/2*(EZ+NOISE),8000)
audiowrite('REFNOISE.wav', 1/2*(REF+NOISE), 8000)


toc
plot(T,EZ)
```

And the same in Python:

```
import numpy as np
import math
import matplotlib.pyplot as plt
import sounddevice as sd
import time

#Jakes Rayleigh simulator. Please check for the reference in this site for further details

N = 20 #Number of multipaths.
T = np.linspace(0, 10000, 100000) #Time vector.
v = 0.001 #Speed of the receiver.



randan = np.random.rand(1, N) * math.pi #Random angles w.r.t. receiver.
rI = np.random.rand(1, N) * 1000 #Random distances of the sources.
phis = np.random.rand(1, N) * 2 * math.pi

def An(t):
return (np.arctan(np.sin(randan) * rI / (np.cos(randan) * rI - v * t))) * (
np.cos(randan) * rI > v * t
) + (np.pi - np.arctan(np.sin(randan) * rI / (v * t - np.cos(randan) * rI))) * (
```

```
  np.cos(randan) * rI <= v * t
  )

  wc = np.pi/2 #Frequency of the signal.
  Beta = 2 * math.pi / (1 / (wc / (2 * math.pi)))

def theta(t):
    return np.cos(An(t)) * Beta * v * t +phis

powers = np.random.rand(1,N)*2 #Random powers of the signals in the multipaths.
powers = powers/np.sum(powers) #Normalize powers..

def Ez(t):
    return np.sum(powers * np.cos(wc * t + theta(t)))

EZ = np.vectorize(Ez)(T)
REF = np.vectorize(lambda time : 2*np.cos(wc * time))(T)

#Play and plot.
fs = 8000
sd.play(EZ,fs,blocking = True)
#sd.play(REF,fs,blocking = True)
plt.plot(T, EZ)
plt.show()
```

References:

- William C. Jakes, "Microwave Mobile Communications", IEEE PRESS, 1974.

2.6 June – Why does MIMO work?

Multiple-input and multiple-output (MIMO) antenna technology is used to exploit the multi-path propagation to improve the capacity of a communication link (i.e. more gigabytes of internet speed). In principle, capacity of the link can be increased merely by increasing the power of the transmitting antenna. However – apart from being energy-consuming – this increases also interference to any other receivers should they transmit in the same frequency band. In MIMO the energy is divided among multiple antennas, and the link capacity is improved without using any extra energy and without increasing the interference to other transmitters.

But what is the secret? Here is how I came up with a simple argument based on stochastic geometry. Let us make some assumptions. We assume a flat and infinite Earth (yeah – you can laugh at this "tin foil assumption", but it is often reasonable). In addition, we assume that our communication channel environment consists of many obstacles so that our transmitting antenna and receiving antenna can not see each other. We can assume that we are in a city

full of houses, cars, trees, etc. Then our data signal is prone to propagate to the receiver through multiple paths; for example through distinct streets around different houses. The aggregate signal in the receiver will be Rayleigh faded. The city is full of mobile phones exchanging data with their base stations which are further handing the data to the receiving mobile phones base station and, on the other hand, causing interference to the other base stations. We make a natural assumption that the interfering base stations are distributed according to the Poisson point process around the receiver.

Let's consider that person A is sending a message (a Telegram message, or whatever you wish) to person B. We are interested in the probability that the base station serving person A will successfully transmit the message to a base station serving person B. Under assumptions above and some other simplified assumptions – as derived here, we can express the probability of a successful transmission as:

$$\mathbf{P}[\text{Successful single-antenna transmission}] = e^{-\frac{\pi^2}{2\sqrt{P}}}, \tag{1}$$

where P denotes the mean transmitting power of the base-stations.

In MIMO we use multiple distinct antennas to transmit the same message. In each antenna, encoding of the message should be different so that it can't mix with the information that the other antennas are transmitting. This is possible by orthogonal modulation. Assuming that each message from each antenna will propagate to the receiver in an independent way, we can calculate from equation (1) by complementary probability that **at least one message transmission gets through:**

$$\mathbf{P}[\text{Successful MIMO transmission with N antennas}] = 1 - \left(1 - e^{-\frac{\pi^2}{2\sqrt{P/N}}}\right)^N,$$

where N is the number of transmitting antennas. Notice that we divided the transmitting power P by the number of antennas, so we don't increase the aggregate power at all. In the other hand, should we have no MIMO technology at hand, we could try to improve to increase the link quality just by increasing the power P of a single antenna. In the following figure we compare these two cases.

It is evident that MIMO is a great solution for increasing the throughput of a wireless communication link. Using multiple antennas we can save power and achieve better data-rates than by merely increasing the power of a single transmitter.

References:

- François Baccelli, Bartlomiej Blaszczyszyn Stochastic Geometry and Wireless Networks, Volume 2 - Applications

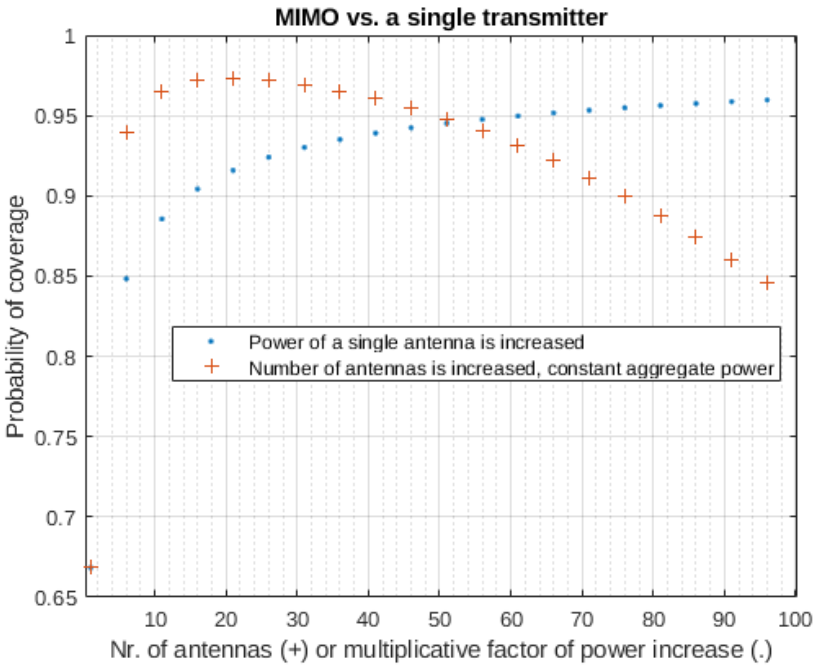


Figure 7: MIMO saves energy and improves the quality

Figure 8: Animated GIF of a noisy QAM symbol

2.7 July – Animating GIFs in GNU Octave

Often it is nice to plot your results as an animation. The following code demonstrates the most effective way to write GIFs in Octave that I could find. Example code outputs an animation of a complex base-band signal symbol in noise. Code uses the quiver plot, but any other plot should do equally well. Matlab compatibility is not guaranteed.

The point is to loop through all frames and save them to a file after each other with functions *getframe* and *imwrite*.

