```
(* Mathematica code for Model3 of Eshra et al. eLife 2021 *)
(* Stefan Hallermann and Hartmut Schmidt Aug 2021 *)
```

Import

general

```
In[•]:= (* imported data all Ca in uM, all times in ms,
    all ampltiudes and Nv in vesicels *)

In[•]:= CmToVesConversionFactor = (1/90.12) * (1/70*^-18);(* explained in methods *)

In[•]:= rrp = 10;(* pool of release-ready vesicles per connection *)

        dir = NotebookDirectory[];
        SetDirectory[dir];
        dataFolder = "../data to fit/";
```

tau1 Cm 5kHz

```
In[•]:= data = Import[dataFolder <> "all_t1_v02_C5.txt", "Table"];
```

```
In[]:= dataT1C5Ca = 0.001 * data[[All, 1]];
dataT1C5RelRate = 1000. * data[[All, 2]];
dataT1C5Delay = data[[All, 3]];
dataT1C5ChiRatio = data[[All, 4]];
dataT1C5Amplitude = CmToVesConversionFactor data[[All, 5]];

dataT1C5Nv = Table[0, {7}];
tmp1 = 1;
tmp2 = 6;
dataT1C5Nv[[tmp1]] = CmToVesConversionFactor data[[All, tmp2]];
tmp1 += 1;
tmp2 += 1;
dataT1C5Nv[[tmp1]] = CmToVesConversionFactor data[[All, tmp2]];
tmp1 += 1;
tmp2 += 1;
dataT1C5Nv[[tmp1]] = CmToVesConversionFactor data[[All, tmp2]];
tmp1 += 1;
tmp2 += 1;
dataT1C5Nv[[tmp1]] = CmToVesConversionFactor data[[All, tmp2]];
tmp1 += 1;
tmp2 += 1;
dataT1C5Nv[[tmp1]] = CmToVesConversionFactor data[[All, tmp2]];
tmp1 += 1;
tmp2 += 1;
dataT1C5Nv[[tmp1]] = CmToVesConversionFactor data[[All, tmp2]];
tmp1 += 1;
tmp2 += 1;
dataT1C5Nv[[tmp1]] = CmToVesConversionFactor data[[All, tmp2]];
```

tau1 Cm 10kHz

```
In[]:= data = Import[dataFolder <> "all_t1_v02_C10.txt", "Table"];
```

```
In[•]:= dataT1C10Ca = 0.001 * data[[All, 1]];
dataT1C10RelRate = 1000. * data[[All, 2]];
dataT1C10Delay = data[[All, 3]];
dataT1C10ChiRatio = data[[All, 4]];
dataT1C10Amplitude = CmToVesConversionFactor data[[All, 5]];

dataT1C10Nv = Table[0, {7}];
tmp1 = 1;
tmp2 = 6;
dataT1C10Nv[[tmp1]] = CmToVesConversionFactor data[[All, tmp2]];
tmp1 += 1;
tmp2 += 1;
dataT1C10Nv[[tmp1]] = CmToVesConversionFactor data[[All, tmp2]];
tmp1 += 1;
tmp2 += 1;
dataT1C10Nv[[tmp1]] = CmToVesConversionFactor data[[All, tmp2]];
tmp1 += 1;
tmp2 += 1;
dataT1C10Nv[[tmp1]] = CmToVesConversionFactor data[[All, tmp2]];
tmp1 += 1;
tmp2 += 1;
dataT1C10Nv[[tmp1]] = CmToVesConversionFactor data[[All, tmp2]];
tmp1 += 1;
tmp2 += 1;
dataT1C10Nv[[tmp1]] = CmToVesConversionFactor data[[All, tmp2]];
tmp1 += 1;
tmp2 += 1;
dataT1C10Nv[[tmp1]] = CmToVesConversionFactor data[[All, tmp2]];
```

tau1 Deconv

```
In[•]:= data = Import[dataFolder <> "all_t1_v02_D.txt", "Table"];
```

```
In[•]:= dataT1DCa = 0.001 * data[[All, 1]];
dataT1DRelRate = 1000. * data[[All, 2]];
dataT1DDelay = data[[All, 3]];
dataT1DChiRatio = data[[All, 4]];
dataT1DAmplitude = data[[All, 5]];

dataT1DNv = Table[0, {7}];
tmp1 = 1; tmp2 = 6; dataT1DNv[[tmp1]] = data[[All, tmp2]];
tmp1 += 1; tmp2 += 1; dataT1DNv[[tmp1]] = data[[All, tmp2]];
tmp1 += 1; tmp2 += 1; dataT1DNv[[tmp1]] = data[[All, tmp2]];
tmp1 += 1; tmp2 += 1; dataT1DNv[[tmp1]] = data[[All, tmp2]];
tmp1 += 1; tmp2 += 1; dataT1DNv[[tmp1]] = data[[All, tmp2]];
tmp1 += 1; tmp2 += 1; dataT1DNv[[tmp1]] = data[[All, tmp2]];
tmp1 += 1; tmp2 += 1; dataT1DNv[[tmp1]] = data[[All, tmp2]];
```

tau2 Cm 5kHz

```
In[•]:= data = Import[dataFolder <> "all_t2_v02_C5.txt", "Table"];
```

```
In[•]:= dataT2C5Ca = 0.001 * data[[All, 1]];
dataT2C5RelRate = 1000. * data[[All, 2]];
dataT2C5Amplitude2 = CmToVesConversionFactor data[[All, 3]];
dataT2C5Amplitude1 = CmToVesConversionFactor data[[All, 4]];
```

tau2 Cm 10kHz

```
In[•]:= data = Import[dataFolder <> "all_t2_v02_C10.txt", "Table"];
```

```
In[•]:= dataT2C10Ca = 0.001 * data[[All, 1]];
dataT2C10RelRate = 1000. * data[[All, 2]];
dataT2C10Amplitude2 = CmToVesConversionFactor data[[All, 3]];
dataT2C10Amplitude1 = CmToVesConversionFactor data[[All, 4]];
```

tau2 Deconv

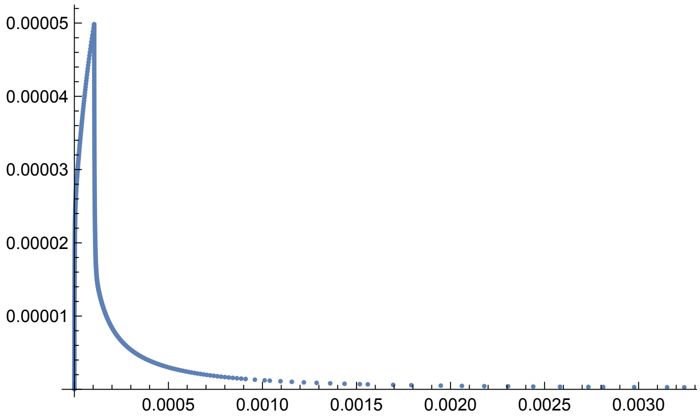
```
In[•]:= data = Import[dataFolder <> "all_t2_v02_D.txt", "Table"];
```

```
In[•]:= dataT2DCa = 0.001 * data[[All, 1]];
dataT2DRelRate = 1000. * data[[All, 2]];
dataT2DAmplitude2 = data[[All, 3]];
dataT2DAmplitude1 = data[[All, 4]];
```

local Ca

In[•]:=
```
data = Import[dataFolder <> "local Ca at 20 nm in uM and ms.txt", "Table"];
dataLocalCa = 1*^-6 data[[All, 1]];
dataLocalCaTime = 1*^-3 data[[All, 2]];
ListPlot[Transpose[{dataLocalCaTime, dataLocalCa}], PlotRange → All]
```

Out[•]=



General parameters and definitions

general stuff

In[]:=
```
(* for calulations: time in s, Ca in M *)
numberOfFitParamToBeSaved = 16;
(*
1 max release

 Mono
2 chi2Mono
3 delayMono
4 ampMono
5 1/tau1Mono

 Bi
6 chi2Mono/chi2Bi
7 delay
8 amp
9 amp1 (=amp*relative amp1)
10 1/tau1
11 1/tau2

 merge
12 delay
13 amp
14 amp1
15 1/tau1
16 1/tau2
*)
cursorStart = -0.002;(*s*)
cursorEnd = 0.01;(*s*)
cursorEndLong = 0.061;(*s*)
timeOfNv = {0.0001, 0.0002, 0.001, 0.005, 0.01, 0.1, 0.4};
SeedRandom[1];
myMaxIterations = 100;
```

In[]:=
```
timeStart = AbsoluteTime[]
```

Out[]= $3.839313740539294 \times 10^{9}$

noiseRepeats

In[•]:=
```
noiseRepeats = 3;
(* should be increased to 50 for a full dataset *)
myQuantile1 = 0.25;
myQuantile2 = 0.75;
```

export parameters

In[•]:=
```
dtOfPlotsForExport = 20*^-5;
exportYes = 1;
```

sampling and myNoise

In[•]:=
```
samplingOfDataInKHzC5 = 5;
myNoiseC5 = CmToVesConversionFactor * 1.36937*^-14 / rrp(*cannot be 0*)
signalToNoiseRatioC5 = 1.;(*minimum s-to-n-ratio to attempt fitting*)
dtOfDataC5 = (1 / (1000 * samplingOfDataInKHzC5));

samplingOfDataInKHzC10 = 10;
myNoiseC10 = CmToVesConversionFactor * 1.67583*^-14 / rrp(*cannot be 0*)
signalToNoiseRatioC10 = 1.;(*minimum s-to-n-ratio to attempt fitting*)
dtOfDataC10 = (1 / (1000 * samplingOfDataInKHzC10));

samplingOfDataInKHzD = 10;
myNoiseD = 0.367584 / rrp(*cannot be 0*)
signalToNoiseRatioD = 1.;(*minimum s-to-n-ratio to attempt fitting*)
dtOfDataD = (1 / (1000 * samplingOfDataInKHzD));

samplingOfDataInKHzLong = 1;
myNoiseLong = myNoiseC5;(*cannot be 0*)
dtOfDataLong = (1 / (1000 * samplingOfDataInKHzLong));
```

Out[•]= 0.217071

Out[•]= 0.265651

Out[•]= 0.0367584

number of simulations per DMN

In[•]:=
```
aNumberDMN05 = 2;
aNumberDMN2 = 2;
aNumberDMN10 = 2;

(* for full dataset: *)
(*
aNumberDMN05=2*20;
aNumberDMN2=2*17;
aNumberDMN10=2*10;
*)
```

Exp fit function

In[•]:=
```
myFitMono[t_] :=
  If[t <= delayMono, 0, ampMono (1 - Exp[-(t - delayMono) / tau1Mono])];

myFitBi[t_] := If[t <= delay, 0,
   amp (1 - amp1 Exp[-(t - delay) / tau1] - (1 - amp1) Exp[-(t - delay) / tau2])];

(*guess for 10 uM; will be changed according to a power of 1 law*)(*in s*)
ampGuess = 2.;(*each pool has size 1.0*)
tau1Guess = 0.001;(*in s*)
delayGuess = 0.0005;(*in s*)
amp1Guess = 0.5;
```

Calculate Ca transients

First, the resting conditions are numerically calculated. Subsequently, the resulting values are used as initial conditions for the main simulations of the flash-evoked Ca2+ transitions . All calculations are repeated in a loop with increasing uncaging efficacy for three different DMN concentrations. The resulting free Ca2+ concentration is later used to drive the release schemes.

General definitions for all DMN conc.

In[•]:=
```
CaListReal = CaListDye = {};
```

0.5 mM DMN

general parameters

In[•]:=
```
TimeWindow = 0.006; (*End of simulation*)
tflash = 0.0;  (*Time of flash*)
PlStart = 0.; (*Plot start*)

af = 0.67; (*fast uncaging fraction; Faas et al*)

(*Select dye*)
OGB1 = 0;
OGB5N = 0;
OGB6F = 0;
Fluo5F = 1;

CaRest = 227. * 10 ^ -9; (*Free pre-flash rersting Ca;
equilibrates with all buffers and DM*)
MgT = 0.5 * 10 ^ -3; (*total Mg in pipette*)
γ = 0.; (*Pump rate*)

(*Concentrations of dye, buffers, DM*)
OGtotal  = 50. * 10^-6 ;
ATPtotal  = 5. * 10^-3 ;
MBtotal  = 480. * 10^-6 ;(*(*Delvendahl, PNAS, 2015*)*)

DMT  = 0.5 * 10^-3;  (*total concentration of DMn *)
(*uncaging efficiency*)
aStartDMN05 = 0.08;
aEndDMN05 = 0.5;
```