```
function writeGIF()
  clear all;
  close all;
  clear imread;
  clear imwrite;

  N = 100; %%Number of frames.

  %%We use quiver plot as an example, but any other plot works equally.
  signal = 5 + 5*i; %%Baseband symbol.
  signal = signal + randn(1) + randn(1)*i ;   %%Add some Gaussian noise.
  quiver(0,0,real(signal), imag(signal))
  axis([[-10, 10],[-10, 10]])

  %%Write the first frame.
  frame = getframe(); %%Get the current frame.
  imwrite(frame.cdata,'gifu.gif','gif','writemode','overwrite',...
          'LoopCount',inf,'DelayTime',0.1);  %%Write the frame to gifu.gif and use delay tim

  for iii = 1 : N
    iii
    %%Write GIF.
    signal = 5 + 5*i; %%Baseband symbol.
    signal = signal  + rand(1) + randn(1)*i ;   %%Add some Gaussian noise.
    quiver(0,0,real(signal), imag(signal))
    axis([[-10, 10],[-10, 10]])

    frame = getframe(); %%Get the current frame.
    imwrite(frame.cdata, 'gifu.gif','gif','writemode','append','DelayTime',0.1); %%Write the
  end
end
```

Figure 9: High frequencies fit in the tunnel

Figure 10: Wavelengths bigger than the diameter of the tunnel will not get through

References:

- Octave Image Processing

2.8 August – Radio waves in a tunnel

Everyone has some experience of how radio turns into static when one is driving into a tunnel. It just does not work (unless there is a special radiating cable installed inside the tunnel). Once upon a night, I could not get sleep pondering on why does this happen. For example, water seems to behave differently; no matter how narrow the pipe, a water wave will go through it. Someone might have told me that FM radio wavelength is just so large that the wave "does not fit the tunnel". But as intuitively acceptable as this explanation could be (I don't think it is even that), it is not a satisfying explanation as such.

After some research, I found that the explanation lies in the behaviour of Maxwell equations (well, what a surprise). All backs down to the boundary conditions of the electromagnetic field at the tunnel wall: the tangential component of the electric field component of the electromagnetic wave has to be near to zero in the tunnel interface. This leads to a situation where the wave has, in a sense, no room the oscillate inside the tunnel. The zero interface condition here is crucial; for example, it does not generally apply to a water wave and this is the reason a wave in the water fit through the smallest hole.

Intuitive or not, this is what the equations tell us. To demonstrate this fact I solved, using FEM, the Helmholtz equation for a plane-wave in a two-dimensional setting mimicking a tunnel and its entrance. The electromagnetic wave's polarization is so that the electrical component is pointing at the right angle w.r.t. the plane (either towards or against the reader of this page). Boundary conditions inside the 2D tunnel are zero. Solutions colouring shown in the figures following represents the magnitude of the electrical component.

We compare the behaviour of two different wavelengths:

.

References:

- Scattering Problem: Matlab PDE Modeler App
- Interface conditions for electromagnetic fields (Wikipedia article)

2.9 September – The universality of the Poisson point process (breaking a pile of sand)

I thought that it would be nice to write about some purely mathematical subject today, so I decided to briefly discuss the famous Poisson process and its properties under a transformation.

It is a well-known fact that some point processes converge to the Poisson point process after a sufficient amount of right transformations. This is of course why the Poisson p.p appears everywhere in nature; for example, grain of sands in a sand castle – made by a child at a beach on a sunny day – will eventually spread by wind (or water) around the beach. The grains will be randomly displaced here and there forming something approximating a realization of a Poisson p.p., even though the original pile was more ordered.

Let's play the game with a pile of sand spreading to a flat area by random translations of the grains.

Consider a pile of sand at origo. Furthermore, let's assume that the pile is small enough (compared to unit distance) so that it is concentrated in a single point. Let's make a natural assumption that the amount of grains in the pile is Poisson distributed. Now our pile of sand is a Poisson point process concentrated in the origo.

Suppose that after a while, wind moves individual sand grains from the pile directing them with a uniformly distributed angle and to a random normally distributed distance between $[0, \infty)$. In other words, the probability that a sand grain is moved from the origo to a location Y_j inside a circle $B(r_1, r_2)$ is $2\int_{r_1}^{r^2} \frac{1}{\sqrt{2\pi}} e^{-y^2} dy$.

After a random translation of all points the Laplace functional of the *distances* of the points is

$$\begin{aligned}
\mathcal{L}_\tau(f) &= \mathbb{E} \exp\left[-\sum_i f(|Y_i|)\right] = \mathbb{E} \prod_i \exp\left[-f(|Y_i|)\right] \\
&= \mathbb{E}_n \left[(2\pi)^{-n/2} \int_0^\infty \cdots \int_0^\infty \prod_{i=0}^{n} \exp\left[-f(y_i)\right] \frac{2}{\sqrt{2\pi}} e^{-y_i^2} dy_1 \ldots dy_n\right] \\
&= \mathbb{E}_n \left[(2\pi)^{-n/2} \left(\int_0^\infty 2\exp\left[-f(y) - y^2\right] dy\right)^n\right] \\
&= \mathbb{E}_n \exp\left[\sum_{i=0}^{n} \log\left(\int_0^\infty \exp\left[-f(y)\right] \frac{2e^{-y^2}}{\sqrt{2\pi}} dy\right)\right]
\end{aligned}$$

Evaluating the Poisson Laplace functional $\mathcal{L}(g)$ of the sand pile with $g =$

$-\log\left(\int_0^\infty \exp\left[-f(y)\right]\frac{e^{-y^2}}{\sqrt{2\pi}}dy\right)$ we get

$$\begin{aligned}
\mathcal{L}(g) = \mathcal{L}_\tau(f) &= \exp\left[-\int_{\mathbb{R}^2} 1 - e^{\log\left(\int_0^\infty \exp[-f(y)]\frac{2e^{-y^2}}{\sqrt{2\pi}}dy\right)}\Lambda(dx)\right] \\
&= \exp\left[-\int_{\mathbb{R}^2}\left(1-\int_0^\infty \exp[-f(y)]\frac{2e^{-y^2}}{\sqrt{2\pi}}dy\right)\Lambda(dx)\right] \\
&= \exp\left[-\int_{\mathbb{R}^2}\int_0^\infty \left(1-\exp[-f(y)]\right)\frac{2e^{-y^2}}{\sqrt{2\pi}}dy\Lambda(dx)\right] \\
&= \exp\left[-\int_0^\infty \left(1-\exp[-f(y)]\right)\sqrt{\frac{2}{\pi}}e^{-y^2}Ndy\right], (1)
\end{aligned}$$

where N denotes the expected number of points in the sand pile (Campbell's theorem). But the formula (1) is the Laplace functional of the Poisson p.p. on $[0,\infty)$ with a density parameter $\sqrt{\frac{2}{\pi}}e^{-x^2}N$. As the direction of the translation of a sand grain was uniformly distributed, we can conclude that the sand grains are Poisson distributed according to the Poisson p.p. with the two-dimensional multivariate Gaussian distribution as the density.

The result above applies to any translations as long as the translation is independently applied to every point (or grain); the Poisson point process remains Poisson in a transformation. Here we had to make the Poisson assumption also for the pile so that the result above could be derived. But indeed, we can show that non-Poisson processes approach Poisson p.p. when the points are translated in a certain manner. However, I leave this is out of the scope of this blog entry for now, and take the remark made here as a simple (possibly a bit weird) example of the universality of the Poisson point process.

.

References:

- D. J. Daley and D. Vere-Jones, The General Poisson Process in "An introduction to the theory of point processes". New York: Springer, Volume II: General Theory and Structure, 2008, pp. 166.
- François Baccelli, Bartlomiej Blaszczyszyn Stochastic Geometry and Wireless Networks, Volume I -Theory

2.10 October – Fast Fourier transform in GNU Octave

Fast fourier transform (FFT) is an algorithm that computes the discrete Fourier transform.

FFT is especially handy in real-time digital signal processing. Digital computers are working on discrete data, thus a input signal is always sampled with

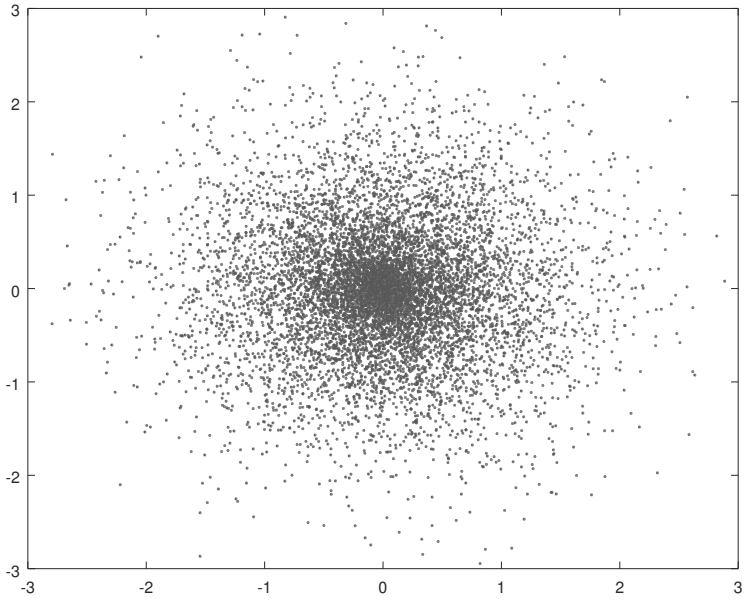


Figure 11: 10000 points randomly moved from the origo to the plane

some sampling rate F_s (Hz). By the Nyquist-Shannon sampling theorem, sampling captures frequencies under $F_s/2$.

Octave calculates the FFT of a discrete signal. In the following code, we calculate the normalized FFT and plot the frequency spectrum w.r.t. the frequency. Signal vector and sampling frequency are given as input.