definitions and loop

In[•]:=
```
(*Dye*)
If[OGB1 == 1,
 kOnOG  = 4.3 * 10^8;
kOffOG  = 103.;]

If[OGB5N == 1,
 kOnOG  = 2.5 * 10^8;
kOffOG  = 6000.;]

If[OGB6F == 1,
 kOnOG  = 3. * 10^8;
kOffOG  = 900.;]

If[Fluo5F == 1,
```

```
 kOnOG  = 3. * 10^8;
kOffOG = 249.;] (*Delvendahl PNAS; before: 432*)

KdOG = kOffOG / kOnOG;
kappaOG = OGtotal / KdOG;

(*ATP*)
kOnATP  = 5. * 10^8;
kOffATP = 100 000.;
kOnMgATP  = 1. * 10^7; (*Bollmann Dissertation S. 59; *)
kOffMgATP = 1000.;

KdATP = kOffATP / kOnATP;
kappaATP = ATPtotal / KdATP;
KdMgATP = kOffMgATP / kOnMgATP;

(*DM*)
If[MgT == 0.,
 kOnDM = 1.98 * 10^7; (*Faas Plos Biol, 2007*)
kOffDM = 0.14;
 ,
 kOnDM = 2.9 * 10^7; (*Faas Biophys J, 2005*)
kOffDM = 0.19;
]

(*Mg binding constants for DMn, DMf, DMs*)
kOnMg = 1.3 * 10^5; (*all values for Mg are from Faas et al., 2005*)
kOffMg = 0.2;

(*Ca binding constants for PP*)
kOnPP2 = kOnPP1 = kOnDM;
kOffPP2 = 3.6 * 10 ^ 3;
If[MgT == 0.,
 kOffPP1 = 7. * 10 ^ 4;,
 kOffPP1 = 6.9 * 10 ^ 4; ]

(*Mg binding constants for PP1,PP2*)
kOffMgPP = 3. * 10 ^ 2; (*for PP1,PP2*)
konMgPP = kOnMg; (*koMgPP not used in below diff. eq., only kOnMg*)

(*Equilibrium constants (not complete)*)
KdDM = kOffDM / kOnDM;
KdPP1 = kOffPP1 / kOnPP1;
KdMg = kOffMg / kOnMg;
kappaDM = DMT / KdDM;
```

```
(*Endogenous buffer*)
kOnMB  = 5 * 10^8;(*Delvendahl, PNAS, 2015*)
kOffMB = 16 000;
KdMB = kOffMB / kOnMB;

TRest = 1000.;


(*---------------------------------  Loop  ----------
 ---------------------------------------*)
(*---------------------------------  Loop  ----------
 ---------------------------------------*)
(*---------------------------------  Loop  ----------
 ---------------------------------------*)


For[aCount = 1, aCount ≤ aNumberDMN05, aCount += 1,
  a = 10 ^ (Log10[aStartDMN05] +
      (aCount - 1) * (Log10[aEndDMN05] - Log10[aStartDMN05]) / (aNumberDMN05 - 1));

  (*------------------------- Resting
    Equations ------------------------------------------*)

  (*Dye*)
  OGRest := {

    OG[0] == OGtotal,
    CaOG[0] == 0,

    OG'[tt] == -kOnOG * CaRest * OG[tt] + kOffOG * CaOG[tt],
    CaOG'[tt] == kOnOG * CaRest * OG[tt] - kOffOG * CaOG[tt]
   }
  ;

  (*ATP*)
  ATPRest := {

    ATP[0] == ATPtotal,
    CaATP[0] == 0,
    MgATP[0] == 0,

    ATP'[tt] == -kOnATP * CaRest * ATP[tt] + kOffATP * CaATP[tt] -
      kOnMgATP * Mg[tt] * ATP[tt] + kOffMgATP * MgATP[tt],
    CaATP'[tt] == kOnATP * CaRest * ATP[tt] - kOffATP * CaATP[tt],
    MgATP'[tt] == kOnMgATP * Mg[tt] * ATP[tt] - kOffMgATP * MgATP[tt]

   }
```

```
;

(*DM nitrophene*)
DMnRest := {

   DMn[0] == (1 - a) * DMT,
   CaDMn[0] == 0.,
   MgDMn[0] == 0.,

   DMn'[tt] == -kOnDM * CaRest * DMn[tt] +
     kOffDM * CaDMn[tt] - kOnMg * Mg[tt] * DMn[tt] + kOffMg * MgDMn[tt],
   CaDMn'[tt] == kOnDM * CaRest * DMn[tt] - kOffDM * CaDMn[tt],
   MgDMn'[tt] == kOnMg * Mg[tt] * DMn[tt] - kOffMg * MgDMn[tt]
  }

;
DMfRest := {

   DMf[0] == a * af * DMT,
   CaDMf[0] == 0.,
   MgDMf[0] == 0.,

   DMf'[tt] == -kOnDM * CaRest * DMf[tt] +
     kOffDM * CaDMf[tt] - kOnMg * Mg[tt] * DMf[tt] + kOffMg * MgDMf[tt],
   CaDMf'[tt] == kOnDM * CaRest * DMf[tt] - kOffDM * CaDMf[tt],
   MgDMf'[tt] == kOnMg * Mg[tt] * DMf[tt] - kOffMg * MgDMf[tt]
  }

;
DMsRest := {

   DMs[0] == a * (1 - af) * DMT,
   CaDMs[0] == 0.,
   MgDMs[0] == 0.,

   DMs'[tt] == -kOnDM * CaRest * DMs[tt] +
     kOffDM * CaDMs[tt] - kOnMg * Mg[tt] * DMs[tt] + kOffMg * MgDMs[tt],
   CaDMs'[tt] == kOnDM * CaRest * DMs[tt] - kOffDM * CaDMs[tt],
   MgDMs'[tt] == kOnMg * Mg[tt] * DMs[tt] - kOffMg * MgDMs[tt]
  }
;

(*Endogeneous buffer*)
MBRest := {

   MB[0] == MBtotal,
```

```
   CaMB[0] == 0,

   MB'[tt] == -kOnMB * CaRest * MB[tt] + kOffMB * CaMB[tt],
   CaMB'[tt] == kOnMB * CaRest * MB[tt] - kOffMB * CaMB[tt]
  }

;

(*Free Mg*)
MgfRest := {
   Mg[0] == MgT,

   Mg'[tt] ==
    -kOnMgATP * Mg[tt] * ATP[tt] + kOffMgATP * MgATP[tt]
     - kOnMg * Mg[tt] * DMn[tt] + kOffMg * MgDMn[tt]
     - kOnMg * Mg[tt] * DMf[tt] + kOffMg * MgDMf[tt]
     - kOnMg * Mg[tt] * DMs[tt] + kOffMg * MgDMs[tt]
  }

;
EqRest := {ATPRest, MgfRest, OGRest, DMnRest, DMfRest, DMsRest, MBRest};
VarsRest := {ATP, Mg, CaATP, MgATP, CaOG, OG, DMn,
   CaDMn, MgDMn, DMf, CaDMf, MgDMf, DMs, CaDMs, MgDMs, MB, CaMB}
;
solr := NDSolve[EqRest, VarsRest, {tt, 0, TRest}]
;

Ca0 = CaRest;
Mg0 = Extract[Mg[TRest] /. solr, 1];
ATP0 = Extract[ATP[TRest] /. solr, 1];
CaATP0 = Extract[CaATP[TRest] /. solr, 1];
MgATP0 = Extract[MgATP[TRest] /. solr, 1];
OG0 = Extract[OG[TRest] /. solr, 1];
CaOG0 = Extract[CaOG[TRest] /. solr, 1];
DMn0 = Extract[DMn[TRest] /. solr, 1];
CaDMn0 = Extract[CaDMn[TRest] /. solr, 1];
MgDMn0 = Extract[MgDMn[TRest] /. solr, 1];
DMf0 = Extract[DMf[TRest] /. solr, 1];
CaDMf0 = Extract[CaDMf[TRest] /. solr, 1];
MgDMf0 = Extract[MgDMf[TRest] /. solr, 1];
DMs0 = Extract[DMs[TRest] /. solr, 1];
CaDMs0 = Extract[CaDMs[TRest] /. solr, 1];
MgDMs0 = Extract[MgDMs[TRest] /. solr, 1];
MB0 = Extract[MB[TRest] /. solr, 1];
CaMB0 = Extract[CaMB[TRest] /. solr, 1];
```

```
ClearAll[EqRest, VarsRest];


(*------------------------- Flash
  Equations ------------------------------------------*)

(*Dye*)
BufferOG := {

  OG[0] == OG0,
  CaOG[0] == CaOG0,

  OG'[tt] == -kOnOG * Ca[tt] * OG[tt] + kOffOG * CaOG[tt],
  CaOG'[tt] == kOnOG * Ca[tt] * OG[tt] - kOffOG * CaOG[tt]}
;

(*ATP*)
BufferATP := {

  ATP[0] == ATP0,
  CaATP[0] == CaATP0,
  MgATP[0] == MgATP0,

  ATP'[tt] == -kOnATP * Ca[tt] * ATP[tt] + kOffATP * CaATP[tt] -
    kOnMgATP * Mg[tt] * ATP[tt] + kOffMgATP * MgATP[tt],
  CaATP'[tt] == kOnATP * Ca[tt] * ATP[tt] - kOffATP * CaATP[tt],
  MgATP'[tt] == kOnMgATP * Mg[tt] * ATP[tt] - kOffMgATP * MgATP[tt]
 }
;

(*fast (tauf) and slow (taus) time constants for uncageing;
Faas et al., 2005,2007*)
If[MgT == 0.,
  tauf = 15.2 * 10^-6;  (*Faas, 2007*)
  taus = 1.9 * 10^-3;
  ,
  tauf = 15. * 10^-6;  (*Faas,2005*)
  taus = 2. * 10^-3; ]

;

(*The differential equations*)
(*non uncaging fraction of DMn*)
BufferDMn := {

  DMn[0] == DMn0,
  CaDMn[0] == CaDMn0,
```

```
    MgDMn[0] == MgDMn0,

    DMn'[tt] == -kOnDM * Ca[tt] * DMn[tt] +
      kOffDM * CaDMn[tt] - kOnMg * Mg[tt] * DMn[tt] + kOffMg * MgDMn[tt],
    CaDMn'[tt] == kOnDM * Ca[tt] * DMn[tt] - kOffDM * CaDMn[tt],
    MgDMn'[tt] == kOnMg * Mg[tt] * DMn[tt] - kOffMg * MgDMn[tt]
  }
;

(*fast uncaging fraction of DMn*)
BufferDMf := {

    DMf[0] == DMf0,
    CaDMf[0] == CaDMf0,
    MgDMf[0] == MgDMf0,

    DMf'[tt] == -kOnDM * Ca[tt] * DMf[tt] + kOffDM * CaDMf[tt] - kOnMg * Mg[tt] *
       DMf[tt] + kOffMg * MgDMf[tt] - 1 / tauf * DMf[tt] * UnitStep[tt - tflash],
    CaDMf'[tt] == kOnDM * Ca[tt] * DMf[tt] - kOffDM * CaDMf[tt] -
      1 / tauf * CaDMf[tt] * UnitStep[tt - tflash],
    MgDMf'[tt] == kOnMg * Mg[tt] * DMf[tt] - kOffMg * MgDMf[tt] -
      1 / tauf * MgDMf[tt] * UnitStep[tt - tflash]
  }

;
(*slow uncaging fraction of DMn*)
BufferDMs := {

    DMs[0] == DMs0,
    CaDMs[0] == CaDMs0,
    MgDMs[0] == MgDMs0,

    DMs'[tt] == -kOnDM * Ca[tt] * DMs[tt] + kOffDM * CaDMs[tt] - kOnMg * Mg[tt] *
       DMs[tt] + kOffMg * MgDMs[tt] - 1 / taus * DMs[tt] * UnitStep[tt - tflash],
    CaDMs'[tt] == kOnDM * Ca[tt] * DMs[tt] - kOffDM * CaDMs[tt] -
      1 / taus * CaDMs[tt] * UnitStep[tt - tflash],
    MgDMs'[tt] == kOnMg * Mg[tt] * DMs[tt] - kOffMg * MgDMs[tt] -
      1 / taus * MgDMs[tt] * UnitStep[tt - tflash]
  }
;

(*Photoproducts*)
(*PP2: comes from DMf,DMs and MgDMf,MgDMs; but also binds Ca*)
BufferPP2 := {

    PP2[0] == 0,
    CaPP2[0] == 0,
```

```
  MgPP2[0] == 0,

  PP2'[tt] == -kOnPP2 * Ca[tt] * PP2[tt] + kOffPP2 * CaPP2[tt] -
    kOnMg * Mg[tt] * PP2[tt] + kOffMgPP * MgPP2[tt] + 2 * (1 / tauf * DMf[tt] *
        UnitStep[tt - tflash] + 1 / taus * DMs[tt] * UnitStep[tt - tflash])
    + 1 / tauf * MgDMf[tt] * UnitStep[tt - tflash] +
    1 / taus * MgDMs[tt] * UnitStep[tt - tflash],

  CaPP2'[tt] == kOnPP2 * Ca[tt] * PP2[tt] - kOffPP2 * CaPP2[tt],

  MgPP2'[tt] == kOnMg * Mg[tt] * PP2[tt] - kOffMgPP * MgPP2[tt] + 1 / tauf *
     MgDMf[tt] * UnitStep[tt - tflash] + 1 / taus * MgDMs[tt] * UnitStep[tt - tflash]
 }
;

(*PP1: Comes from CaDMf,CaDMs and binds Ca and Mg*)
BufferPP1 := {

  PP1[0] == 0,
  CaPP1[0] == 0,
  MgPP1[0] == 0,

  PP1'[tt] == -kOnPP1 * Ca[tt] * PP1[tt] +
    kOffPP1 * CaPP1[tt] - kOnMg * Mg[tt] * PP1[tt] + kOffMgPP * MgPP1[tt]
    + 1 / tauf * CaDMf[tt] * UnitStep[tt - tflash] +
    1 / taus * CaDMs[tt] * UnitStep[tt - tflash],

  CaPP1'[tt] == kOnPP1 * Ca[tt] * PP1[tt] -
    kOffPP1 * CaPP1[tt] + 1 / tauf * CaDMf[tt] * UnitStep[tt - tflash] +
    1 / taus * CaDMs[tt] * UnitStep[tt - tflash],

  MgPP1'[tt] == kOnMg * Mg[tt] * PP1[tt] - kOffMgPP * MgPP1[tt]

 }
;

(*Endogeneous Buffer*)
BufferMB := {

  MB[0] == MB0,
  CaMB[0] == CaMB0,

  MB'[tt] == -kOnMB * Ca[tt] * MB[tt] + kOffMB * CaMB[tt],
  CaMB'[tt] == kOnMB * Ca[tt] * MB[tt] - kOffMB * CaMB[tt]}

;
```

```
(*Clear[Eqns,Vars,sol]*)

  (*Free Ca*)
  FreeCa := {
    Ca[0] == Ca0,

    Ca'[tt] == -γ * (Ca[tt] - CaRest)
      (*DMn*)
      - kOnPP1 * Ca[tt] * PP1[tt] + kOffPP1 * CaPP1[tt]
      - kOnPP2 * Ca[tt] * PP2[tt] + kOffPP2 * CaPP2[tt]
      - kOnDM * Ca[tt] * DMn[tt] + kOffDM * CaDMn[tt]
      - kOnDM * Ca[tt] * DMf[tt] + kOffDM * CaDMf[tt]
      - kOnDM * Ca[tt] * DMs[tt] + kOffDM * CaDMs[tt]
      (*buffers*)
      - kOnATP * Ca[tt] * ATP[tt] + kOffATP * CaATP[tt]
      - kOnMB * Ca[tt] * MB[tt] + kOffMB * CaMB[tt]
      (*dye*)
      - kOnOG * Ca[tt] * OG[tt] + kOffOG * CaOG[tt]
    }
  ;

  (*Free Mg*)
  FreeMg := {
    Mg[0] == Mg0,

    Mg'[tt] ==
     -kOnMgATP * Mg[tt] * ATP[tt] + kOffMgATP * MgATP[tt]
      - kOnMg * Mg[tt] * DMn[tt] + kOffMg * MgDMn[tt]
      - kOnMg * Mg[tt] * DMf[tt] + kOffMg * MgDMf[tt]
      - kOnMg * Mg[tt] * DMs[tt] + kOffMg * MgDMs[tt]
      - kOnMg * Mg[tt] * PP2[tt] + kOffMgPP * MgPP2[tt]
      - kOnMg * Mg[tt] * PP1[tt] + kOffMgPP * MgPP1[tt]
   }

  ;

  Eqns := {BufferDMn, BufferDMf, BufferDMs, BufferATP,
    BufferPP1, BufferPP2, FreeCa, FreeMg, BufferMB, BufferOG}
  ;
  Vars := {ATP, CaATP, MgATP, Ca, Mg, CaDMn, DMn, CaDMf, DMf, CaDMs,
    DMs, CaPP1, PP1, CaPP2, PP2, MgPP2, MgPP1, MB, CaMB, OG, CaOG}
  ;
  sol := NDSolve[Eqns, Vars, {tt, 0., TimeWindow}
    (*,Method→{"EquationSimplification"->"Solve"}*)]
  ;
  CafP = Extract[Ca[TimeWindow] /. sol, 1];
  CafOG = KdOG * CaOG[tt] / OG[tt];
```

```
  AppendTo[CaListReal, Evaluate[{Ca[tt]} /. sol]];
  AppendTo[CaListDye, Evaluate[{CafOG} /. sol]];

 ];
```

2 mM DMN

general parameters

```
In[•]:= TimeWindow = 0.006; (*End of simulation*)
tflash = 0.0;  (*Time of flash*)
PlStart = 0.; (*Plot start*)

af = 0.67; (*fast uncaging fraction; Faas et al*)

(*Select dye*)
OGB1 = 0;
OGB5N = 1;
OGB6F = 0;
Fluo5F = 0;

CaRest = 227. * 10^-9; (*Free pre-flash rersting Ca;
equilibrates with all buffers and DM*)
MgT = 0.5 * 10^-3; (*total Mg in pipette*)
γ = 0.; (*Pump rate*)

(*Concentrations of dye, buffers, DM*)
OGtotal  = 200. * 10^-6 ;
ATPtotal  = 5. * 10^-3 ;
MBtotal  = 480. * 10^-6 ;(*(*Delvendahl, PNAS, 2015*)*)

DMT  = 2. * 10^-3;  (*total concentration of DMn *)
(*uncaging efficiency*)
aStartDMN2 = 0.15;
aEndDMN2 = 0.55;
```

definitions and loop

```
In[•]:= (*Dye*)
If[OGB1 == 1,
 kOnOG  = 4.3 * 10^8;
kOffOG  = 103.;]
```

```
If[OGB5N == 1,
 kOnOG  = 2.5 * 10^8;
kOffOG = 6000.;]

If[OGB6F == 1,
 kOnOG  = 3. * 10^8;
kOffOG = 900.;]

If[Fluo5F == 1,
 kOnOG  = 3. * 10^8;
kOffOG = 249.;] (*Delvendahl PNAS; before: 432*)

KdOG = kOffOG / kOnOG;
kappaOG = OGtotal / KdOG;

(*ATP*)
kOnATP  = 5. * 10^8;
kOffATP = 100 000.;
kOnMgATP  = 1. * 10^7; (*Bollmann Dissertation S. 59; *)
kOffMgATP = 1000.;

KdATP = kOffATP / kOnATP;
kappaATP = ATPtotal / KdATP;
KdMgATP = kOffMgATP / kOnMgATP;

(*DM*)
If[MgT == 0.,
 kOnDM = 1.98 * 10^7; (*Faas Plos Biol, 2007*)
kOffDM = 0.14;
 ,
 kOnDM = 2.9 * 10^7; (*Faas Biophys J, 2005*)
kOffDM = 0.19;
]

(*Mg binding constants for DMn, DMf, DMs*)
kOnMg = 1.3 * 10^5; (*all values for Mg are from Faas et al., 2005*)
kOffMg = 0.2;

(*Ca binding constants for PP*)
kOnPP2 = kOnPP1 = kOnDM;
kOffPP2 = 3.6 * 10^3;
If[MgT == 0.,
 kOffPP1 = 7. * 10^4;,
 kOffPP1 = 6.9 * 10^4; ]

(*Mg binding constants for PP1,PP2*)
```

```
kOffMgPP = 3. * 10^2; (*for PP1,PP2*)
konMgPP = kOnMg; (*koMgPP not used in below diff. eq., only kOnMg*)

(*Equilibrium constants (not complete)*)
KdDM = kOffDM / kOnDM;
KdPP1 = kOffPP1 / kOnPP1;
KdMg = kOffMg / kOnMg;
kappaDM = DMT / KdDM;

 (*Endogenous buffer*)
kOnMB  = 5 * 10^8;(*Delvendahl, PNAS, 2015*)
kOffMB = 16000;
KdMB = kOffMB / kOnMB;

TRest = 1000.;


(*---------------------------------  Loop  ----------
 --------------------------------------*)
(*---------------------------------  Loop  ----------
 --------------------------------------*)
(*---------------------------------  Loop  ----------
 --------------------------------------*)

For[aCount = 1, aCount ≤ aNumberDMN2, aCount += 1,
  a = 10^(Log10[aStartDMN2] +
      (aCount - 1) * (Log10[aEndDMN2] - Log10[aStartDMN2]) / (aNumberDMN2 - 1));

  (*------------------------ Resting
    Equations -----------------------------------------*)

  (*Dye*)
  OGRest := {

    OG[0] == OGtotal,
    CaOG[0] == 0,

    OG'[tt] == -kOnOG * CaRest * OG[tt] + kOffOG * CaOG[tt],
    CaOG'[tt] == kOnOG * CaRest * OG[tt] - kOffOG * CaOG[tt]
   }
  ;

  (*ATP*)
  ATPRest := {

    ATP[0] == ATPtotal,
    CaATP[0] == 0,
```

```
    MgATP[0] == 0,

    ATP'[tt] == -kOnATP * CaRest * ATP[tt] + kOffATP * CaATP[tt] -
      kOnMgATP * Mg[tt] * ATP[tt] + kOffMgATP * MgATP[tt],
    CaATP'[tt] == kOnATP * CaRest * ATP[tt] - kOffATP * CaATP[tt],
    MgATP'[tt] == kOnMgATP * Mg[tt] * ATP[tt] - kOffMgATP * MgATP[tt]

   }

  ;

  (*DM nitrophene*)
  DMnRest := {

    DMn[0] == (1 - a) * DMT,
    CaDMn[0] == 0.,
    MgDMn[0] == 0.,

    DMn'[tt] == -kOnDM * CaRest * DMn[tt] +
      kOffDM * CaDMn[tt] - kOnMg * Mg[tt] * DMn[tt] + kOffMg * MgDMn[tt],
    CaDMn'[tt] == kOnDM * CaRest * DMn[tt] - kOffDM * CaDMn[tt],
    MgDMn'[tt] == kOnMg * Mg[tt] * DMn[tt] - kOffMg * MgDMn[tt]
   }

  ;
  DMfRest := {

    DMf[0] == a * af * DMT,
    CaDMf[0] == 0.,
    MgDMf[0] == 0.,

    DMf'[tt] == -kOnDM * CaRest * DMf[tt] +
      kOffDM * CaDMf[tt] - kOnMg * Mg[tt] * DMf[tt] + kOffMg * MgDMf[tt],
    CaDMf'[tt] == kOnDM * CaRest * DMf[tt] - kOffDM * CaDMf[tt],
    MgDMf'[tt] == kOnMg * Mg[tt] * DMf[tt] - kOffMg * MgDMf[tt]
   }

  ;
  DMsRest := {

    DMs[0] == a * (1 - af) * DMT,
    CaDMs[0] == 0.,
    MgDMs[0] == 0.,

    DMs'[tt] == -kOnDM * CaRest * DMs[tt] +
      kOffDM * CaDMs[tt] - kOnMg * Mg[tt] * DMs[tt] + kOffMg * MgDMs[tt],
    CaDMs'[tt] == kOnDM * CaRest * DMs[tt] - kOffDM * CaDMs[tt],
```

```
  MgDMs'[tt] == kOnMg * Mg[tt] * DMs[tt] - kOffMg * MgDMs[tt]
 }
;

(*Endogeneous buffer*)
MBRest := {

  MB[0] == MBtotal,
  CaMB[0] == 0,

  MB'[tt] == -kOnMB * CaRest * MB[tt] + kOffMB * CaMB[tt],
  CaMB'[tt] == kOnMB * CaRest * MB[tt] - kOffMB * CaMB[tt]
 }

;

(*Free Mg*)
MgfRest := {
  Mg[0] == MgT,

  Mg'[tt] ==
   -kOnMgATP * Mg[tt] * ATP[tt] + kOffMgATP * MgATP[tt]
    - kOnMg * Mg[tt] * DMn[tt] + kOffMg * MgDMn[tt]
    - kOnMg * Mg[tt] * DMf[tt] + kOffMg * MgDMf[tt]
    - kOnMg * Mg[tt] * DMs[tt] + kOffMg * MgDMs[tt]
 }

;
EqRest := {ATPRest, MgfRest, OGRest, DMnRest, DMfRest, DMsRest, MBRest};
VarsRest := {ATP, Mg, CaATP, MgATP, CaOG, OG, DMn,
  CaDMn, MgDMn, DMf, CaDMf, MgDMf, DMs, CaDMs, MgDMs, MB, CaMB}
;
solr := NDSolve[EqRest, VarsRest, {tt, 0, TRest}]
;

Ca0 = CaRest;
Mg0 = Extract[Mg[TRest] /. solr, 1];
ATP0 = Extract[ATP[TRest] /. solr, 1];
CaATP0 = Extract[CaATP[TRest] /. solr, 1];
MgATP0 = Extract[MgATP[TRest] /. solr, 1];
OG0 = Extract[OG[TRest] /. solr, 1];
CaOG0 = Extract[CaOG[TRest] /. solr, 1];
DMn0 = Extract[DMn[TRest] /. solr, 1];
CaDMn0 = Extract[CaDMn[TRest] /. solr, 1];
MgDMn0 = Extract[MgDMn[TRest] /. solr, 1];
DMf0 = Extract[DMf[TRest] /. solr, 1];
CaDMf0 = Extract[CaDMf[TRest] /. solr, 1];
```

```
MgDMf0 = Extract[MgDMf[TRest] /. solr, 1];
DMs0 = Extract[DMs[TRest] /. solr, 1];
CaDMs0 = Extract[CaDMs[TRest] /. solr, 1];
MgDMs0 = Extract[MgDMs[TRest] /. solr, 1];
MB0 = Extract[MB[TRest] /. solr, 1];
CaMB0 = Extract[CaMB[TRest] /. solr, 1];


ClearAll[EqRest, VarsRest];


(*------------------------- Flash
  Equations ------------------------------------------*)

(*Dye*)
BufferOG := {

  OG[0] == OG0,
  CaOG[0] == CaOG0,

  OG'[tt] == -kOnOG * Ca[tt] * OG[tt] + kOffOG * CaOG[tt],
  CaOG'[tt] == kOnOG * Ca[tt] * OG[tt] - kOffOG * CaOG[tt]}
;

(*ATP*)
BufferATP := {

  ATP[0] == ATP0,
  CaATP[0] == CaATP0,
  MgATP[0] == MgATP0,

  ATP'[tt] == -kOnATP * Ca[tt] * ATP[tt] + kOffATP * CaATP[tt] -
    kOnMgATP * Mg[tt] * ATP[tt] + kOffMgATP * MgATP[tt],
  CaATP'[tt] == kOnATP * Ca[tt] * ATP[tt] - kOffATP * CaATP[tt],
  MgATP'[tt] == kOnMgATP * Mg[tt] * ATP[tt] - kOffMgATP * MgATP[tt]
 }
;

(*fast (tauf) and slow (taus) time constants for uncageing;
Faas et al., 2005,2007*)
If[MgT == 0.,
  tauf = 15.2 * 10^-6;  (*Faas, 2007*)
  taus = 1.9 * 10^-3;
  ,
  tauf = 15. * 10^-6;  (*Faas,2005*)
  taus = 2. * 10^-3; ]
```

```
;

(*The differential equations*)
(*non uncaging fraction of DMn*)
BufferDMn := {

  DMn[0] == DMn0,
  CaDMn[0] == CaDMn0,
  MgDMn[0] == MgDMn0,

  DMn'[tt] == -kOnDM * Ca[tt] * DMn[tt] +
    kOffDM * CaDMn[tt] - kOnMg * Mg[tt] * DMn[tt] + kOffMg * MgDMn[tt],
  CaDMn'[tt] == kOnDM * Ca[tt] * DMn[tt] - kOffDM * CaDMn[tt],
  MgDMn'[tt] == kOnMg * Mg[tt] * DMn[tt] - kOffMg * MgDMn[tt]
 }
;

(*fast uncaging fraction of DMn*)
BufferDMf := {

  DMf[0] == DMf0,
  CaDMf[0] == CaDMf0,
  MgDMf[0] == MgDMf0,

  DMf'[tt] == -kOnDM * Ca[tt] * DMf[tt] + kOffDM * CaDMf[tt] - kOnMg * Mg[tt] *
     DMf[tt] + kOffMg * MgDMf[tt] - 1 / tauf * DMf[tt] * UnitStep[tt - tflash],
  CaDMf'[tt] == kOnDM * Ca[tt] * DMf[tt] - kOffDM * CaDMf[tt] -
    1 / tauf * CaDMf[tt] * UnitStep[tt - tflash],
  MgDMf'[tt] == kOnMg * Mg[tt] * DMf[tt] - kOffMg * MgDMf[tt] -
    1 / tauf * MgDMf[tt] * UnitStep[tt - tflash]
 }

;
(*slow uncaging fraction of DMn*)
BufferDMs := {

  DMs[0] == DMs0,
  CaDMs[0] == CaDMs0,
  MgDMs[0] == MgDMs0,

  DMs'[tt] == -kOnDM * Ca[tt] * DMs[tt] + kOffDM * CaDMs[tt] - kOnMg * Mg[tt] *
     DMs[tt] + kOffMg * MgDMs[tt] - 1 / taus * DMs[tt] * UnitStep[tt - tflash],
  CaDMs'[tt] == kOnDM * Ca[tt] * DMs[tt] - kOffDM * CaDMs[tt] -
    1 / taus * CaDMs[tt] * UnitStep[tt - tflash],
  MgDMs'[tt] == kOnMg * Mg[tt] * DMs[tt] - kOffMg * MgDMs[tt] -
    1 / taus * MgDMs[tt] * UnitStep[tt - tflash]
 }
```

```
;

(*Photoproducts*)
(*PP2: comes from DMf,DMs and MgDMf,MgDMs; but also binds Ca*)
BufferPP2 := {

  PP2[0] == 0,
  CaPP2[0] == 0,
  MgPP2[0] == 0,

  PP2'[tt] == -kOnPP2 * Ca[tt] * PP2[tt] + kOffPP2 * CaPP2[tt] -
    kOnMg * Mg[tt] * PP2[tt] + kOffMgPP * MgPP2[tt] + 2 * (1 / tauf * DMf[tt] *
        UnitStep[tt - tflash] + 1 / taus * DMs[tt] * UnitStep[tt - tflash])
    + 1 / tauf * MgDMf[tt] * UnitStep[tt - tflash] +
    1 / taus * MgDMs[tt] * UnitStep[tt - tflash],

  CaPP2'[tt] == kOnPP2 * Ca[tt] * PP2[tt] - kOffPP2 * CaPP2[tt],

  MgPP2'[tt] == kOnMg * Mg[tt] * PP2[tt] - kOffMgPP * MgPP2[tt] + 1 / tauf *
     MgDMf[tt] * UnitStep[tt - tflash] + 1 / taus * MgDMs[tt] * UnitStep[tt - tflash]
 }
;

(*PP1: Comes from CaDMf,CaDMs and binds Ca and Mg*)
BufferPP1 := {

  PP1[0] == 0,
  CaPP1[0] == 0,
  MgPP1[0] == 0,

  PP1'[tt] == -kOnPP1 * Ca[tt] * PP1[tt] +
    kOffPP1 * CaPP1[tt] - kOnMg * Mg[tt] * PP1[tt] + kOffMgPP * MgPP1[tt]
    + 1 / tauf * CaDMf[tt] * UnitStep[tt - tflash] +
    1 / taus * CaDMs[tt] * UnitStep[tt - tflash],

  CaPP1'[tt] == kOnPP1 * Ca[tt] * PP1[tt] -
    kOffPP1 * CaPP1[tt] + 1 / tauf * CaDMf[tt] * UnitStep[tt - tflash] +
    1 / taus * CaDMs[tt] * UnitStep[tt - tflash],

  MgPP1'[tt] == kOnMg * Mg[tt] * PP1[tt] - kOffMgPP * MgPP1[tt]

 }
;

(*Endogeneous Buffer*)
BufferMB := {
```

```
      MB[0] == MB0,
      CaMB[0] == CaMB0,

      MB'[tt] == -kOnMB * Ca[tt] * MB[tt] + kOffMB * CaMB[tt],
      CaMB'[tt] == kOnMB * Ca[tt] * MB[tt] - kOffMB * CaMB[tt]}

    ;

  (*Clear[Eqns,Vars,sol]*)

    (*Free Ca*)
    FreeCa := {
      Ca[0] == Ca0,

      Ca'[tt] == -γ * (Ca[tt] - CaRest)
        (*DMn*)
        - kOnPP1 * Ca[tt] * PP1[tt] + kOffPP1 * CaPP1[tt]
        - kOnPP2 * Ca[tt] * PP2[tt] + kOffPP2 * CaPP2[tt]
        - kOnDM * Ca[tt] * DMn[tt] + kOffDM * CaDMn[tt]
        - kOnDM * Ca[tt] * DMf[tt] + kOffDM * CaDMf[tt]
        - kOnDM * Ca[tt] * DMs[tt] + kOffDM * CaDMs[tt]
        (*buffers*)
        - kOnATP * Ca[tt] * ATP[tt] + kOffATP * CaATP[tt]
        - kOnMB * Ca[tt] * MB[tt] + kOffMB * CaMB[tt]
        (*dye*)
        - kOnOG * Ca[tt] * OG[tt] + kOffOG * CaOG[tt]
      }
    ;

    (*Free Mg*)
    FreeMg := {
      Mg[0] == Mg0,

      Mg'[tt] ==
       -kOnMgATP * Mg[tt] * ATP[tt] + kOffMgATP * MgATP[tt]
        - kOnMg * Mg[tt] * DMn[tt] + kOffMg * MgDMn[tt]
        - kOnMg * Mg[tt] * DMf[tt] + kOffMg * MgDMf[tt]
        - kOnMg * Mg[tt] * DMs[tt] + kOffMg * MgDMs[tt]
        - kOnMg * Mg[tt] * PP2[tt] + kOffMgPP * MgPP2[tt]
        - kOnMg * Mg[tt] * PP1[tt] + kOffMgPP * MgPP1[tt]
     }

    ;

    Eqns := {BufferDMn, BufferDMf, BufferDMs, BufferATP,
      BufferPP1, BufferPP2, FreeCa, FreeMg, BufferMB, BufferOG}
    ;
```

```
Vars := {ATP, CaATP, MgATP, Ca, Mg, CaDMn, DMn, CaDMf, DMf, CaDMs,
  DMs, CaPP1, PP1, CaPP2, PP2, MgPP2, MgPP1, MB, CaMB, OG, CaOG}
;
sol := NDSolve[Eqns, Vars, {tt, 0., TimeWindow}
  (*,Method→{"EquationSimplification"->"Solve"}*)]
;
CafP = Extract[Ca[TimeWindow] /. sol, 1];
CafOG = KdOG * CaOG[tt] / OG[tt];

AppendTo[CaListReal, Evaluate[{Ca[tt]} /. sol]];
AppendTo[CaListDye, Evaluate[{CafOG} /. sol]];

];
```

10 mM DMN

general parameters

```
In[]:= TimeWindow = 0.006; (*End of simulation*)
tflash = 0.0;  (*Time of flash*)
PlStart = 0.; (*Plot start*)

af = 0.67; (*fast uncaging fraction; Faas et al*)

(*Select dye*)
OGB1 = 0;
OGB5N = 1;
OGB6F = 0;
Fluo5F = 0;

CaRest = 227. * 10^-9; (*Free pre-flash rersting Ca;
equilibrates with all buffers and DM*)
MgT = 0.5 * 10^-3; (*total Mg in pipette*)
γ = 0.;  (*Pump rate*)

(*Concentrations of dye, buffers, DM*)
OGtotal  = 200. * 10^-6 ;
ATPtotal  = 5. * 10^-3 ;
MBtotal  = 480. * 10^-6 ;(*(*Delvendahl, PNAS, 2015*)*)

DMT  =  10. * 10^-3;  (*total concentration of DMn *)
(*uncaging efficiency*)
aStartDMN10 = 0.14;
aEndDMN10 = 0.25;
```

definitions and loop

```
In[*]:= (*Dye*)
If[OGB1 == 1,
 kOnOG  = 4.3 * 10^8;
kOffOG = 103.;]

If[OGB5N == 1,
 kOnOG  = 2.5 * 10^8;
kOffOG = 6000.;]

If[OGB6F == 1,
 kOnOG  = 3. * 10^8;
kOffOG = 900.;]

If[Fluo5F == 1,
 kOnOG  = 3. * 10^8;
kOffOG = 249.;] (*Delvendahl PNAS; before: 432*)

KdOG = kOffOG / kOnOG;
kappaOG = OGtotal / KdOG;

(*ATP*)
kOnATP  = 5. * 10^8;
kOffATP = 100 000.;
kOnMgATP  = 1. * 10^7; (*Bollmann Dissertation S. 59; *)
kOffMgATP = 1000.;

KdATP = kOffATP / kOnATP;
kappaATP = ATPtotal / KdATP;
KdMgATP = kOffMgATP / kOnMgATP;

(*DM*)
If[MgT == 0.,
 kOnDM = 1.98 * 10^7; (*Faas Plos Biol, 2007*)
kOffDM = 0.14;
 ,
 kOnDM = 2.9 * 10^7; (*Faas Biophys J, 2005*)
kOffDM = 0.19;
]

(*Mg binding constants for DMn, DMf, DMs*)
kOnMg = 1.3 * 10^5; (*all values for Mg are from Faas et al., 2005*)
kOffMg = 0.2;
```

```
(*Ca binding constants for PP*)
kOnPP2 = kOnPP1 = kOnDM;
kOffPP2 = 3.6 * 10^3;
If[MgT == 0.,
 kOffPP1 = 7. * 10^4;,
 kOffPP1 = 6.9 * 10^4; ]

(*Mg binding constants for PP1,PP2*)
kOffMgPP = 3. * 10^2; (*for PP1,PP2*)
konMgPP = kOnMg; (*koMgPP not used in below diff. eq., only kOnMg*)

(*Equilibrium constants (not complete)*)
KdDM = kOffDM / kOnDM;
KdPP1 = kOffPP1 / kOnPP1;
KdMg = kOffMg / kOnMg;
kappaDM = DMT / KdDM;

 (*Endogenous buffer*)
kOnMB  = 5 * 10^8;(*Delvendahl, PNAS, 2015*)
kOffMB = 16000;
KdMB = kOffMB / kOnMB;

TRest = 1000.;


(*---------------------------------  Loop  ----------
 ---------------------------------------*)
(*---------------------------------  Loop  ----------
 ---------------------------------------*)
(*---------------------------------  Loop  ----------
 ---------------------------------------*)

For[aCount = 1, aCount ≤ aNumberDMN10, aCount += 1,
  a = 10^(Log10[aStartDMN10] +
      (aCount - 1) * (Log10[aEndDMN10] - Log10[aStartDMN10]) / (aNumberDMN10 - 1));

  (*------------------------ Resting
    Equations -----------------------------------------*)

  (*Dye*)
  OGRest := {

    OG[0] == OGtotal,
    CaOG[0] == 0,

    OG'[tt] == -kOnOG * CaRest * OG[tt] + kOffOG * CaOG[tt],
    CaOG'[tt] == kOnOG * CaRest * OG[tt] - kOffOG * CaOG[tt]
```

```
 }
;

(*ATP*)
ATPRest := {

  ATP[0] == ATPtotal,
  CaATP[0] == 0,
  MgATP[0] == 0,

  ATP'[tt] == -kOnATP * CaRest * ATP[tt] + kOffATP * CaATP[tt] -
    kOnMgATP * Mg[tt] * ATP[tt] + kOffMgATP * MgATP[tt],
  CaATP'[tt] == kOnATP * CaRest * ATP[tt] - kOffATP * CaATP[tt],
  MgATP'[tt] == kOnMgATP * Mg[tt] * ATP[tt] - kOffMgATP * MgATP[tt]

 }

;

(*DM nitrophene*)
DMnRest := {

  DMn[0] == (1 - a) * DMT,
  CaDMn[0] == 0.,
  MgDMn[0] == 0.,

  DMn'[tt] == -kOnDM * CaRest * DMn[tt] +
    kOffDM * CaDMn[tt] - kOnMg * Mg[tt] * DMn[tt] + kOffMg * MgDMn[tt],
  CaDMn'[tt] == kOnDM * CaRest * DMn[tt] - kOffDM * CaDMn[tt],
  MgDMn'[tt] == kOnMg * Mg[tt] * DMn[tt] - kOffMg * MgDMn[tt]
 }

;
DMfRest := {

  DMf[0] == a * af * DMT,
  CaDMf[0] == 0.,
  MgDMf[0] == 0.,

  DMf'[tt] == -kOnDM * CaRest * DMf[tt] +
    kOffDM * CaDMf[tt] - kOnMg * Mg[tt] * DMf[tt] + kOffMg * MgDMf[tt],
  CaDMf'[tt] == kOnDM * CaRest * DMf[tt] - kOffDM * CaDMf[tt],
  MgDMf'[tt] == kOnMg * Mg[tt] * DMf[tt] - kOffMg * MgDMf[tt]
 }

;
DMsRest := {
```

```
  DMs[0] == a * (1 - af) * DMT,
  CaDMs[0] == 0.,
  MgDMs[0] == 0.,

  DMs'[tt] == -kOnDM * CaRest * DMs[tt] +
    kOffDM * CaDMs[tt] - kOnMg * Mg[tt] * DMs[tt] + kOffMg * MgDMs[tt],
  CaDMs'[tt] == kOnDM * CaRest * DMs[tt] - kOffDM * CaDMs[tt],
  MgDMs'[tt] == kOnMg * Mg[tt] * DMs[tt] - kOffMg * MgDMs[tt]
 }
;

(*Endogeneous buffer*)
MBRest := {

  MB[0] == MBtotal,
  CaMB[0] == 0,

  MB'[tt] == -kOnMB * CaRest * MB[tt] + kOffMB * CaMB[tt],
  CaMB'[tt] == kOnMB * CaRest * MB[tt] - kOffMB * CaMB[tt]
 }

;

(*Free Mg*)
MgfRest := {
  Mg[0] == MgT,

  Mg'[tt] ==
   -kOnMgATP * Mg[tt] * ATP[tt] + kOffMgATP * MgATP[tt]
    - kOnMg * Mg[tt] * DMn[tt] + kOffMg * MgDMn[tt]
    - kOnMg * Mg[tt] * DMf[tt] + kOffMg * MgDMf[tt]
    - kOnMg * Mg[tt] * DMs[tt] + kOffMg * MgDMs[tt]
 }

;
EqRest := {ATPRest, MgfRest, OGRest, DMnRest, DMfRest, DMsRest, MBRest};
VarsRest := {ATP, Mg, CaATP, MgATP, CaOG, OG, DMn,
  CaDMn, MgDMn, DMf, CaDMf, MgDMf, DMs, CaDMs, MgDMs, MB, CaMB}
;
solr := NDSolve[EqRest, VarsRest, {tt, 0, TRest}]
;

Ca0 = CaRest;
Mg0 = Extract[Mg[TRest] /. solr, 1];
ATP0 = Extract[ATP[TRest] /. solr, 1];
CaATP0 = Extract[CaATP[TRest] /. solr, 1];
```

```
MgATP0 = Extract[MgATP[TRest] /. solr, 1];
OG0 = Extract[OG[TRest] /. solr, 1];
CaOG0 = Extract[CaOG[TRest] /. solr, 1];
DMn0 = Extract[DMn[TRest] /. solr, 1];
CaDMn0 = Extract[CaDMn[TRest] /. solr, 1];
MgDMn0 = Extract[MgDMn[TRest] /. solr, 1];
DMf0 = Extract[DMf[TRest] /. solr, 1];
CaDMf0 = Extract[CaDMf[TRest] /. solr, 1];
MgDMf0 = Extract[MgDMf[TRest] /. solr, 1];
DMs0 = Extract[DMs[TRest] /. solr, 1];
CaDMs0 = Extract[CaDMs[TRest] /. solr, 1];
MgDMs0 = Extract[MgDMs[TRest] /. solr, 1];
MB0 = Extract[MB[TRest] /. solr, 1];
CaMB0 = Extract[CaMB[TRest] /. solr, 1];


ClearAll[EqRest, VarsRest];


(*------------------------- Flash
  Equations -------------------------------------------*)

(*Dye*)
BufferOG := {

  OG[0] == OG0,
  CaOG[0] == CaOG0,

  OG'[tt] == -kOnOG * Ca[tt] * OG[tt] + kOffOG * CaOG[tt],
  CaOG'[tt] == kOnOG * Ca[tt] * OG[tt] - kOffOG * CaOG[tt]}
;

(*ATP*)
BufferATP := {

  ATP[0] == ATP0,
  CaATP[0] == CaATP0,
  MgATP[0] == MgATP0,

  ATP'[tt] == -kOnATP * Ca[tt] * ATP[tt] + kOffATP * CaATP[tt] -
    kOnMgATP * Mg[tt] * ATP[tt] + kOffMgATP * MgATP[tt],
  CaATP'[tt] == kOnATP * Ca[tt] * ATP[tt] - kOffATP * CaATP[tt],
  MgATP'[tt] == kOnMgATP * Mg[tt] * ATP[tt] - kOffMgATP * MgATP[tt]
 }
;

(*fast (tauf) and slow (taus) time constants for uncageing;
```

```
  Faas et al., 2005,2007*)
If[MgT == 0.,
   tauf = 15.2 * 10^-6;  (*Faas, 2007*)
   taus = 1.9 * 10^-3;
   ,
   tauf = 15. * 10^-6;  (*Faas,2005*)
   taus = 2. * 10^-3; ]

  ;

  (*The differential equations*)
  (*non uncaging fraction of DMn*)
  BufferDMn := {

    DMn[0] == DMn0,
    CaDMn[0] == CaDMn0,
    MgDMn[0] == MgDMn0,

    DMn'[tt] == -kOnDM * Ca[tt] * DMn[tt] +
      kOffDM * CaDMn[tt] - kOnMg * Mg[tt] * DMn[tt] + kOffMg * MgDMn[tt],
    CaDMn'[tt] == kOnDM * Ca[tt] * DMn[tt] - kOffDM * CaDMn[tt],
    MgDMn'[tt] == kOnMg * Mg[tt] * DMn[tt] - kOffMg * MgDMn[tt]
   }
  ;

  (*fast uncaging fraction of DMn*)
  BufferDMf := {

    DMf[0] == DMf0,
    CaDMf[0] == CaDMf0,
    MgDMf[0] == MgDMf0,

    DMf'[tt] == -kOnDM * Ca[tt] * DMf[tt] + kOffDM * CaDMf[tt] - kOnMg * Mg[tt] *
        DMf[tt] + kOffMg * MgDMf[tt] - 1 / tauf * DMf[tt] * UnitStep[tt - tflash],
    CaDMf'[tt] == kOnDM * Ca[tt] * DMf[tt] - kOffDM * CaDMf[tt] -
      1 / tauf * CaDMf[tt] * UnitStep[tt - tflash],
    MgDMf'[tt] == kOnMg * Mg[tt] * DMf[tt] - kOffMg * MgDMf[tt] -
      1 / tauf * MgDMf[tt] * UnitStep[tt - tflash]
   }

  ;
  (*slow uncaging fraction of DMn*)
  BufferDMs := {

    DMs[0] == DMs0,
    CaDMs[0] == CaDMs0,
    MgDMs[0] == MgDMs0,
```

```
  DMs'[tt] == -kOnDM * Ca[tt] * DMs[tt] + kOffDM * CaDMs[tt] - kOnMg * Mg[tt] *
     DMs[tt] + kOffMg * MgDMs[tt] - 1 / taus * DMs[tt] * UnitStep[tt - tflash],
  CaDMs'[tt] == kOnDM * Ca[tt] * DMs[tt] - kOffDM * CaDMs[tt] -
    1 / taus * CaDMs[tt] * UnitStep[tt - tflash],
  MgDMs'[tt] == kOnMg * Mg[tt] * DMs[tt] - kOffMg * MgDMs[tt] -
    1 / taus * MgDMs[tt] * UnitStep[tt - tflash]
 }
;

(*Photoproducts*)
(*PP2: comes from DMf,DMs and MgDMf,MgDMs; but also binds Ca*)
BufferPP2 := {

  PP2[0] == 0,
  CaPP2[0] == 0,
  MgPP2[0] == 0,

  PP2'[tt] == -kOnPP2 * Ca[tt] * PP2[tt] + kOffPP2 * CaPP2[tt] -
    kOnMg * Mg[tt] * PP2[tt] + kOffMgPP * MgPP2[tt] + 2 * (1 / tauf * DMf[tt] *
        UnitStep[tt - tflash] + 1 / taus * DMs[tt] * UnitStep[tt - tflash])
    + 1 / tauf * MgDMf[tt] * UnitStep[tt - tflash] +
    1 / taus * MgDMs[tt] * UnitStep[tt - tflash],

  CaPP2'[tt] == kOnPP2 * Ca[tt] * PP2[tt] - kOffPP2 * CaPP2[tt],

  MgPP2'[tt] == kOnMg * Mg[tt] * PP2[tt] - kOffMgPP * MgPP2[tt] + 1 / tauf *
     MgDMf[tt] * UnitStep[tt - tflash] + 1 / taus * MgDMs[tt] * UnitStep[tt - tflash]
 }
;

(*PP1: Comes from CaDMf,CaDMs and binds Ca and Mg*)
BufferPP1 := {

  PP1[0] == 0,
  CaPP1[0] == 0,
  MgPP1[0] == 0,

  PP1'[tt] == -kOnPP1 * Ca[tt] * PP1[tt] +
    kOffPP1 * CaPP1[tt] - kOnMg * Mg[tt] * PP1[tt] + kOffMgPP * MgPP1[tt]
    + 1 / tauf * CaDMf[tt] * UnitStep[tt - tflash] +
    1 / taus * CaDMs[tt] * UnitStep[tt - tflash],

  CaPP1'[tt] == kOnPP1 * Ca[tt] * PP1[tt] -
    kOffPP1 * CaPP1[tt] + 1 / tauf * CaDMf[tt] * UnitStep[tt - tflash] +
    1 / taus * CaDMs[tt] * UnitStep[tt - tflash],
```