```
##Calculates the FFT and plots the frequency spectrum of a signal. Sampletimes have to start

function fftvector = plotfft(signal, Fs)
  N = length(signal); #Signal length.
  FFT =  fft(signal);
  if(mod(N,2) == 0) #Check if signal length is odd or even.
    FFT = 2*FFT(1 : N/2)/N;
    f = Fs*(0 : (N - 1)/2)/(N - 1);
  else
    FFT = 2*FFT(1 : (N - 1)/2)/N;
    f = Fs*(0 : (N - 2)/2)/(N - 1);
  end
  fftvector = [f; FFT];

  figure(1)
  plot(f, abs(FFT));
  title('Fast fourier transform')
  xlabel('Frequency (Hz)')
  print  plot.jpg
end
```

The following figure shows my track Tappimarssi in the frequency domain. The track was imported to Octave by

```
[signal, Fs] = audioread('Tappimarssi.wav');
```

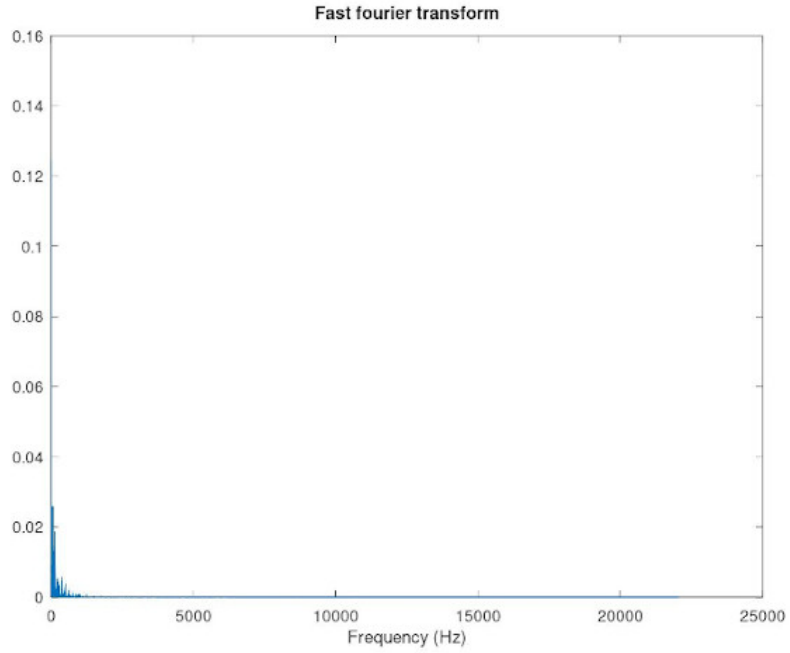
- Octave Documentation – Signal Processing

2.11 November – Digital filtering

Let's construct a simple digital band-pass filter based in the sinc filter.

As everyone know, the sinc function is the impulse response of a low-pass filter. That is, the convolution of a signal $S(t)$ and the sinc function $t \mapsto \frac{\sin(2\pi B_L t)}{\pi t}$ will produce a low-pass filtered signal $S(t)$ without frequencies higher than B_L. Similarly, the function $t \mapsto \delta(t) - \frac{\sin(2\pi B_H t)}{\pi t}$, where $\delta(t)$ represents the Dirac delta function, is the frequency response of the ideal high-pass filter of frequency B_H.

Heuristically, we can right away derive the discrete versions of the impulse responses:



$$\mathbb{Z} \ni i \mapsto \frac{\sin(2\pi \frac{B_L}{f_c} i)}{\pi i},$$

for the low-pass filter, and

$$i \mapsto \delta[i] - \frac{\sin(2\pi \frac{B_H}{f_c} i)}{\pi i},$$

for the high-pass filter. The variable f_c is the sampling frequency and $i \mapsto \delta[i] := \delta_{0i}$ is the Kronecker delta.

Now, having a discrete signal at hand, band-pass filtering is (in the simplest approach) just a matter of discrete convolution of the signal with the functions above. The following octave code does the job.

```
##Sinc band-pass filter. f0 = B_L/fs, f1 = B_H/fs

function filtered = sincfilter(signal, f0, f1)
  if(mod(length(signal), 2) != 0) #Check if the signal length is even or odd.
    signal = signal(1 : length(signal) - 1);
  end

  M = 10000; #Increase this to increase accuracy.
  sincF = zeros(1, M);
```

```
  for m =  -M/2 + 1 : 1 : M/2
    if(!(m  == 0))
      sincF(m + M/2) = sin(2*pi*f1*m)./(pi*m);
    else
      sincF(m + M/2) = 2*f1;
    end
  end

  sincH = zeros(1,M);
  for m =  -M/2 + 1 : 1 : M/2
    if(!(m  == 0))
      sincH(m + M/2) = -sin(2*pi*f0*m)./(pi*m);
    else
      sincH(m + M/2) = -2*f0 + 1;
    end
  end

  ##Plot stuff.
  figure(1)
  plot(sincF)
  figure(2)
  plot(signal)

  tic
  filtered = conv(signal, sincF, "same");
  filtered = conv(filtered, sincH, "same");
  toc