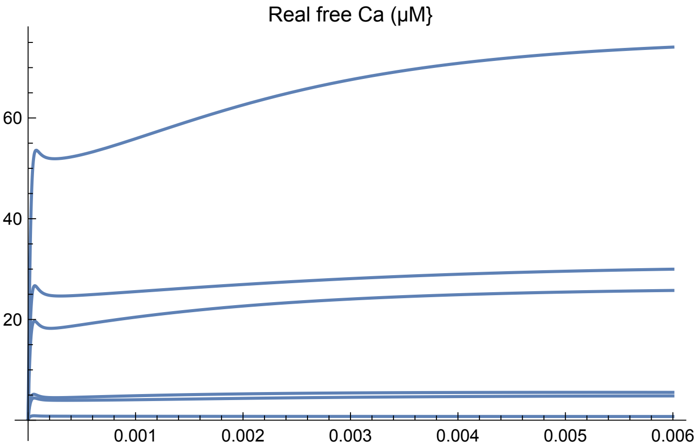
```
    MgPP1'[tt] == kOnMg * Mg[tt] * PP1[tt] - kOffMgPP * MgPP1[tt]

   }
  ;

  (*Endogeneous Buffer*)
  BufferMB := {

    MB[0] == MB0,
    CaMB[0] == CaMB0,

    MB'[tt] == -kOnMB * Ca[tt] * MB[tt] + kOffMB * CaMB[tt],
    CaMB'[tt] == kOnMB * Ca[tt] * MB[tt] - kOffMB * CaMB[tt]}

  ;

(*Clear[Eqns,Vars,sol]*)

  (*Free Ca*)
  FreeCa := {
    Ca[0] == Ca0,

    Ca'[tt] == -γ * (Ca[tt] - CaRest)
      (*DMn*)
      - kOnPP1 * Ca[tt] * PP1[tt] + kOffPP1 * CaPP1[tt]
      - kOnPP2 * Ca[tt] * PP2[tt] + kOffPP2 * CaPP2[tt]
      - kOnDM * Ca[tt] * DMn[tt] + kOffDM * CaDMn[tt]
      - kOnDM * Ca[tt] * DMf[tt] + kOffDM * CaDMf[tt]
      - kOnDM * Ca[tt] * DMs[tt] + kOffDM * CaDMs[tt]
      (*buffers*)
      - kOnATP * Ca[tt] * ATP[tt] + kOffATP * CaATP[tt]
      - kOnMB * Ca[tt] * MB[tt] + kOffMB * CaMB[tt]
      (*dye*)
      - kOnOG * Ca[tt] * OG[tt] + kOffOG * CaOG[tt]
    }
  ;

  (*Free Mg*)
  FreeMg := {
    Mg[0] == Mg0,

    Mg'[tt] ==
     -kOnMgATP * Mg[tt] * ATP[tt] + kOffMgATP * MgATP[tt]
      - kOnMg * Mg[tt] * DMn[tt] + kOffMg * MgDMn[tt]
      - kOnMg * Mg[tt] * DMf[tt] + kOffMg * MgDMf[tt]
      - kOnMg * Mg[tt] * DMs[tt] + kOffMg * MgDMs[tt]
      - kOnMg * Mg[tt] * PP2[tt] + kOffMgPP * MgPP2[tt]
```

```
    - kOnMg * Mg[tt] * PP1[tt] + kOffMgPP * MgPP1[tt]
  }

 ;

 Eqns := {BufferDMn, BufferDMf, BufferDMs, BufferATP,
   BufferPP1, BufferPP2, FreeCa, FreeMg, BufferMB, BufferOG}
 ;
 Vars := {ATP, CaATP, MgATP, Ca, Mg, CaDMn, DMn, CaDMf, DMf, CaDMs,
   DMs, CaPP1, PP1, CaPP2, PP2, MgPP2, MgPP1, MB, CaMB, OG, CaOG}
 ;
 sol := NDSolve[Eqns, Vars, {tt, 0., TimeWindow}
   (*,Method→{"EquationSimplification"->"Solve"}*)]
 ;
 CafP = Extract[Ca[TimeWindow] /. sol, 1];
 CafOG = KdOG * CaOG[tt] / OG[tt];


 AppendTo[CaListReal, Evaluate[{Ca[tt]} /. sol]];
 AppendTo[CaListDye, Evaluate[{CafOG} /. sol]];
];
```

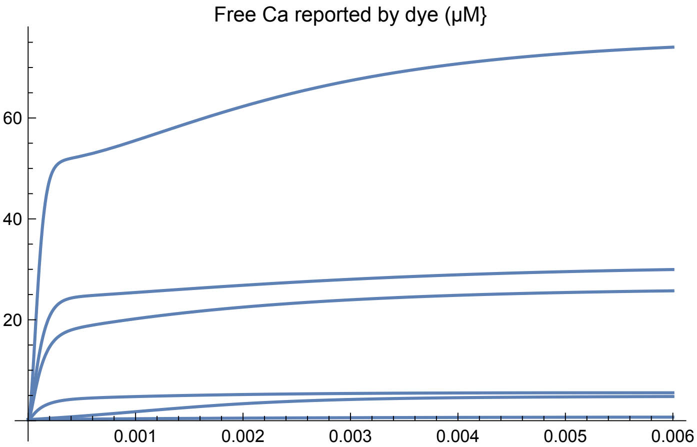
Plot and further processing

```
In[•]:= Plot[10^6 * CaListReal, {tt, 0, TimeWindow}, PlotLabel → "Real free Ca (μM}"]
Plot[10^6 * CaListDye, {tt, 0, TimeWindow},
 PlotLabel → "Free Ca reported by dye (μM}"]
LogPlot[10^6 * CaListReal, {tt, 0, TimeWindow}, PlotLabel → "Real free Ca (μM}"]
LogPlot[10^6 * CaListDye, {tt, 0, TimeWindow},
 PlotLabel → "Free Ca reported by dye (μM}"]
CaListDyePeak =
 Table[NMaximize[{CaListDye[[nn]][[1]][[1]], 0 ≤ tt ≤ TimeWindow}, tt][[1]],
  {nn, Length[CaListDye]}]
  (*"[[1]][[1]]" is neede to get rid of these brackets {{}}*)
```

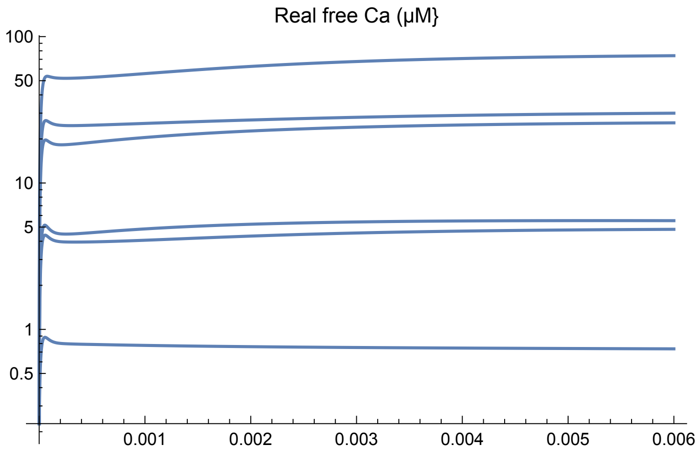
Out[ ]=


Real free Ca (μM}
60
40
20
0.001
0.002
0.003
0.004
0.005
0.006

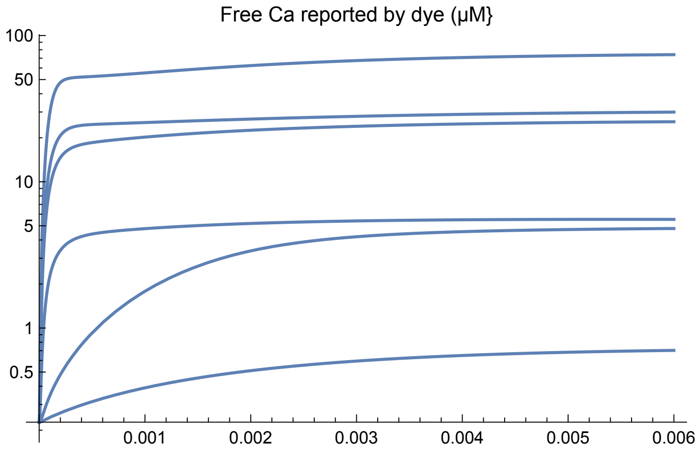
Out[ ]=


Free Ca reported by dye (μM}
60
40
20
0.001
0.002
0.003
0.004
0.005
0.006

Out[ ]=


Real free Ca (μM}
100
50
10
5
1
0.5
0.001
0.002
0.003
0.004
0.005
0.006

Out[•]=



Out[•]= {7.03073×10^{-7}, 4.79194×10^{-6}, 5.53376×10^{-6}, 0.0000257412, 0.0000299583, 0.0000740312}

In[•]:=
```
runs = Length[CaListDyePeak];
```

Release scheme and parameters

Define release scheme A

In[•]:=
```
nStates = 7;
matA = Table[0, {nStates}, {nStates}];
```

forward rates

```
In[•]:= from = 1; (*0 ca bound*)
kk = 5 konA;
matA[[from + 1, from]] += kk; matA[[from, from]] += -kk;

from = 2;
kk = 4 konA;
matA[[from + 1, from]] += kk; matA[[from, from]] += -kk;

from = 3;
kk = 3 konA;
matA[[from + 1, from]] += kk; matA[[from, from]] += -kk;

from = 4;
kk = 2 konA;
matA[[from + 1, from]] += kk; matA[[from, from]] += -kk;

from = 5; (*4 ca bound*)
kk = konA;
matA[[from + 1, from]] += kk; matA[[from, from]] += -kk;

from = 6; (*5 ca bound*)
kk = gammaA;
matA[[from + 1, from]] += kk; matA[[from, from]] += -kk;
```

backwards rates

```
In[•]:= from = 2;(*1 ca bound*)
kk = koffA bA^0;
matA[[from - 1, from]] += kk; matA[[from, from]] += -kk;

from = 3;
kk = 2 koffA bA^1;
matA[[from - 1, from]] += kk; matA[[from, from]] += -kk;

from = 4;
kk = 3 koffA bA^2;
matA[[from - 1, from]] += kk; matA[[from, from]] += -kk;

from = 5;
kk = 4 koffA bA^3;
matA[[from - 1, from]] += kk; matA[[from, from]] += -kk;

from = 6;(*5 ca bound*)
kk = 5 koffA bA^4;
matA[[from - 1, from]] += kk; matA[[from, from]] += -kk;
```

outflux from matrix

```
In[•]:= matA[[1, 1]] += -kunprimA - kprimB;
 (*due to multiplication with ss1A[t] in mat.ss it results in:
   -kunprimA ss1A[t] -kprimB ss1A[t]
  *)
```

influx in matrix

```
In[•]:= matA[[1, 1]] += kprimA (1 - ss1A[t]) / ss1A[t] + kunprimB ss1B[t] / ss1A[t];
(* due to multiplication with ss1A[t] in mat.ss it results in:
    kprimA + kunprimB ss1B[t] *)
```

Matrix

In[∘]:=
```
matA // TableForm
```

Out[∘]//TableForm=

$-5\ konA - kprimB - kunprimA + \frac{kprimA\,(1-ss1A[t])}{ss1A[t]} + \frac{kunprimB\ ss1B[t]}{ss1A[t]}$	koffA	0
5 konA	− koffA − 4 konA	2 bA koff
0	4 konA	− 2 bA koı
0	0	3 konA
0	0	0
0	0	0
0	0	0

Define release scheme B

In[∘]:=
```
nStates = 7;
matB = Table[0, {nStates}, {nStates}];
```

forward rates

In[∘]:=
```
from = 1; (*0 ca bound*)
kk = 5 konB;
matB[[from + 1, from]] += kk; matB[[from, from]] += -kk;

from = 2;
kk = 4 konB;
matB[[from + 1, from]] += kk; matB[[from, from]] += -kk;

from = 3;
kk = 3 konB;
matB[[from + 1, from]] += kk; matB[[from, from]] += -kk;

from = 4;
kk = 2 konB;
matB[[from + 1, from]] += kk; matB[[from, from]] += -kk;

from = 5; (*4 ca bound*)
kk = konB;
matB[[from + 1, from]] += kk; matB[[from, from]] += -kk;

from = 6; (*5 ca bound*)
kk = gammaB;
matB[[from + 1, from]] += kk; matB[[from, from]] += -kk;
```

backwards rates

```
In[•]:= from = 2; (*1 ca bound*)
kk = koffB bB ^ 0;
matB[[from - 1, from]] += kk; matB[[from, from]] += -kk;

from = 3;
kk = 2 koffB bB ^ 1;
matB[[from - 1, from]] += kk; matB[[from, from]] += -kk;

from = 4;
kk = 3 koffB bB ^ 2;
matB[[from - 1, from]] += kk; matB[[from, from]] += -kk;

from = 5;
kk = 4 koffB bB ^ 3;
matB[[from - 1, from]] += kk; matB[[from, from]] += -kk;

from = 6; (*5 ca bound*)
kk = 5 koffB bB ^ 4;
matB[[from - 1, from]] += kk; matB[[from, from]] += -kk;
```

outflux from matrix

```
In[•]:= matB[[1, 1]] += -kunprimB;
(* due to multiplication with ss1B[t] in mat.ss it results in:
  -kunprimB ss1B[t]
*)
```

influx in matrix

```
In[•]:= matB[[1, 1]] += kprimB ss1A[t] / ss1B[t];
(* due to multiplication with ss1B[t] in mat.ss it results in:
  kprimB ss1A[t]
*)
```

Matrix

In[ ]:=
```
matB // TableForm
```

Out[ ]//TableForm=

$-5\ \text{konB} - \text{kunprimB} + \frac{\text{kprimB ss1A[t]}}{\text{ss1B[t]}}$	koffB	0	0
5 konB	− koffB − 4 konB	2 bB koffB	0
0	4 konB	− 2 bB koffB − 3 konB	$3\ \text{bB}^2\ \text{koffB}$
0	0	3 konB	$-3\ \text{bB}^2\ \text{koffB}$
0	0	0	2 konB
0	0	0	0
0	0	0	0

Parameters of release scheme

In[ ]:=
```
q10 = 2.3;
tempFact = q10 ^ ((37 - 24) / 10);
```

In[ ]:=
```
Clear[caFunc, konSchemeA, konSchemeB];
(*Clear is needed if the cell is exectued for a 2nd time when
 caFunc is already set to a value or an Interpolationfunction*)

caRest = CaRest;(*227nM see above*)

(*slow ves*)
affinityFactorA = 3.0;
konSchemeA = 0.5 * caFunc[t] Sqrt[affinityFactorA] tempFact 1.*^8;
(*M^-1 s^-1*)
koffSchemeA = 10. * (1 / Sqrt[affinityFactorA]) tempFact 15 000.;(*s^-1*)
gammaSchemeA = tempFact 6000.;(*s^-1*)
bSchemeA = 0.25;

KdA = 2.*^-6;
kprimSchemeA = 30. + 0. * 30. * (caFunc[t] / (KdA + caFunc[t])) ;
kunprimSchemeA = 30. + 0. * 30. * (caRest / (KdA + caRest)) ;

(*fast ves*)
affinityFactorB = 3.0;
konSchemeB = caFunc[t] Sqrt[affinityFactorB] tempFact 1.*^8;(*M^-1 s^-1*)
koffSchemeB = (1 / Sqrt[affinityFactorB]) tempFact 15 000.;(*s^-1*)
gammaSchemeB = tempFact 6000.;(*s^-1*)
bSchemeB = 0.25;

KdB = 2.*^-6;
kprimSchemeB = 0.5 + 30. * (caFunc[t] / (KdB + caFunc[t])) ;
kunprimSchemeB = 0.5 + 30. * (caRest / (KdB + caRest)) ;
```

```
repl = {
   konA → konSchemeA,
   koffA → koffSchemeA ,
   gammaA → gammaSchemeA ,
    bA → bSchemeA,

   kprimA → kprimSchemeA ,
   kunprimA → kunprimSchemeA ,


   konB → konSchemeB,
   koffB → koffSchemeB ,
   gammaB → gammaSchemeB ,
    bB → bSchemeB,

   kprimB → kprimSchemeB ,
   kunprimB → kunprimSchemeB
  };
```

In[•]:=
```
tauOfDecayOfUncagedCa = 0.4;
```

Initial occupancy

In[•]:=
```
(*test initial equilibrium occupancy*)
caFunc[t_] := caTmp;
ss0AInitial = kprimSchemeA / kunprimSchemeA;
ss0BInitial = (kprimSchemeA / kunprimSchemeA) * (kprimSchemeB / kunprimSchemeB);
(ss0AInitial /. caTmp → 180*^-9)
(ss0AInitial /. caTmp → 30*^-9)
(ss0AInitial /. caTmp → 180*^-9) / (ss0AInitial /. caTmp → 30*^-9)
(ss0BInitial /. caTmp → 180*^-9)
(ss0BInitial /. caTmp → 30*^-9)
(ss0BInitial /. caTmp → 180*^-9) / (ss0BInitial /. caTmp → 30*^-9)
```

Out[•]= 1.

Out[•]= 1.

Out[•]= 1.

Out[•]= 0.836742

Out[•]= 0.26514

Out[•]= 3.15584

In[•]:=
```
(*calualte initial equilibrium occupancy*)
caFunc[t_] := caRest;
ss0AInitial = kprimSchemeA / kunprimSchemeA;
ss0BInitial = (kprimSchemeA / kunprimSchemeA) * (kprimSchemeB / kunprimSchemeB);
```

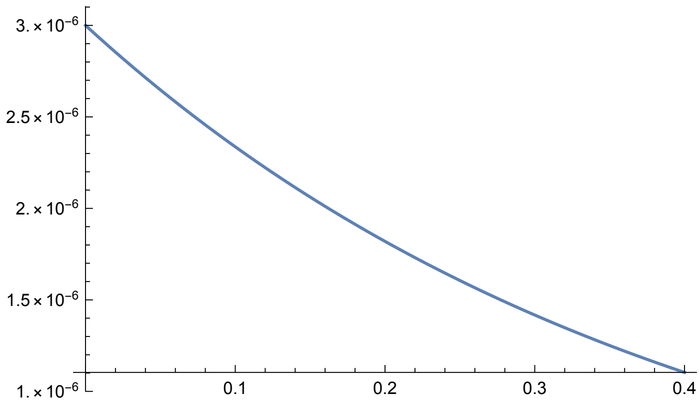
Diff eq.

In[•]:=
```
Clear[caFunc, eq];(*Clear is needed if the cell is exectued for a 2nd time
 when caFunc is already set to a value or an Interpolationfunction*)
ssA[t_] = {ss1A[t], ss2A[t], ss3A[t], ss4A[t], ss5A[t], ss6A[t], ss7A[t]};
ssB[t_] = {ss1B[t], ss2B[t], ss3B[t], ss4B[t], ss5B[t], ss6B[t], ss7B[t]};
eq = {
   ssA'[t] == (matA /. repl).ssA[t],
   ssA[0] == {ss0AInitial, 0, 0, 0, 0, 0, 0},

   ssB'[t] == (matB /. repl).ssB[t],
   ssB[0] == {ss0BInitial, 0, 0, 0, 0, 0, 0}
  };
```

Solve all states

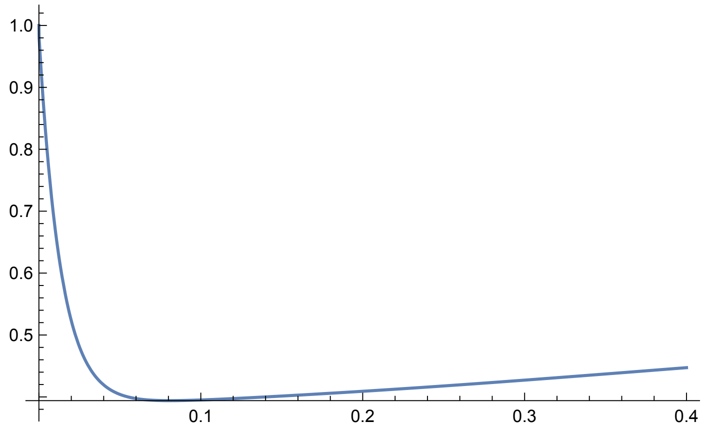
In[•]:=
```
caFunc[t_] := 3*^-6 * Exp[-t / tauOfDecayOfUncagedCa];
Plot[caFunc[t], {t, 0, 0.4}, PlotRange → All]
```

Out[•]=

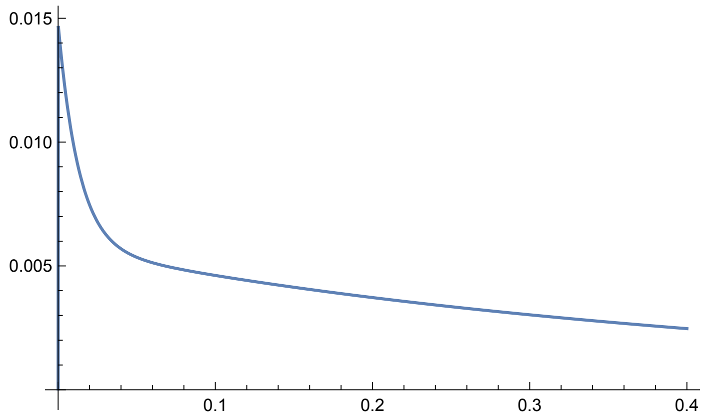


```
In[•]:= myNDSolveResults = NDSolve[eq, {ss1A, ss2A, ss3A, ss4A, ss5A,
          ss6A, ss7A, ss1B, ss2B, ss3B, ss4B, ss5B, ss6B, ss7B}, {t, 0, 10.}];
       tEndForPlot = .4;
       Plot[(ss1A[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
       Plot[(ss2A[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
       Plot[(ss3A[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
       Plot[(ss4A[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
       Plot[(ss5A[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
       Plot[(ss6A[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
       Plot[(ss7A[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
       Print[
        "-----------------------------------------------------------------------"]
       Plot[(ss1B[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
       Plot[(ss2B[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
       Plot[(ss3B[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
       Plot[(ss4B[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
       Plot[(ss5B[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
       Plot[(ss6B[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
       Plot[(ss7B[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
```

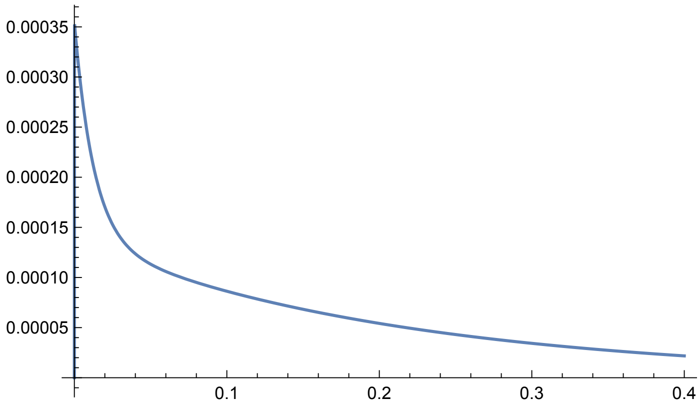
Out[•]=



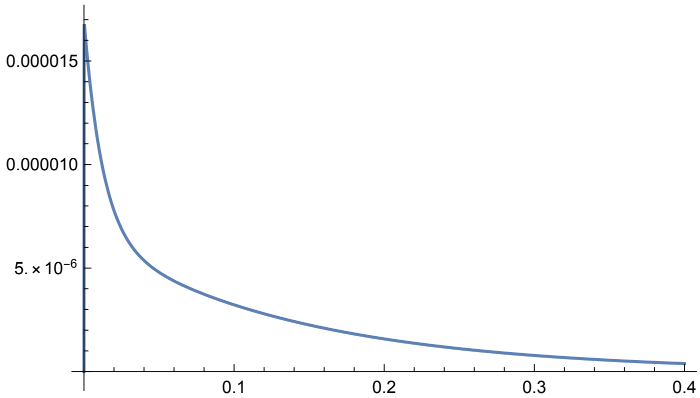
Out[•]=



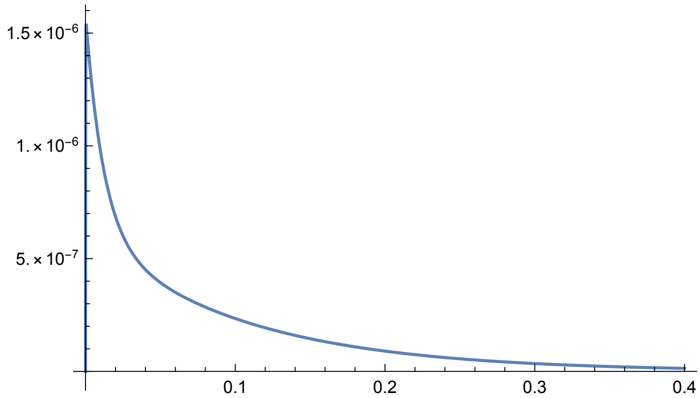
Out[∘]=



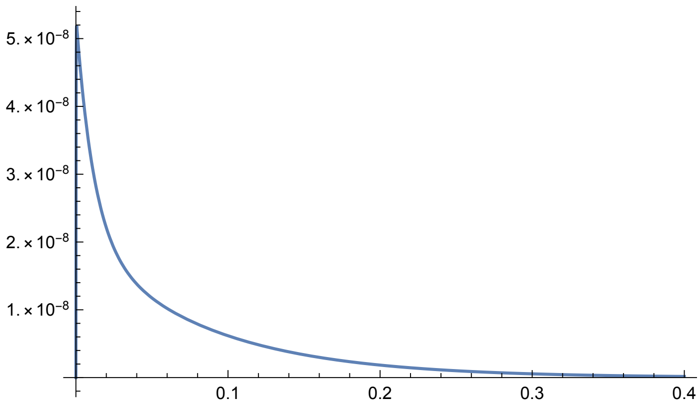
Out[∘]=



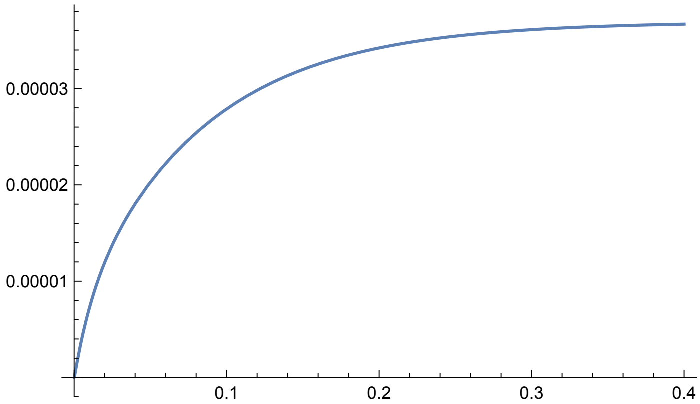
Out[∘]=



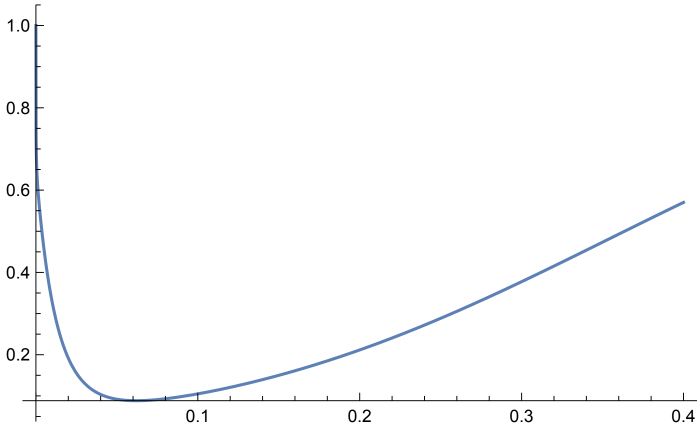
Out[∘]=



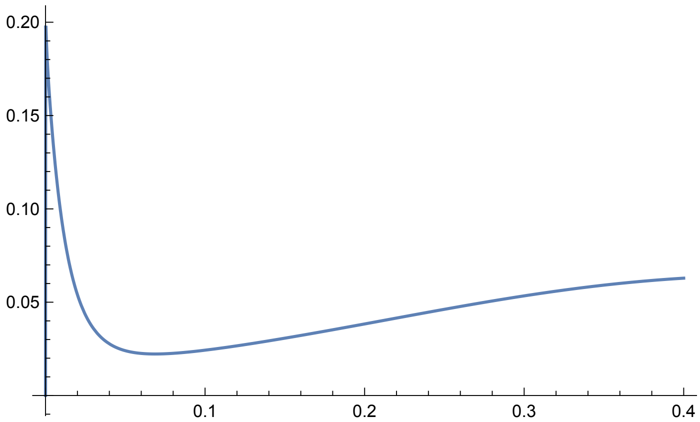
Out[•]=



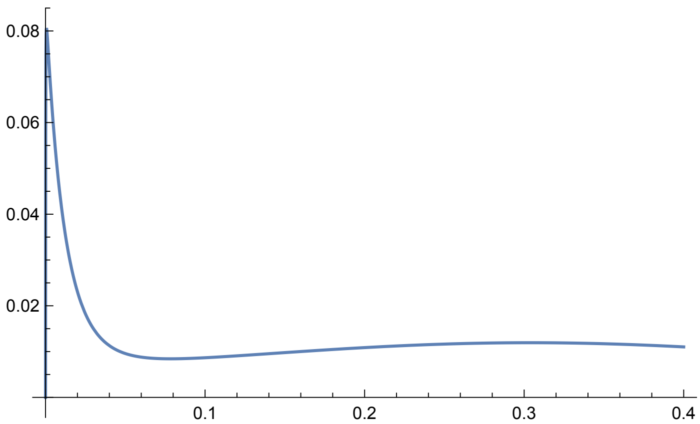
Out[•]=



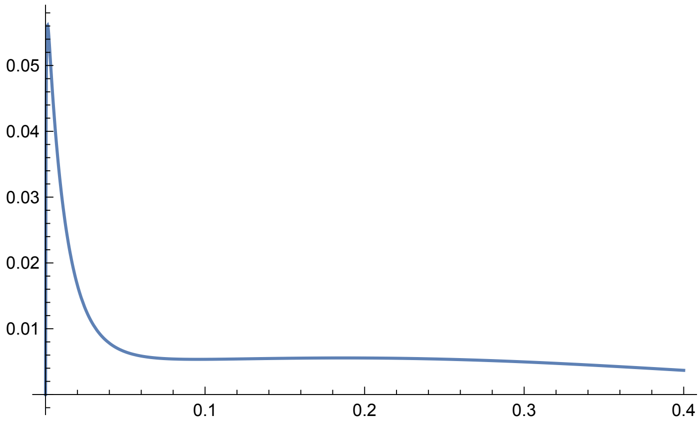
Out[•]=



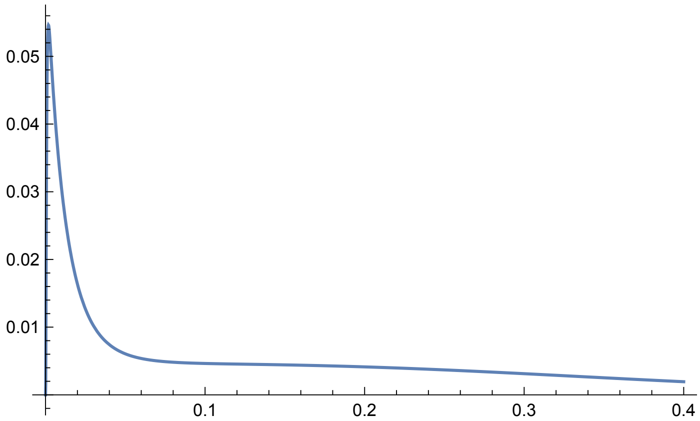
Out[•]=



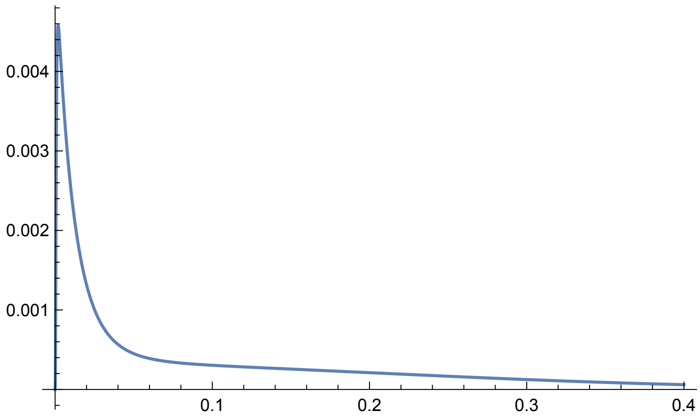
Out[∘]=



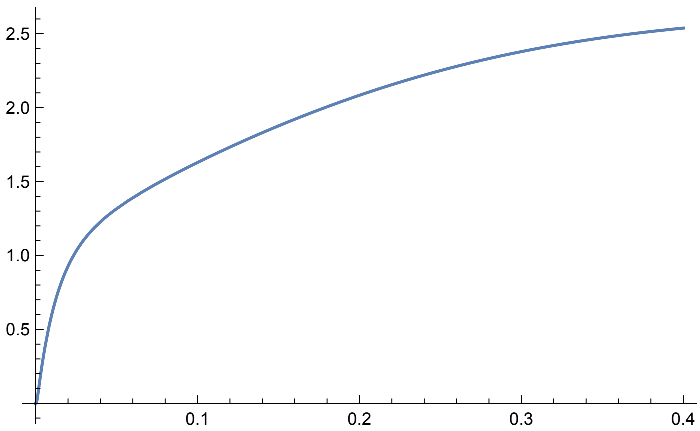
Out[∘]=



Out[∘]=



Out[∘]=



Loop

make lists for later

```
In[•]:= simCaList = Table[0, {runs}];
simParamNv = Table[0, {7}, {runs}];
```