  figure(3)
  plot(filtered)
end
```

For more accuracy and effectiveness, one should use windowed or recursive filters. You can check in the references for more sophisticated methods!

References:

- Steven W. Smith, "Digital Signal Processing – A Practical Guide for Engineers and Scientists", Elsevier Science, 2003.

2.12 December – Contour plot in Mathematica

A contour plot is a nice way to represent 2D data. Mathematica has great for constructing such plots, but you might want to tweak the fonts and graphics a bit. Sometimes fonts should be enlarged so that they are visible e.g. in a journal article where there is not too much space to use.

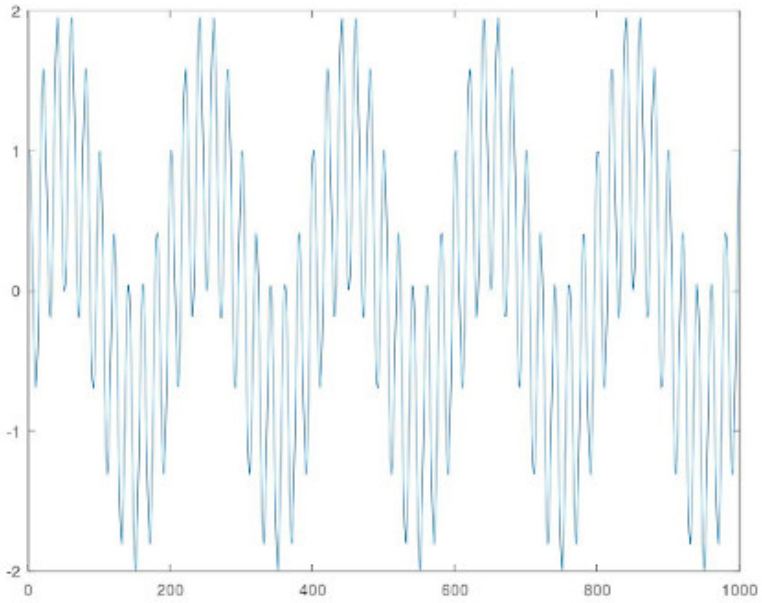


Figure 12: Signal of form $\cos(20\pi t) + \sin(2\pi t)$

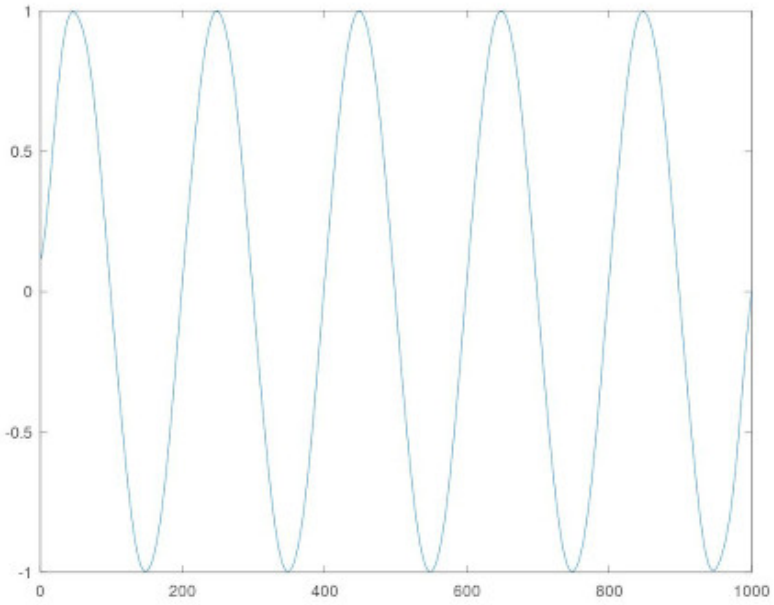
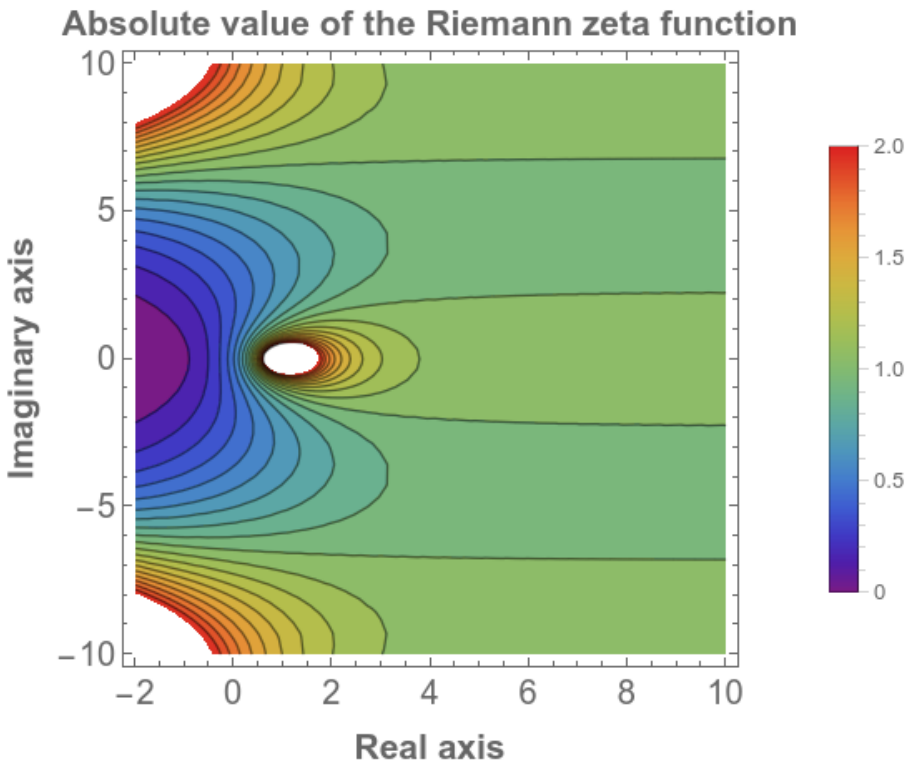


Figure 13: The same signal with the first term filtered away



The following code returns a contour plot of the Riemann zeta function's absolute value in an area in the complex plane; $(-2, 10) \times (-10i, 10i) \subset \mathbb{C}$. Text fonts are enlarged from default. Only values under max = 2 are plotted. Values above that are "clipped" away as white colour. By definition of the absolute value, values are always more than 0, otherwise values under 0 would get clipped away with black colour. Contours are drawn in intervals of 0.1, as adjusted in the Range function.