C5

```
In[•]:= baselineC5 = Table[{ttt, 0}, {ttt, cursorStart, -1*^-6, dtOfDataC5}];
(*List to save data within loop*)
(*ca; tau; number of released vesilces at time=cursorEnd; delay*)
simParamNoiseC5 = Table[0, {numberOfFitParamToBeSaved}, {noiseRepeats}];
simParamMedianC5 = Table[0, {numberOfFitParamToBeSaved}, {runs}];
simParamQuantile1C5 = Table[0, {numberOfFitParamToBeSaved}, {runs}];
simParamQuantile2C5 = Table[0, {numberOfFitParamToBeSaved}, {runs}];
```

C10

```
In[•]:= baselineC10 = Table[{ttt, 0}, {ttt, cursorStart, -1*^-6, dtOfDataC10}];
(*List to save data within loop*)
(*ca; tau; number of released vesilces at time=cursorEnd; delay*)
simParamNoiseC10 = Table[0, {numberOfFitParamToBeSaved}, {noiseRepeats}];
simParamMedianC10 = Table[0, {numberOfFitParamToBeSaved}, {runs}];
simParamQuantile1C10 = Table[0, {numberOfFitParamToBeSaved}, {runs}];
simParamQuantile2C10 = Table[0, {numberOfFitParamToBeSaved}, {runs}];
```

D

```
In[•]:= baselineD = Table[{ttt, 0}, {ttt, cursorStart, -1*^-6, dtOfDataD}];
(*List to save data within loop*)
(*ca; tau; number of released vesilces at time=cursorEnd; delay*)
simParamNoiseD = Table[0, {numberOfFitParamToBeSaved}, {noiseRepeats}];
simParamMedianD = Table[0, {numberOfFitParamToBeSaved}, {runs}];
simParamQuantile1D = Table[0, {numberOfFitParamToBeSaved}, {runs}];
simParamQuantile2D = Table[0, {numberOfFitParamToBeSaved}, {runs}];
```

Long

```
In[•]:= baselineLong = Table[{ttt, 0}, {ttt, -0.05, -1*^-6, dtOfDataLong}];
```

Loop

```
In[•]:= For[r = 1, r ≤ runs, r += 1,
  myCaNow = CaListDyePeak[[r]];
  lastSimulatedCaListReal = CaListReal[[r]][[1]][[1]] /. tt → TimeWindow;
  (*"[[1]][[1]]" is neede to get rid of these brackets {{}}*)
  Print[
   "-----------------------------------------------------------------------
     ----------------------------------"];
  Print["----------------    Ca = ", 1*^6 myCaNow,
   " uM         -------------------------------------------------------"];
  Print[
   "-----------------------------------------------------------------------
     ----------------------------------"];
  simCaList[[r]] = myCaNow;
  caFunc[ttt_] := If[ttt < TimeWindow,
    CaListReal[[r]][[1]][[1]] /. tt → ttt,
    (*tt because this is the symbol used in "Calculate Ca transients"*)
    lastSimulatedCaListReal * Exp[-ttt / tauOfDecayOfUncagedCa]
   ];
  (* solve Diff Eq.: *)
  myNDSolveResults = NDSolve[eq, {ss7A, ss7B}, {t, 0, 0.4}];
  (* plot results: *)
  fused[t_] := (ss7A[t] + ss7B[t]);
  Plot[(fused[t] /. myNDSolveResults),
    {t, 0, cursorEnd}, PlotRange → All] // Print;
  Plot[(fused[t] /. myNDSolveResults), {t, 0, cursorEndLong},
    PlotRange → All] // Print;
  If[exportYes == 1,
   toExport = Table[{t, (fused[t] /. myNDSolveResults)[[1]]},
     {t, 0., cursorEndLong, dtOfPlotsForExport}];
   Export["withinLoop r" <> ToString[r] <> " Ca" <> ToString[1*^6 myCaNow] <>
     " withoutNoise.txt", toExport, "Table"];
  ];
  (* sample data and add baseline: --------*)
  tmpCumRelC5 = Table[{ttt, (fused[ttt] /. myNDSolveResults)[[1]]},
    {ttt, 0., cursorEnd, dtOfDataC5}];
  tmpToFitC5 = Catenate[{baselineC5, tmpCumRelC5}];
  tmpCumRelC10 = Table[{ttt, (fused[ttt] /. myNDSolveResults)[[1]]},
    {ttt, 0., cursorEnd, dtOfDataC10}];
  tmpToFitC10 = Catenate[{baselineC10, tmpCumRelC10}];
  tmpCumRelD = Table[{ttt, (fused[ttt] /. myNDSolveResults)[[1]]},
    {ttt, 0., cursorEnd, dtOfDataD}];
  tmpToFitD = Catenate[{baselineD, tmpCumRelD}];
  tmpCumRelLong = Table[{ttt, (fused[ttt] /. myNDSolveResults)[[1]]},
    {ttt, 0., cursorEndLong, dtOfDataLong}];
  tmpToFitLong = Catenate[{baselineLong, tmpCumRelLong}];
```

```
(*---------    get amplitude
  without noise and without fitting -----------------*)
For[NvCount = 1, NvCount ≤ 7, NvCount += 1,
 simParamNv[[NvCount, r]] =
   (fused[timeOfNv[[NvCount]]] /. myNDSolveResults)[[1]];
  (* the [[1]] is somehow needed to get rid of a list structure
  probably related to the interpolate function*)
];

(*--------------   Startvalues for
  fit the same for all C5 C10 and D -------------*)
caAdjustedTau1Guess = tau1Guess / ((myCaNow / 10.*^-6) ^4);
caAdjustedTau1Guess = tau1Guess / ((myCaNow / 10.*^-6) ^1);
caAdjustedTau2Guess = 10 caAdjustedTau1Guess;
caAdjustedDelayGuess = delayGuess / ((myCaNow / 10.*^-6) ^1);

(*---------    Fitting of data,
saving of results, and plotting-----------------*)
(*-------------------------------------------------------------------
                                    ----*)
(*------------------------------------    C5     -----------------------
                        ----*)
(*-------------------------------------------------------------------
                                    ----*)
Print[
 "---------------------------------------------------------------------
   C5"];
(*check that signal is large enough relative to noise to obtain
 useful fit results. If not, do not do fitting and set everything to {}*)
If[simParamNv[[5, r]] > signalToNoiseRatioC5 myNoiseC5, (*add noise
  several times and do the fitting and than average the results*)
 For[noiseN = 1, noiseN ≤ noiseRepeats, noiseN += 1,
  (*Print[noiseN];*)
  tmpToFitNoise = Transpose[{tmpToFitC5[[All, 1]],
     # + RandomVariate[NormalDistribution[0, myNoiseC5]] & /@
      tmpToFitC5[[All, 2]]}];
  (*fit mono-exp*)
  fitResultsTMP = NonlinearModelFit[tmpToFitNoise, myFitMono[x1],
    {{delayMono, delayGuess}, {ampMono, ampGuess}, {tau1Mono,
      caAdjustedTau1Guess}}, x1, MaxIterations → myMaxIterations];
  fitResultMono = fitResultsTMP[{"BestFitParameters"}];
  simParamNoiseC5[[1, noiseN]] = 0; (*not used anymore*)
  simParamNoiseC5[[2, noiseN]] =
   fitResultsTMP["ANOVATableSumsOfSquares"][[2]];
  (*Print[fitResultsTMP["ANOVATableSumsOfSquares"]];*)
  simParamNoiseC5[[3, noiseN]] = (delayMono /. fitResultMono)[[1]];
  simParamNoiseC5[[4, noiseN]] = ((ampMono /. fitResultMono))[[1]];
```

```
simParamNoiseC5[[5, noiseN]] = (1/(tau1Mono /. fitResultMono))[[1]];
(*fit bi-exp*)
fitResultsTMP =
 NonlinearModelFit[tmpToFitNoise, myFitBi[x1], {{delay, delayGuess},
   {amp, ampGuess}, {amp1, amp1Guess}, {tau1, caAdjustedTau1Guess},
   {tau2, caAdjustedTau2Guess}}, x1, MaxIterations → myMaxIterations];
fitResultBi = fitResultsTMP[{"BestFitParameters"}];
simParamNoiseC5[[6, noiseN]] = simParamNoiseC5[[2, noiseN]] /
  fitResultsTMP["ANOVATableSumsOfSquares"][[2]];
simParamNoiseC5[[7, noiseN]] = (delay /. fitResultBi)[[1]];
simParamNoiseC5[[8, noiseN]] = ((amp /. fitResultBi))[[1]];
simParamNoiseC5[[9, noiseN]] = ((amp amp1 /. fitResultBi))[[1]];
(*relative amp1*)
simParamNoiseC5[[10, noiseN]] = (1/(tau1 /. fitResultBi))[[1]];
simParamNoiseC5[[11, noiseN]] = (1/(tau2 /. fitResultBi))[[1]];
(*merge*)
If[
 (*to use the bi-exp fit, the following cirteria should be fullfilled:*)
 (*chi2 should imporve(=decrease) by >10%*)
 (simParamNoiseC5[[6, noiseN]] > 1.04)
  &&
  (*tau1 and tau2 of bi fit should be factor of >3 different*)
  ((simParamNoiseC5[[11, noiseN]] / simParamNoiseC5[[10, noiseN]]) < 3.)
  &&
  (*relative amplitude of 1st component should be > 5% *)
  ((( amp1 /. fitResultBi))[[1]] > 0.05)
  &&
  (*relative amplitude of 1st component should be < 95% *)
  ((( amp1 /. fitResultBi))[[1]] < 0.95)
 ,
 (*take bi*)
 Print["take bi"];
 simParamNoiseC5[[12, noiseN]] = simParamNoiseC5[[7, noiseN]];
 (*delay*)
 simParamNoiseC5[[13, noiseN]] = simParamNoiseC5[[8, noiseN]];
 (*amp*)
 simParamNoiseC5[[14, noiseN]] = simParamNoiseC5[[9, noiseN]];
 (*amp1*)
 simParamNoiseC5[[15, noiseN]] = simParamNoiseC5[[10, noiseN]];
 (*tau1*)
 simParamNoiseC5[[16, noiseN]] = simParamNoiseC5[[11, noiseN]];(*tau2*)
 ,
 (*take mono*)
 Print["take mono"];
 simParamNoiseC5[[12, noiseN]] = simParamNoiseC5[[3, noiseN]];
 (*delay*)
```

```
    simParamNoiseC5[[13, noiseN]] = simParamNoiseC5[[4, noiseN]];
    (*amp*)
    simParamNoiseC5[[14, noiseN]] = NaN; (*amp1*)
    simParamNoiseC5[[15, noiseN]] = simParamNoiseC5[[5, noiseN]];
    (*tau1*)
    simParamNoiseC5[[16, noiseN]] = NaN;(*tau2*)
   ];
  ];
  (*plot last example of the noise loop*)
  gr1 = ListPlot[tmpToFitNoise, PlotRange → All, PlotStyle → Black];
  gr2 = Plot[myFitMono[x1] /. fitResultMono,
    {x1, cursorStart, cursorEnd}, PlotRange → All, PlotStyle → {Blue, Dashed}];
  gr3 = Plot[myFitBi[x1] /. fitResultBi, {x1, cursorStart, cursorEnd},
    PlotRange → All, PlotStyle → {Green, Dashed}];
  Show[gr1, gr2, gr3, PlotRange → All ] // Print;
  If[exportYes == 1,
   Export["withinLoop r" <> ToString[r] <> " Ca" <>
     ToString[1*^6 myCaNow] <> " C5 data.txt", tmpToFitNoise, "Table"];
   toExport = Table[{t, (myFitMono[t] /. fitResultMono)[[1]]},
     {t, cursorStart, cursorEnd, dtOfPlotsForExport}];
   Export["withinLoop r" <> ToString[r] <> " Ca" <> ToString[1*^6 myCaNow] <>
     " C5 fitMono.txt", toExport, "Table"];
   toExport = Table[{t, (myFitBi[t] /. fitResultBi)[[1]]},
     {t, cursorStart, cursorEnd, dtOfPlotsForExport}];
   Export["withinLoop r" <> ToString[r] <> " Ca" <>
     ToString[1*^6 myCaNow] <> " C5 fitBi.txt", toExport, "Table"];
  ];
  noiseN = noiseRepeats; (*fit parameter of last noisetrace*)
  Print["Mono: chi2 = ", simParamNoiseC5[[2, noiseN]], "     d = ",
   simParamNoiseC5[[2, noiseN]], "     a = ", simParamNoiseC5[[4, noiseN]],
   "     t = ", 1/simParamNoiseC5[[5, noiseN]]];
  Print["Bi: ratio chi2Mono/chi2Bi = ", simParamNoiseC5[[6, noiseN]],
   "     d = ", simParamNoiseC5[[7, noiseN]], "     a = ",
   simParamNoiseC5[[8, noiseN]], "     a1 = ", simParamNoiseC5[[9, noiseN]],
   "     t1 =", 1/simParamNoiseC5[[10, noiseN]],
   "     t2 = ", 1/simParamNoiseC5[[11, noiseN]]];
  (*average fit results*)
  For[p = 1, p ≤ numberOfFitParamToBeSaved, p += 1,
   simParamMedianC5[[p, r]] =
    Median[simParamNoiseC5[[p, All]] /. NaN → Sequence[]];
   simParamQuantile1C5[[p, r]] = Quantile[
     simParamNoiseC5[[p, All]] /. NaN → Sequence[], myQuantile1];
   simParamQuantile2C5[[p, r]] = Quantile[
     simParamNoiseC5[[p, All]] /. NaN → Sequence[], myQuantile2];
  ];
```

```
(*if tau1 merge > 10 ms, use long trace for fitting*)
If[(1/simParamMedianC5[[15, r]]) > 0.01,
 Print["Long trace was used for fitting."];
 For[noiseN = 1, noiseN ≤ noiseRepeats, noiseN += 1,
  tmpToFitNoiseLong = Transpose[{tmpToFitLong[[All, 1]],
      # + RandomVariate[NormalDistribution[0, myNoiseLong]] & /@
       tmpToFitLong[[All, 2]]}];
  (*fit mono-exp to Long trace*)
  fitResultsTMP = NonlinearModelFit[tmpToFitNoiseLong, myFitMono[x1],
    {{delayMono, delayGuess}, {ampMono, ampGuess}, {tau1Mono,
      caAdjustedTau1Guess}}, x1, MaxIterations → myMaxIterations];
  fitResultLongMono = fitResultsTMP[{"BestFitParameters"}];
  simParamNoiseC5[[2, noiseN]] =
   fitResultsTMP["ANOVATableSumsOfSquares"][[2]];
  (*Print[fitResultsTMP["ANOVATableSumsOfSquares"]];*)
  simParamNoiseC5[[3, noiseN]] = (delayMono /. fitResultLongMono)[[1]];
  simParamNoiseC5[[4, noiseN]] = ((ampMono /. fitResultLongMono))[[1]];
  simParamNoiseC5[[5, noiseN]] = (1/(tau1Mono /. fitResultLongMono))[[1]];
  (*fit bi-exp*)
  fitResultsTMP =
   NonlinearModelFit[tmpToFitNoiseLong, myFitBi[x1], {{delay, delayGuess},
     {amp, ampGuess}, {amp1, amp1Guess}, {tau1, caAdjustedTau1Guess},
     {tau2, caAdjustedTau2Guess}}, x1, MaxIterations → myMaxIterations];
  fitResultLongBi = fitResultsTMP[{"BestFitParameters"}];
  simParamNoiseC5[[6, noiseN]] = simParamNoiseC5[[2, noiseN]] /
    fitResultsTMP["ANOVATableSumsOfSquares"][[2]];
  simParamNoiseC5[[7, noiseN]] = (delay /. fitResultLongBi)[[1]];
  simParamNoiseC5[[8, noiseN]] = ((amp /. fitResultLongBi))[[1]];
  simParamNoiseC5[[9, noiseN]] = ((amp amp1 /. fitResultLongBi))[[1]];
  (*relative amp1*)
  simParamNoiseC5[[10, noiseN]] = (1/(tau1 /. fitResultLongBi))[[1]];
  simParamNoiseC5[[11, noiseN]] = (1/(tau2 /. fitResultLongBi))[[1]];
  (*merge*)
  If[
   (*to use the bi-exp fit,
   the following cirteria should be fullfilled:*)
   (*chi2 should imporve(=decrease) by >10%*)
   (simParamNoiseC5[[6, noiseN]] > 1.04)
    &&
    (*tau1 and tau2 of bi fit should be factor of >3 different*)
    ((simParamNoiseC5[[11, noiseN]]/simParamNoiseC5[[10, noiseN]]) < 3.)
    &&
    (*relative amplitude of 1st component should be > 5% *)
    ((( amp1 /. fitResultLongBi))[[1]] > 0.05)
    &&
    (*relative amplitude of 1st component should be < 95% *)
```

```
    (((amp1 /. fitResultLongBi))[[1]] < 0.95)
   ,
   (*take bi*)
   Print["take bi"];
   simParamNoiseC5[[12, noiseN]] = simParamNoiseC5[[7, noiseN]];
   (*delay*)
   simParamNoiseC5[[13, noiseN]] = simParamNoiseC5[[8, noiseN]];
   (*amp*)
   simParamNoiseC5[[14, noiseN]] = simParamNoiseC5[[9, noiseN]];
   (*amp1*)
   simParamNoiseC5[[15, noiseN]] = simParamNoiseC5[[10, noiseN]];
   (*tau1*)
   simParamNoiseC5[[16, noiseN]] = simParamNoiseC5[[11, noiseN]]; (*tau2*)
   ,
   (*take mono*)
   Print["take mono"];
   simParamNoiseC5[[12, noiseN]] = simParamNoiseC5[[3, noiseN]];
   (*delay*)
   simParamNoiseC5[[13, noiseN]] = simParamNoiseC5[[4, noiseN]];
   (*amp*)
   simParamNoiseC5[[14, noiseN]] = NaN; (*amp1*)
   simParamNoiseC5[[15, noiseN]] = simParamNoiseC5[[5, noiseN]];
   (*tau1*)
   simParamNoiseC5[[16, noiseN]] = NaN; (*tau2*)
  ];
 ];
(*plot last example of the noise loop*)
gr1 = ListPlot[tmpToFitNoiseLong, PlotRange → All, PlotStyle → Black];
gr2 = Plot[myFitMono[x1] /. fitResultLongMono, {x1, cursorStart,
    cursorEndLong}, PlotRange → All, PlotStyle → {Blue, Dashed}];
gr3 = Plot[myFitBi[x1] /. fitResultLongBi, {x1, cursorStart, cursorEndLong},
   PlotRange → All, PlotStyle → {Green, Dashed}];
Show[gr1, gr2, gr3, PlotRange → All ] // Print;
If[exportYes == 1,
 Export["withinLoop r" <> ToString[r] <> " Ca" <> ToString[1*^6 myCaNow] <>
    " C5 dataLong.txt", tmpToFitNoiseLong, "Table"];
 toExport = Table[{t, (myFitMono[t] /. fitResultLongMono)[[1]]},
    {t, cursorStart, cursorEndLong, dtOfPlotsForExport}];
 Export["withinLoop r" <> ToString[r] <> " Ca" <> ToString[1*^6 myCaNow] <>
    " C5 fitLongMono.txt", toExport, "Table"];
 toExport = Table[{t, (myFitBi[t] /. fitResultLongBi)[[1]]},
    {t, cursorStart, cursorEndLong, dtOfPlotsForExport}];
 Export["withinLoop r" <> ToString[r] <> " Ca" <> ToString[1*^6 myCaNow] <>
    " C5 fitLongBi.txt", toExport, "Table"];

];
noiseN = noiseRepeats; (*fit parameter of last noisetrace*)
```

```
   Print["Mono: chi2 = ", simParamNoiseC5[[2, noiseN]], "     d = ",
    simParamNoiseC5[[3, noiseN]], "     a = ", simParamNoiseC5[[4, noiseN]],
    "     t = ", 1/simParamNoiseC5[[5, noiseN]]];
   Print["Bi: ratio chi2Mono/chi2Bi = ", simParamNoiseC5[[6, noiseN]],
    "     d = ", simParamNoiseC5[[7, noiseN]], "     a = ",
    simParamNoiseC5[[8, noiseN]], "     a1 = ", simParamNoiseC5[[9, noiseN]],
    "     t1 =", 1/simParamNoiseC5[[10, noiseN]],
    "     t2 = ", 1/simParamNoiseC5[[11, noiseN]]];
   (*average fit results*)
   For[p = 1, p ≤ numberOfFitParamToBeSaved, p += 1,
    simParamMedianC5[[p, r]] =
     Median[simParamNoiseC5[[p, All]] /. NaN → Sequence[]];
    simParamQuantile1C5[[p, r]] = Quantile[
      simParamNoiseC5[[p, All]] /. NaN → Sequence[], myQuantile1];
    simParamQuantile2C5[[p, r]] = Quantile[
      simParamNoiseC5[[p, All]] /. NaN → Sequence[], myQuantile2];
   ];
  ];

  , (*else: signal is not large enough*)
  For[p = 1, p ≤ numberOfFitParamToBeSaved, p += 1,
    simParamMedianC5[[p, r]] = {};
    simParamQuantile1C5[[p, r]] = {};
    simParamQuantile2C5[[p, r]] = {};
   ];
];


(*---------    Fitting of data,
saving of results, and plotting-----------------*)
(*--------------------------------------------------------------------
                                    ----*)
(*------------------------------------    C10     ----------------------
                       ----*)
(*--------------------------------------------------------------------
                                    ----*)
Print[
 "---------------------------------------------------------------------
   C10"];
(*check that signal is large enough relative to noise to obtain
 useful fit results. If not, do not do fitting and set everything to {}*)
If[simParamNv[[5, r]] > signalToNoiseRatioC10 myNoiseC10, (*add noise
  and do the fitting several times and then average the results*)
 For[noiseN = 1, noiseN ≤ noiseRepeats, noiseN += 1,
  (*Print[noiseN];*)
  tmpToFitNoise = Transpose[{tmpToFitC10[[All, 1]],
     # + RandomVariate[NormalDistribution[0, myNoiseC10]] & /@
```

```
    tmpToFitC10[[All, 2]]}];
(*fit mono-exp*)
fitResultsTMP = NonlinearModelFit[tmpToFitNoise, myFitMono[x1],
  {{delayMono, delayGuess}, {ampMono, ampGuess}, {tau1Mono,
    caAdjustedTau1Guess}}, x1, MaxIterations → myMaxIterations];
fitResultMono = fitResultsTMP[{"BestFitParameters"}];
simParamNoiseC10[[1, noiseN]] = 0; (*not used anymore*)
simParamNoiseC10[[2, noiseN]] =
 fitResultsTMP["ANOVATableSumsOfSquares"][[2]];
(*Print[fitResultsTMP["ANOVATableSumsOfSquares"]];*)
simParamNoiseC10[[3, noiseN]] = (delayMono /. fitResultMono) [[1]];
simParamNoiseC10[[4, noiseN]] = ((ampMono /. fitResultMono)) [[1]];
simParamNoiseC10[[5, noiseN]] = (1 / (tau1Mono /. fitResultMono)) [[1]];
(*fit bi-exp*)
fitResultsTMP =
 NonlinearModelFit[tmpToFitNoise, myFitBi[x1],  {{delay, delayGuess},
   {amp, ampGuess}, {amp1, amp1Guess}, {tau1, caAdjustedTau1Guess},
   {tau2, caAdjustedTau2Guess}}, x1, MaxIterations → myMaxIterations];
fitResultBi = fitResultsTMP[{"BestFitParameters"}];
simParamNoiseC10[[6, noiseN]] = simParamNoiseC10[[2, noiseN]] /
  fitResultsTMP["ANOVATableSumsOfSquares"][[2]];
simParamNoiseC10[[7, noiseN]] = (delay /. fitResultBi) [[1]];
simParamNoiseC10[[8, noiseN]] = ((amp /. fitResultBi)) [[1]];
simParamNoiseC10[[9, noiseN]] = ((amp amp1 /. fitResultBi)) [[1]];
(*relative amp1*)
simParamNoiseC10[[10, noiseN]] = (1 / (tau1 /. fitResultBi)) [[1]];
simParamNoiseC10[[11, noiseN]] = (1 / (tau2 /. fitResultBi)) [[1]];
(*merge*)
If[
 (*to use the bi-exp fit, the following cirteria should be fullfilled:*)
 (*chi2 should improve(=decrease) by >4%*)
 (simParamNoiseC10[[6, noiseN]] > 1.04)
  &&
  (*tau1 and tau2 of bi fit should be factor of >3 different*)
  ((simParamNoiseC10[[11, noiseN]] / simParamNoiseC10[[10, noiseN]]) < 3.)
  &&
  (*relative amplitude of 1st component should be > 5% *)
  ((( amp1 /. fitResultBi)) [[1]] > 0.05)
  &&
  (*relative amplitude of 1st component should be < 95% *)
  ((( amp1 /. fitResultBi)) [[1]] < 0.95)
 ,
 (*take bi*)
 Print["take bi"];
 simParamNoiseC10[[12, noiseN]] = simParamNoiseC10[[7, noiseN]];
 (*delay*)
```

```
    simParamNoiseC10[[13, noiseN]] = simParamNoiseC10[[8, noiseN]];
    (*amp*)
    simParamNoiseC10[[14, noiseN]] = simParamNoiseC10[[9, noiseN]];
    (*amp1*)
    simParamNoiseC10[[15, noiseN]] = simParamNoiseC10[[10, noiseN]];
    (*tau1*)
    simParamNoiseC10[[16, noiseN]] = simParamNoiseC10[[11, noiseN]];(*tau2*)
    ,
    (*take mono*)
    Print["take mono"];
    simParamNoiseC10[[12, noiseN]] = simParamNoiseC10[[3, noiseN]];
    (*delay*)
    simParamNoiseC10[[13, noiseN]] = simParamNoiseC10[[4, noiseN]];
    (*amp*)
    simParamNoiseC10[[14, noiseN]] = NaN; (*amp1*)
    simParamNoiseC10[[15, noiseN]] = simParamNoiseC10[[5, noiseN]];
    (*tau1*)
    simParamNoiseC10[[16, noiseN]] = NaN;(*tau2*)
  ];
];
(*plot last example of the noise loop*)
gr1 = ListPlot[tmpToFitNoise, PlotRange → All, PlotStyle → Black];
gr2 = Plot[myFitMono[x1] /. fitResultMono,
   {x1, cursorStart, cursorEnd}, PlotRange → All, PlotStyle → {Blue, Dashed}];
gr3 = Plot[myFitBi[x1] /. fitResultBi, {x1, cursorStart, cursorEnd},
   PlotRange → All, PlotStyle → {Green, Dashed}];
Show[gr1, gr2, gr3, PlotRange → All ] // Print;
If[exportYes == 1,
 Export["withinLoop r" <> ToString[r] <> " Ca" <>
   ToString[1*^6 myCaNow] <> " C10 data.txt", tmpToFitNoise, "Table"];
 toExport = Table[{t, (myFitMono[t] /. fitResultMono) [[1]]},
   {t, cursorStart, cursorEnd, dtOfPlotsForExport}];
 Export["withinLoop r" <> ToString[r] <> " Ca" <> ToString[1*^6 myCaNow] <>
   " C10 fitMono.txt", toExport, "Table"];
 toExport = Table[{t, (myFitBi[t] /. fitResultBi) [[1]]},
   {t, cursorStart, cursorEnd, dtOfPlotsForExport}];
 Export["withinLoop r" <> ToString[r] <> " Ca" <> ToString[1*^6 myCaNow] <>
   " C10 fitBi.txt", toExport, "Table"];
];
noiseN = noiseRepeats; (*fit parameter of last noisetrace*)
Print["Mono: chi2 = ", simParamNoiseC10[[2, noiseN]], "     d = ",
 simParamNoiseC10[[2, noiseN]], "     a = ", simParamNoiseC10[[4, noiseN]],
 "     t = ", 1/simParamNoiseC10[[5, noiseN]]];
Print["Bi: ratio chi2Mono/chi2Bi = ", simParamNoiseC10[[6, noiseN]],
 "     d = ", simParamNoiseC10[[7, noiseN]], "     a = ",
 simParamNoiseC10[[8, noiseN]], "     a1 = ", simParamNoiseC10[[9, noiseN]],
 "     t1 =", 1/simParamNoiseC10[[10, noiseN]],
```

```
 "     t2 = ", 1/simParamNoiseC10[[11, noiseN]]];
(*average fit results*)
For[p = 1, p ≤ numberOfFitParamToBeSaved, p += 1,
 simParamMedianC10[[p, r]] =
  Median[simParamNoiseC10[[p, All]] /. NaN → Sequence[]];
 simParamQuantile1C10[[p, r]] = Quantile[
   simParamNoiseC10[[p, All]] /. NaN → Sequence[], myQuantile1];
 simParamQuantile2C10[[p, r]] = Quantile[
   simParamNoiseC10[[p, All]] /. NaN → Sequence[], myQuantile2];
];

(*if tau1 merge > 10 ms, use long trace for fitting*)
If[(1/simParamMedianC10[[15, r]]) > 0.01,
 Print["Long trace was used for fitting."];
 For[noiseN = 1, noiseN ≤ noiseRepeats, noiseN += 1,
  tmpToFitNoiseLong = Transpose[{tmpToFitLong[[All, 1]],
     # + RandomVariate[NormalDistribution[0, myNoiseLong]] & /@
      tmpToFitLong[[All, 2]]}];
  (*fit mono-exp to Long trace*)
  fitResultsTMP = NonlinearModelFit[tmpToFitNoiseLong, myFitMono[x1],
    {{delayMono, delayGuess}, {ampMono, ampGuess}, {tau1Mono,
      caAdjustedTau1Guess}}, x1, MaxIterations → myMaxIterations];
  fitResultLongMono = fitResultsTMP[{"BestFitParameters"}];
  simParamNoiseC10[[2, noiseN]] =
   fitResultsTMP["ANOVATableSumsOfSquares"][[2]];
  (*Print[fitResultsTMP["ANOVATableSumsOfSquares"]];*)
  simParamNoiseC10[[3, noiseN]] = (delayMono /. fitResultLongMono)[[1]];
  simParamNoiseC10[[4, noiseN]] = ((ampMono /. fitResultLongMono))[[1]];
  simParamNoiseC10[[5, noiseN]] = (1/(tau1Mono /. fitResultLongMono))[[1]];
  (*fit bi-exp*)
  fitResultsTMP =
   NonlinearModelFit[tmpToFitNoiseLong, myFitBi[x1], {{delay, delayGuess},
     {amp, ampGuess}, {amp1, amp1Guess}, {tau1, caAdjustedTau1Guess},
     {tau2, caAdjustedTau2Guess}}, x1, MaxIterations → myMaxIterations];
  fitResultLongBi = fitResultsTMP[{"BestFitParameters"}];
  simParamNoiseC10[[6, noiseN]] = simParamNoiseC10[[2, noiseN]] /
    fitResultsTMP["ANOVATableSumsOfSquares"][[2]];
  simParamNoiseC10[[7, noiseN]] = (delay /. fitResultLongBi)[[1]];
  simParamNoiseC10[[8, noiseN]] = ((amp /. fitResultLongBi))[[1]];
  simParamNoiseC10[[9, noiseN]] = ((amp amp1 /. fitResultLongBi))[[1]];
  (*relative amp1*)
  simParamNoiseC10[[10, noiseN]] = (1/(tau1 /. fitResultLongBi))[[1]];
  simParamNoiseC10[[11, noiseN]] = (1/(tau2 /. fitResultLongBi))[[1]];
  (*merge*)
  If[
   (*to use the bi-exp fit,
```

```
    the following cirteria should be fullfilled:*)
    (*chi2 should improve(=decrease) by >4%*)
    (simParamNoiseC10[[6, noiseN]] > 1.04)
     &&
     (*tau1 and tau2 of bi fit should be factor of >3 different*)
     ((simParamNoiseC10[[11, noiseN]] / simParamNoiseC10[[10, noiseN]]) < 3.)
     &&
     (*relative amplitude of 1st component should be > 5% *)
     (((amp1 /. fitResultLongBi))[[1]] > 0.05)
     &&
     (*relative amplitude of 1st component should be < 95% *)
     (((amp1 /. fitResultLongBi))[[1]] < 0.95)
    ,
    (*take bi*)
    Print["take bi"];
    simParamNoiseC10[[12, noiseN]] = simParamNoiseC10[[7, noiseN]];
    (*delay*)
    simParamNoiseC10[[13, noiseN]] = simParamNoiseC10[[8, noiseN]];
    (*amp*)
    simParamNoiseC10[[14, noiseN]] = simParamNoiseC10[[9, noiseN]];
    (*amp1*)
    simParamNoiseC10[[15, noiseN]] = simParamNoiseC10[[10, noiseN]];
    (*tau1*)
    simParamNoiseC10[[16, noiseN]] = simParamNoiseC10[[11, noiseN]]; (*tau2*)
    ,
    (*take mono*)
    Print["take mono"];
    simParamNoiseC10[[12, noiseN]] = simParamNoiseC10[[3, noiseN]];
    (*delay*)
    simParamNoiseC10[[13, noiseN]] = simParamNoiseC10[[4, noiseN]];
    (*amp*)
    simParamNoiseC10[[14, noiseN]] = NaN; (*amp1*)
    simParamNoiseC10[[15, noiseN]] = simParamNoiseC10[[5, noiseN]];
    (*tau1*)
    simParamNoiseC10[[16, noiseN]] = NaN; (*tau2*)
  ];
];
(*plot last example of the noise loop*)
gr1 = ListPlot[tmpToFitNoiseLong, PlotRange → All, PlotStyle → Black];
gr2 = Plot[myFitMono[x1] /. fitResultLongMono, {x1, cursorStart,
     cursorEndLong}, PlotRange → All, PlotStyle → {Blue, Dashed}];
gr3 = Plot[myFitBi[x1] /. fitResultLongBi, {x1, cursorStart, cursorEndLong},
    PlotRange → All, PlotStyle → {Green, Dashed}];
Show[gr1, gr2, gr3, PlotRange → All ] // Print;
If[exportYes == 1,
  Export["withinLoop r" <> ToString[r] <> " Ca" <> ToString[1*^6 myCaNow] <>
     " C10 dataLong.txt", tmpToFitNoiseLong, "Table"];
```

```
    toExport = Table[{t, (myFitMono[t] /. fitResultLongMono)[[1]]},
      {t, cursorStart, cursorEndLong, dtOfPlotsForExport}];
    Export["withinLoop r" <> ToString[r] <> " Ca" <> ToString[1*^6 myCaNow] <>
      " C10 fitLongMono.txt", toExport, "Table"];
    toExport = Table[{t, (myFitBi[t] /. fitResultLongBi)[[1]]},
      {t, cursorStart, cursorEndLong, dtOfPlotsForExport}];
    Export["withinLoop r" <> ToString[r] <> " Ca" <> ToString[1*^6 myCaNow] <>
      " C10 fitLongBi.txt", toExport, "Table"];

  ];
  noiseN = noiseRepeats; (*fit parameter of last noisetrace*)
  Print["Mono: chi2 = ", simParamNoiseC10[[2, noiseN]], "     d = ",
   simParamNoiseC10[[3, noiseN]], "     a = ", simParamNoiseC10[[4, noiseN]],
   "     t = ", 1/simParamNoiseC10[[5, noiseN]]];
  Print["Bi: ratio chi2Mono/chi2Bi = ", simParamNoiseC10[[6, noiseN]],
   "     d = ", simParamNoiseC10[[7, noiseN]], "     a = ",
   simParamNoiseC10[[8, noiseN]], "     a1 = ", simParamNoiseC10[[9, noiseN]],
   "     t1 =", 1/simParamNoiseC10[[10, noiseN]],
   "     t2 = ", 1/simParamNoiseC10[[11, noiseN]]];
  (*average fit results*)
  For[p = 1, p ≤ numberOfFitParamToBeSaved, p += 1,
   simParamMedianC10[[p, r]] =
    Median[simParamNoiseC10[[p, All]] /. NaN → Sequence[]];
   simParamQuantile1C10[[p, r]] = Quantile[
     simParamNoiseC10[[p, All]] /. NaN → Sequence[], myQuantile1];
   simParamQuantile2C10[[p, r]] = Quantile[
     simParamNoiseC10[[p, All]] /. NaN → Sequence[], myQuantile2];
  ];
 ];

 , (*else: signal is not large enough*)
 For[p = 1, p ≤ numberOfFitParamToBeSaved, p += 1,
   simParamMedianC10[[p, r]] = {};
   simParamQuantile1C10[[p, r]] = {};
   simParamQuantile2C10[[p, r]] = {};
  ];
];

(*---------    Fitting of data,
saving of results, and plotting-----------------*)
(*---------------------------------------------------------------------
                                  ----*)
(*------------------------------------    D     -------------------------
                       --*)
(*---------------------------------------------------------------------
                                  ----*)
Print[
```

```
"-----------------------------------------------------------------------
  D"];
(*check that signal is large enough relative to noise to obtain
 useful fit results. If not, do not do fitting and set everything to {}*)
If[simParamNv[[5, r]] > signalToNoiseRatioD myNoiseD, (*add noise
  several times and do the fitting and than average the results*)
 For[noiseN = 1, noiseN ≤ noiseRepeats, noiseN += 1,
  (*Print[noiseN];*)
  tmpToFitNoise = Transpose[{tmpToFitD[[All, 1]],
     # + RandomVariate[NormalDistribution[0, myNoiseD]] & /@
      tmpToFitD[[All, 2]]}];
  (*fit mono-exp*)
  fitResultsTMP = NonlinearModelFit[tmpToFitNoise, myFitMono[x1],
    {{delayMono, delayGuess}, {ampMono, ampGuess}, {tau1Mono,
      caAdjustedTau1Guess}}, x1, MaxIterations → myMaxIterations];
  fitResultMono = fitResultsTMP[{"BestFitParameters"}];
  simParamNoiseD[[1, noiseN]] = 0; (*not used anymore*)
  simParamNoiseD[[2, noiseN]] =
   fitResultsTMP["ANOVATableSumsOfSquares"][[2]];
  (*Print[fitResultsTMP["ANOVATableSumsOfSquares"]];*)
  simParamNoiseD[[3, noiseN]] = (delayMono /. fitResultMono)[[1]];
  simParamNoiseD[[4, noiseN]] = ((ampMono /. fitResultMono))[[1]];
  simParamNoiseD[[5, noiseN]] = (1 / (tau1Mono /. fitResultMono))[[1]];
  (*fit bi-exp*)
  fitResultsTMP =
   NonlinearModelFit[tmpToFitNoise, myFitBi[x1], {{delay, delayGuess},
     {amp, ampGuess}, {amp1, amp1Guess}, {tau1, caAdjustedTau1Guess},
     {tau2, caAdjustedTau2Guess}}, x1, MaxIterations → myMaxIterations];
  fitResultBi = fitResultsTMP[{"BestFitParameters"}];
  simParamNoiseD[[6, noiseN]] = simParamNoiseD[[2, noiseN]] /
    fitResultsTMP["ANOVATableSumsOfSquares"][[2]];
  simParamNoiseD[[7, noiseN]] = (delay /. fitResultBi)[[1]];
  simParamNoiseD[[8, noiseN]] = ((amp /. fitResultBi))[[1]];
  simParamNoiseD[[9, noiseN]] = ((amp amp1 /. fitResultBi))[[1]];
  (*relative amp1*)
  simParamNoiseD[[10, noiseN]] = (1 / (tau1 /. fitResultBi))[[1]];
  simParamNoiseD[[11, noiseN]] = (1 / (tau2 /. fitResultBi))[[1]];
  (*merge*)
  If[
   (*to use the bi-exp fit, the following cirteria should be fullfilled:*)
   (*chi2 should improve(=decrease) by >4%*)
   (simParamNoiseD[[6, noiseN]] > 1.04)
    &&
    (*tau1 and tau2 of bi fit should be factor of >3 different*)
    ((simParamNoiseD[[11, noiseN]] / simParamNoiseD[[10, noiseN]]) < 3.)
    &&
```