```
max = 2;
ContourPlot[Abs[N[Zeta[x + y*I]]], {x, -2, 10}, {y, -10, 10},
 PlotLegends -> BarLegend[{"Rainbow", {0, max}}],
 ColorFunction -> "Rainbow", ContourStyle -> Black,
 Contours -> Function[{min, max}, Range[min, max, 0.1]],
 PlotLabel ->
  Style["Absolute value of the Riemann zeta function", 16, Bold],
 BaseStyle -> {FontSize -> 15},
 FrameLabel -> {Style["Real axis" , 16, Bold],
   Style["Imaginary axis", 16, Bold]}, PlotRange -> {0, max},
 ClippingStyle -> {Black, White}]
```

References:

- reference.wolfram.com
- Díaz-Francés, Eloísa; Rubio, Francisco J. (2012-01-24). "On the existence of a normal approximation to the distribution of the ratio of two independent normal random variables". Statistical Papers. Springer Science and Business Media LLC.

3 Blog posts 2022

General thoughts on mathematics and engineering, practical instructions, artistic non-sense. The themes of this blog are much inspired by the challenges encountered in my professional life. The blog serves also as my personal notes.

3.1 January – Inverse of a Gaussian variable

There is quite a pile of literature on the subject of the inverse of a Gaussian distributed variable (this should not be fixed with inverse Gaussian distribution – it is a different matter). In fact, the inverse distribution is ill-behaved; the mean and variance does not generally exist.

I came up with a simple approximation that works well if the mean is large enough and the variance is small enough (I have not worked out the details of the exact conditions for this approximation. However, the results can be verified by Monte Carlo simulations).

First, approximate the Gaussian distributed variable $X \sim \mathcal{N}(\mu, \sigma^2)$ by a log-normally distributed variable $X \approx Y \sim \text{Lognormal}(\mu_{\text{LN}}, \sigma_{\text{LN}})$, with corresponding mean and variance, i.e.

$$\mu = \exp\left(\mu_{\text{LN}} + \frac{\sigma_{\text{LN}}^2}{2}\right)$$

and

$$\sigma = (\exp(\sigma_{\text{LN}}^2) - 1)\exp(2\mu_{\text{LN}} + \sigma_{\text{LN}}^2).$$

We leave the solving of μ_{LN} and σ_{LN} as an easy exercise for the reader.

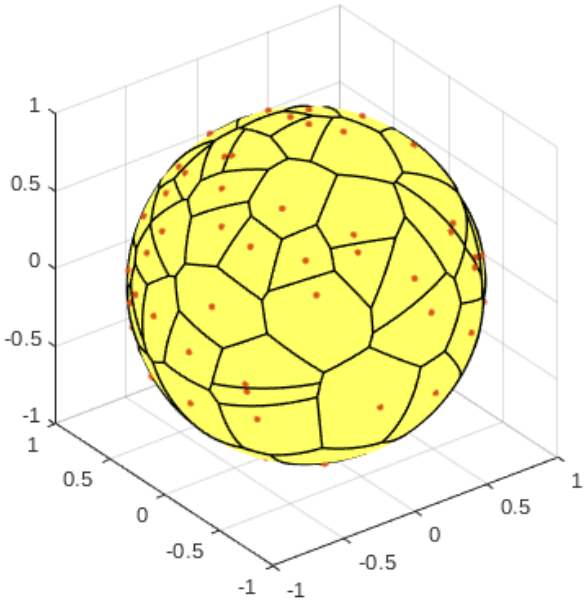
Using the theory of log-normal distribution, the inverse of X is now given by

$$1/X \approx 1/Y \sim \text{Lognormal}(-\mu_{\text{LN}}, \sigma_{\text{LN}}).$$

That's it!

References:

- Log-normal distribution
- Díaz-Francés, Eloísa; Rubio, Francisco J. (2012-01-24). "On the existence of a normal approximation to the distribution of the ratio of two independent normal random variables". Statistical Papers. Springer Science and Business Media LLC.



3.2 February – Voronoi tesselation on a sphere

Voronoi tessellation consists neighborhood areas of a given set of points, or Voronoi cells: every cell is surrounded by the area that consists of locations that are closer to the cell than any other cell. In addition to the artistic value, Voronoi Diagram, or Voronoi tessellation, have applications in wireless communications (among the million other things).

The next Matlab code produces a Voronoi tessellation on a sphere Poisson points as cells. It is based in Grady Wrights codes openly available in Github.

```
%%Create Voronoi tesselation around Poisson points on a Sphere.

clear all;
close all;

addpath(fullfile(cd,'rbfsphere'));
density = 100;
X = poissononsphere(density)'; %Poisson points in spherical coordinates.
voronoiSph(X); %This function is downloaded from gradywright's Github. It is placed in the f

function refc = poissononsphere(density)
```

```
  yMin = -1; yMax = 1;
  xMin = -pi; xMax = pi;

  xDelta = xMax - xMin; yDelta = yMax - yMin; %Rectangle dimensions
  numbPoints = poissrnd(density);    %Number of points in the area is a Poisson variable of
  x = xDelta*(rand(numbPoints,1)) + xMin;    %Pick points from uniform distribution
  y = yDelta*(rand(numbPoints,1)) + yMin;    %Map referencepoints to geographical coordinate
  ref = [x y]';

  refs = [x'; asin(y)'];%Map geographical coordinates to Cartesian coordinates on a unit cir
  r = 1;
  refc = [r*sin(refs(2,:)+pi/2).*cos(refs(1,:)+pi);...
          r*sin(refs(2,:)+pi/2).*sin(refs(1,:)+pi);...
          r*cos(refs(2,:)+pi/2)];
end
```

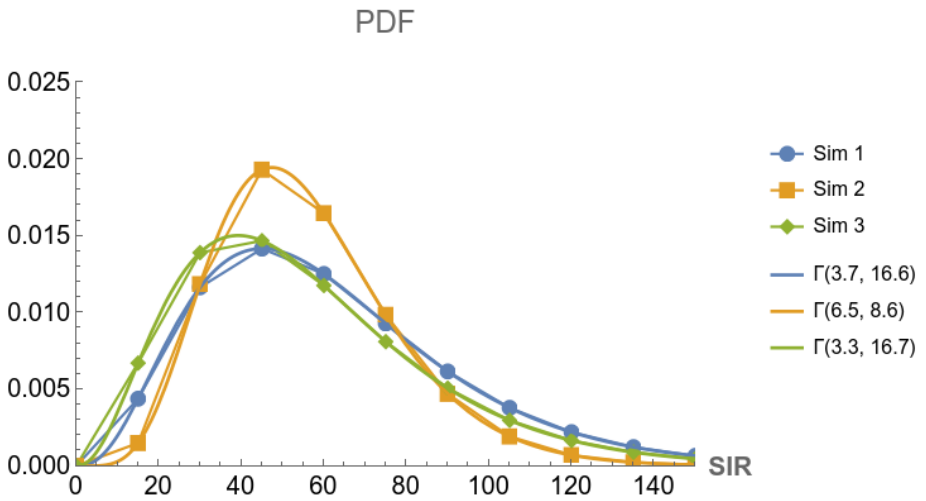
References:

- gradywright Github

3.3 March – Plotting in Mathematica

Here's a plot for Mathematica that you can use as a template to plot ListLinePlot and Plot in the same figure. Font sizes are increased to be well visible in a research article.

```
xmax = 150;
ymax = 0.025;
Show[
 (*Histogram[res1,Automatic,"PDF"],*)

 ListLinePlot[{Table[{t,
     PDF[GammaDistribution[3.715287738432738`, 16.575007504627795`]][
      t]}, {t, 0, xmax, xmax/10}],
   Table[{t,
     PDF[GammaDistribution[6.502676225156808`, 8.60919186301341`]][
      t]}, {t, 0, xmax, xmax/10}],
   Table[{t,
     PDF[GammaDistribution[3.3444580518527207`, 16.73897127052263`]][
      t]}, {t, 0, xmax, xmax/10}]},
  PlotRange -> {{0, xmax}, {0, ymax}},
  PlotLabel -> PDF,
  AxesLabel -> {Style["SIR", 16, Bold], Style["", 16, Bold]},
  PlotLegends -> Placed[{"Sim 1", "Sim 2", "Sim 3"}, Right],
  PlotMarkers -> {Automatic, 10}, BaseStyle -> {FontSize -> 15}],
 Plot[
```




```
 {
  PDF[GammaDistribution[3.715287738432738`, 16.575007504627795`]][
   t],
  PDF[GammaDistribution[6.502676225156808`, 8.60919186301341`]][t],
  PDF[GammaDistribution[3.3444580518527207`, 16.73897127052263`]][t]
  }, {t, 0.01, xmax}, PlotStyle -> Thick,
 PlotLegends ->
  Placed[{"\[CapitalGamma](3.7, 16.6)", "\[CapitalGamma](6.5, 8.6)",
    "\[CapitalGamma](3.3, 16.7)"}, Right]]
]
```

Output:
References:

- reference.wolfram.com

3.4 April – Satellite communication toolbox in Matlab

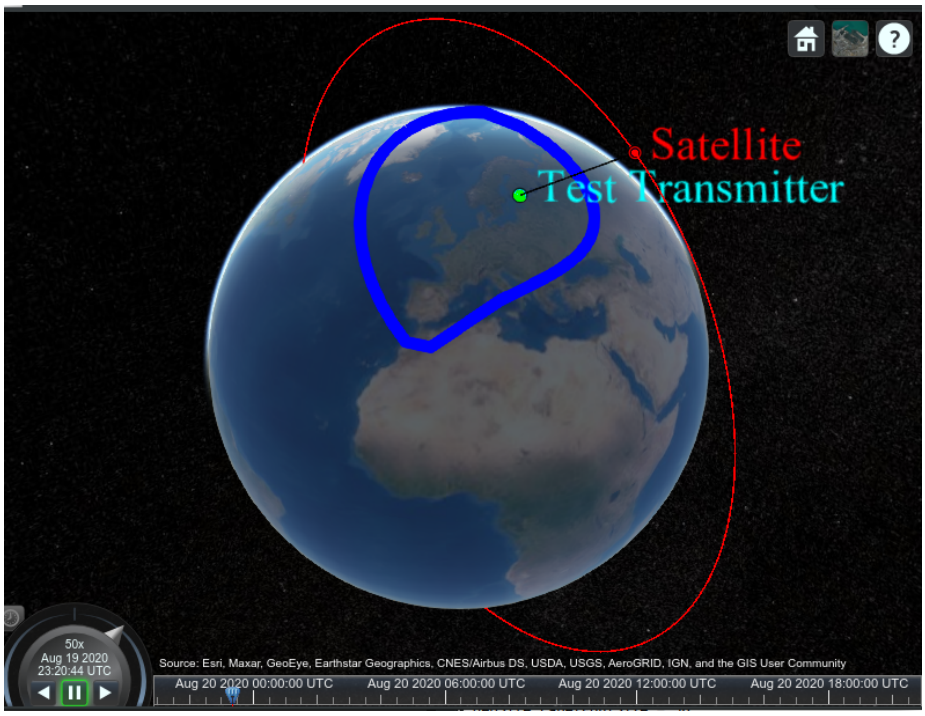
Matlab's Satellite Communication Toolbox can be used to study and visualize satellite networks. The following code places a terrestrial station in Otaniemi, and a LEO satellite in 2000 km tracking the station. Satellite have to be at least in a elevation angle of 35 degrees to contact to the base station. Satellites field of view and a 3 dB footprint of width 1.5 degrees is also represented.

```
clear all;
close all;

startTime = datetime(2020,8,19,20,55,0); % 19 August 2020 8:55 PM UTC
stopTime = startTime + days(1);          % 20 August 2020 8:55 PM UTC
```

```
sampleTime = 60;                          % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);


semiMajorAxis = (6378 + 2000)*1000;          % meters
eccentricity = 0;
inclination = 90;                     % degrees
rightAscensionOfAscendingNode = 0; % degrees
argumentOfPeriapsis = 0;           % degrees
trueAnomaly = 0;                   % degrees
sat1 = satellite(sc, ...
    semiMajorAxis, ...
    eccentricity, ...
    inclination, ...
    rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis, ...
    trueAnomaly, ...
                "Name","Satellite", ...
                "OrbitPropagator","two-body-keplerian");

sat1.LabelFontSize = 30;
cam1 = conicalSensor(sat1,"MaxViewAngle",90)
cam2 = conicalSensor(sat1,"MaxViewAngle",1.5/2)

name = "Test Transmitter";
minElevationAngle = 35; % degrees
lat = 60.185;
lon = 24.83;
geoSite = groundStation(sc, lat, lon, "Name", name, "MinElevationAngle", minElevationAngle)
geoSite.LabelFontSize = 30;
ac1 = access(cam1,geoSite);
ac2 = access(cam2,geoSite);

fov1 = fieldOfView(cam1, "LineColor",'blue', "LineWidth", 10)
ac1.LineColor = 'black';

fov2 = fieldOfView(cam2, "LineColor", 'green', "LineWidth", 10)

pointAt(sat1,geoSite);

play(sc);
```

Output:
References:

- Mathworks



3.5 May – Digital to analog modulation

Digital signal consists of values at discrete times, whereas analog signal is continuous in time t. As the orthogonal set of sinc functions

$$\{t \mapsto \text{sinc}(F_s t - n)\}_n,$$

spans the space of signals of bandwidth $F_s/2$, an analog signal can be (uniquely) produced from a digital signal of length N

$$\{x[n]\}_{n=0}^{N-1}$$

as a superposition of the sinc basis-functions

$$S(t) = \sum_{n=0}^{N-1} x[n]\text{sinc}(F_s t - n),$$

where F_s is the sampling rate of the digital signal.

The following GNU Octave function produces the analog signal from a given digital signal

```
%%Function returns the values at times T of the analog signal of the corresponding digital s
function xb = digitaltoanalog(T, digitalsignal, Fs)
  xb = [];
```

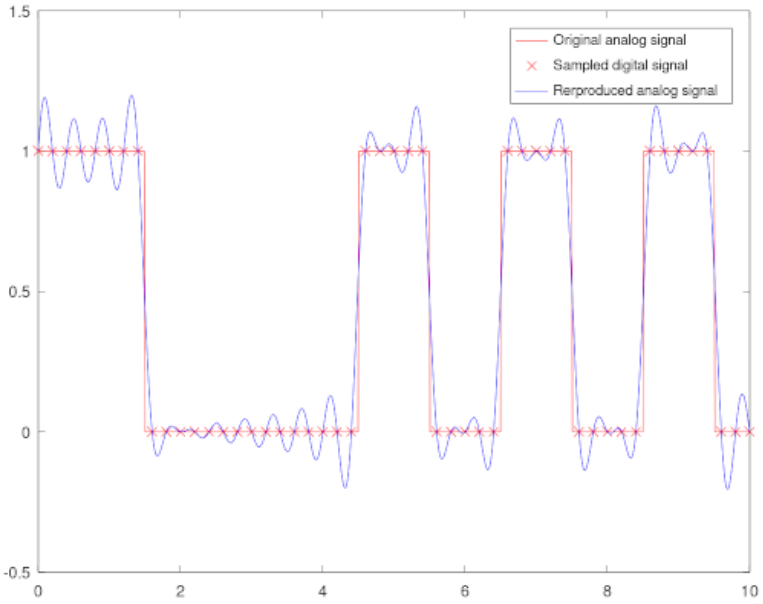
```
  for t = T
    xb =  [xb sum(digitalsignal.*sinc(Fs*t - (0 : length(digitalsignal) - 1)))];
  end
end
```

The following code simulates a rectangle pulse train signal given as input, samples the input to a digital signal x, and modulates the sampled digital signal back to an analog signal S. As the Nyquist rate restricts the bandwidth of the signal S to $F_s/2$, information is lost in the sampling stage, because the original rectangle train has an infinite bandwidth.

```
pkg load signal;
close all;
clear all;

input  = @(t) pulstran(t,[0,1,5,7,9],"rectpuls") %%The original analog signal at time t.

Fs = 5; %Sampling rate.
t = 10 %Time length of the signal.
Ts = 0 : 1/Fs : t; %Sampling time instances.
x =  input(Ts); %Sampled values, i.e. the digital signal.

Ta = linspace(0, t, 1000); %Analog signal time instances for the plot.
S = digitaltoanalog(Ta, x, Fs); %Modulated analog signal.

%Plot.
plot(Ta, input(Ta), 'color', 'r');
hold on;
plot(Ts, x, 'x', 'color', 'r')
plot(Ta, S, 'color', 'b');
legend( 'Original analog signal','Sampled digital signal', 'Rerproduced analog signal')
```

References:

- Tse, David., and Pramod. Viswanath. Fundamentals of Wireless Communication. Cambridge, U.K.;: C


Original analog signal
Sampled digital signal
Rerproduced analog signal
1.5
1
0.5
0
-0.5
0
2
4
6
8
10