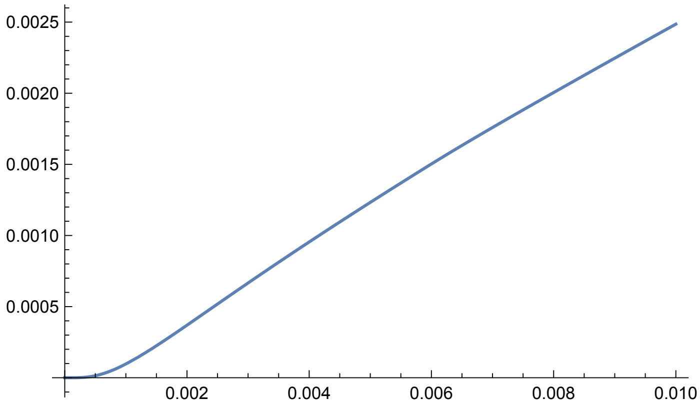
```
    (*relative amplitude of 1st component should be > 5% *)
    (((amp1 /. fitResultBi)) [[1]] > 0.05)
    &&
    (*relative amplitude of 1st component should be < 95% *)
    (((amp1 /. fitResultBi)) [[1]] < 0.95)
   ,
   (*take bi*)
   Print["take bi"];
   simParamNoiseD[[12, noiseN]] = simParamNoiseD[[7, noiseN]];
   (*delay*)
   simParamNoiseD[[13, noiseN]] = simParamNoiseD[[8, noiseN]];
   (*amp*)
   simParamNoiseD[[14, noiseN]] = simParamNoiseD[[9, noiseN]];
   (*amp1*)
   simParamNoiseD[[15, noiseN]] = simParamNoiseD[[10, noiseN]];
   (*tau1*)
   simParamNoiseD[[16, noiseN]] = simParamNoiseD[[11, noiseN]];(*tau2*)
   ,
   (*take mono*)
   Print["take mono"];
   simParamNoiseD[[12, noiseN]] = simParamNoiseD[[3, noiseN]];
   (*delay*)
   simParamNoiseD[[13, noiseN]] = simParamNoiseD[[4, noiseN]];
   (*amp*)
   simParamNoiseD[[14, noiseN]] = NaN; (*amp1*)
   simParamNoiseD[[15, noiseN]] = simParamNoiseD[[5, noiseN]];
   (*tau1*)
   simParamNoiseD[[16, noiseN]] = NaN;(*tau2*)
  ];
 ];
(*plot last example of the noise loop*)
gr1 = ListPlot[tmpToFitNoise, PlotRange → All, PlotStyle → Black];
gr2 = Plot[myFitMono[x1] /. fitResultMono,
   {x1, cursorStart, cursorEnd}, PlotRange → All, PlotStyle → {Blue, Dashed}];
gr3 = Plot[myFitBi[x1] /. fitResultBi, {x1, cursorStart, cursorEnd},
   PlotRange → All, PlotStyle → {Green, Dashed}];
Show[gr1, gr2, gr3, PlotRange → All ] // Print;
If[exportYes == 1,
 Export["withinLoop r" <> ToString[r] <> " Ca" <>
   ToString[1*^6 myCaNow] <> " D data.txt", tmpToFitNoise, "Table"];
 toExport = Table[{t, (myFitMono[t] /. fitResultMono) [[1]]},
   {t, cursorStart, cursorEnd, dtOfPlotsForExport}];
 Export["withinLoop r" <> ToString[r] <> " Ca" <> ToString[1*^6 myCaNow] <>
   " D fitMono.txt", toExport, "Table"];
 toExport = Table[{t, (myFitBi[t] /. fitResultBi) [[1]]},
   {t, cursorStart, cursorEnd, dtOfPlotsForExport}];
 Export["withinLoop r" <> ToString[r] <> " Ca" <>
```

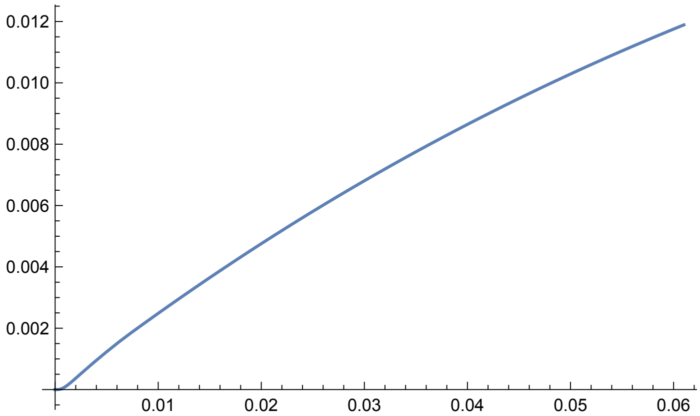
```
    ToString[1*^6 myCaNow] <> " D fitBi.txt", toExport, "Table"];
  ];
noiseN = noiseRepeats; (*fit parameter of last noisetrace*)
Print["Mono: chi2 = ", simParamNoiseD[[2, noiseN]],
 "     d = ", simParamNoiseD[[2, noiseN]], "     a = ",
 simParamNoiseD[[4, noiseN]], "     t = ", 1/simParamNoiseD[[5, noiseN]]];
Print["Bi: ratio chi2Mono/chi2Bi = ", simParamNoiseD[[6, noiseN]],
 "     d = ", simParamNoiseD[[7, noiseN]], "     a = ",
 simParamNoiseD[[8, noiseN]], "     a1 = ", simParamNoiseD[[9, noiseN]],
 "     t1 =", 1/simParamNoiseD[[10, noiseN]],
 "     t2 = ", 1/simParamNoiseD[[11, noiseN]]];
(*average fit results*)
For[p = 1, p ≤ numberOfFitParamToBeSaved, p += 1,
 simParamMedianD[[p, r]] =
  Median[simParamNoiseD[[p, All]] /. NaN → Sequence[]];
 simParamQuantile1D[[p, r]] = Quantile[
   simParamNoiseD[[p, All]] /. NaN → Sequence[], myQuantile1];
 simParamQuantile2D[[p, r]] = Quantile[
   simParamNoiseD[[p, All]] /. NaN → Sequence[], myQuantile2];
];

(*if tau1 merge > 10 ms, use long trace for fitting*)
If[(1/simParamMedianD[[15, r]]) > 0.01,
 Print["Long trace was used for fitting."];
 For[noiseN = 1, noiseN ≤ noiseRepeats, noiseN += 1,
  tmpToFitNoiseLong = Transpose[{tmpToFitLong[[All, 1]],
      # + RandomVariate[NormalDistribution[0, myNoiseLong]] & /@
       tmpToFitLong[[All, 2]]}];
  (*fit mono-exp to Long trace*)
  fitResultsTMP = NonlinearModelFit[tmpToFitNoiseLong, myFitMono[x1],
     {{delayMono, delayGuess}, {ampMono, ampGuess}, {tau1Mono,
       caAdjustedTau1Guess}}, x1, MaxIterations → myMaxIterations];
  fitResultLongMono = fitResultsTMP[{"BestFitParameters"}];
  simParamNoiseD[[2, noiseN]] =
   fitResultsTMP["ANOVATableSumsOfSquares"][[2]];
  (*Print[fitResultsTMP["ANOVATableSumsOfSquares"]];*)
  simParamNoiseD[[3, noiseN]] = (delayMono /. fitResultLongMono)[[1]];
  simParamNoiseD[[4, noiseN]] = ((ampMono /. fitResultLongMono))[[1]];
  simParamNoiseD[[5, noiseN]] = (1/(tau1Mono /. fitResultLongMono))[[1]];
  (*fit bi-exp*)
  fitResultsTMP =
   NonlinearModelFit[tmpToFitNoiseLong, myFitBi[x1], {{delay, delayGuess},
     {amp, ampGuess}, {amp1, amp1Guess}, {tau1, caAdjustedTau1Guess},
     {tau2, caAdjustedTau2Guess}}, x1, MaxIterations → myMaxIterations];
  fitResultLongBi = fitResultsTMP[{"BestFitParameters"}];
  simParamNoiseD[[6, noiseN]] = simParamNoiseD[[2, noiseN]]/
    fitResultsTMP["ANOVATableSumsOfSquares"][[2]];
```

```
simParamNoiseD[[7, noiseN]] = (delay /. fitResultLongBi) [[1]];
simParamNoiseD[[8, noiseN]] = ((amp /. fitResultLongBi)) [[1]];
simParamNoiseD[[9, noiseN]] = ((amp amp1 /. fitResultLongBi)) [[1]];
(*relative amp1*)
simParamNoiseD[[10, noiseN]] = (1/ (tau1 /. fitResultLongBi)) [[1]];
simParamNoiseD[[11, noiseN]] = (1/ (tau2 /. fitResultLongBi)) [[1]];
(*merge*)
If[
 (*to use the bi-exp fit,
 the following cirteria should be fullfilled:*)
 (*chi2 should improve(=decrease) by >4%*)
 (simParamNoiseD[[6, noiseN]] > 1.04)
  &&
  (*tau1 and tau2 of bi fit should be factor of >3 different*)
  ((simParamNoiseD[[11, noiseN]] / simParamNoiseD[[10, noiseN]]) < 3.)
  &&
  (*relative amplitude of 1st component should be > 5% *)
  ((( amp1 /. fitResultLongBi)) [[1]] > 0.05)
  &&
  (*relative amplitude of 1st component should be < 95% *)
  ((( amp1 /. fitResultLongBi)) [[1]] < 0.95)
 ,
 (*take bi*)
 Print["take bi"];
 simParamNoiseD[[12, noiseN]] = simParamNoiseD[[7, noiseN]];
 (*delay*)
 simParamNoiseD[[13, noiseN]] = simParamNoiseD[[8, noiseN]];
 (*amp*)
 simParamNoiseD[[14, noiseN]] = simParamNoiseD[[9, noiseN]];
 (*amp1*)
 simParamNoiseD[[15, noiseN]] = simParamNoiseD[[10, noiseN]];
 (*tau1*)
 simParamNoiseD[[16, noiseN]] = simParamNoiseD[[11, noiseN]];(*tau2*)
 ,
 (*take mono*)
 Print["take mono"];
 simParamNoiseD[[12, noiseN]] = simParamNoiseD[[3, noiseN]];
 (*delay*)
 simParamNoiseD[[13, noiseN]] = simParamNoiseD[[4, noiseN]];
 (*amp*)
 simParamNoiseD[[14, noiseN]] = NaN; (*amp1*)
 simParamNoiseD[[15, noiseN]] = simParamNoiseD[[5, noiseN]];
 (*tau1*)
 simParamNoiseD[[16, noiseN]] = NaN;(*tau2*)
];
];
```

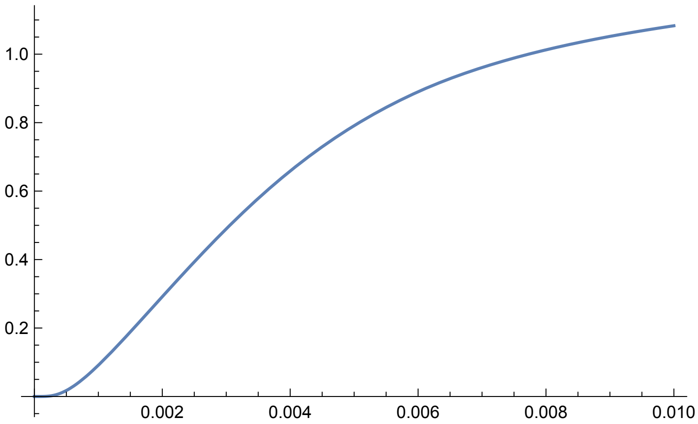
```
 (*plot last example of the noise loop*)
 gr1 = ListPlot[tmpToFitNoiseLong, PlotRange → All, PlotStyle → Black];
 gr2 = Plot[myFitMono[x1] /. fitResultLongMono, {x1, cursorStart,
    cursorEndLong}, PlotRange → All, PlotStyle → {Blue, Dashed}];
 gr3 = Plot[myFitBi[x1] /. fitResultLongBi, {x1, cursorStart, cursorEndLong},
   PlotRange → All, PlotStyle → {Green, Dashed}];
 Show[gr1, gr2, gr3, PlotRange → All ] // Print;
 If[exportYes == 1,
  Export["withinLoop r" <> ToString[r] <> " Ca" <> ToString[1*^6 myCaNow] <>
    " D dataLong.txt", tmpToFitNoiseLong, "Table"];
  toExport = Table[{t, (myFitMono[t] /. fitResultLongMono) [[1]]},
    {t, cursorStart, cursorEndLong, dtOfPlotsForExport}];
  Export["withinLoop r" <> ToString[r] <> " Ca" <> ToString[1*^6 myCaNow] <>
    " D fitLongMono.txt", toExport, "Table"];
  toExport = Table[{t, (myFitBi[t] /. fitResultLongBi) [[1]]},
    {t, cursorStart, cursorEndLong, dtOfPlotsForExport}];
  Export["withinLoop r" <> ToString[r] <> " Ca" <> ToString[1*^6 myCaNow] <>
    " D fitLongBi.txt", toExport, "Table"];

 ];
 noiseN = noiseRepeats; (*fit parameter of last noisetrace*)
 Print["Mono: chi2 = ", simParamNoiseD[[2, noiseN]],
  "     d = ", simParamNoiseD[[3, noiseN]], "     a = ",
  simParamNoiseD[[4, noiseN]], "     t = ", 1/simParamNoiseD[[5, noiseN]]];
 Print["Bi: ratio chi2Mono/chi2Bi = ", simParamNoiseD[[6, noiseN]],
  "     d = ", simParamNoiseD[[7, noiseN]], "     a = ",
  simParamNoiseD[[8, noiseN]], "     a1 = ", simParamNoiseD[[9, noiseN]],
  "     t1 =", 1/simParamNoiseD[[10, noiseN]],
  "     t2 = ", 1/simParamNoiseD[[11, noiseN]]];
 (*average fit results*)
 For[p = 1, p ≤ numberOfFitParamToBeSaved, p += 1,
  simParamMedianD[[p, r]] =
   Median[simParamNoiseD[[p, All]] /. NaN → Sequence[]];
  simParamQuantile1D[[p, r]] = Quantile[
    simParamNoiseD[[p, All]] /. NaN → Sequence[], myQuantile1];
  simParamQuantile2D[[p, r]] = Quantile[
    simParamNoiseD[[p, All]] /. NaN → Sequence[], myQuantile2];
 ];
];

, (*else: signal is not large enough*)
For[p = 1, p ≤ numberOfFitParamToBeSaved, p += 1,
  simParamMedianD[[p, r]] = {};
  simParamQuantile1D[[p, r]] = {};
  simParamQuantile2D[[p, r]] = {};
 ];
```

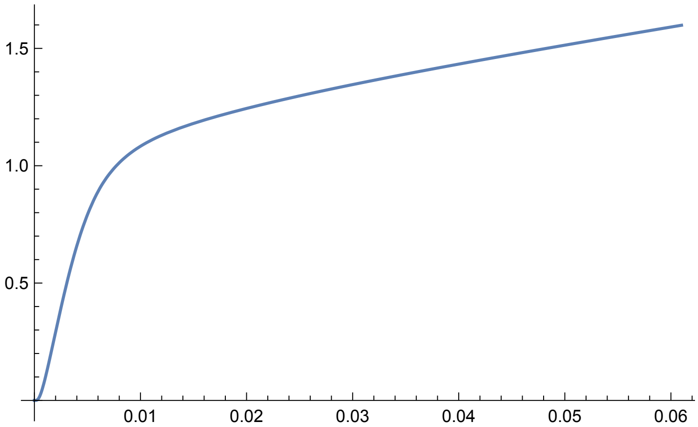
```
  ];
 ];
-----------------------------------------------------------------------------------
  -------------------------
----------------    Ca = 0.703073
 uM          --------------------------------------------------------
-----------------------------------------------------------------------------------
  -------------------------
```






```
---------------------------------------------------------------------------- C5
---------------------------------------------------------------------------- C10
---------------------------------------------------------------------------- D
-----------------------------------------------------------------------------------
  -------------------------
----------------    Ca = 4.79194
 uM          --------------------------------------------------------
-----------------------------------------------------------------------------------
  -------------------------
```





-- C5

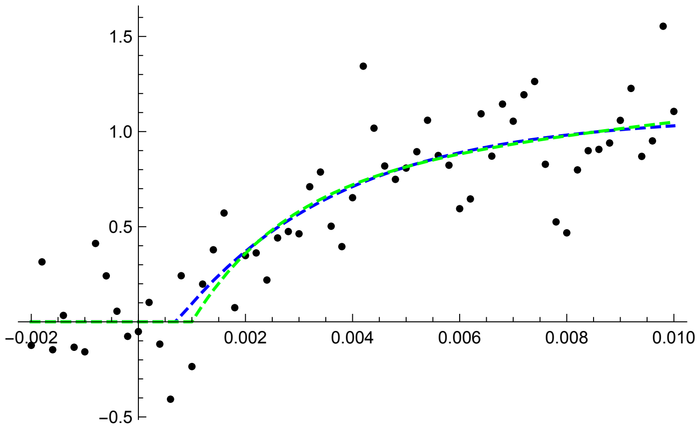
NonlinearModelFit: Failed to converge to the requested accuracy or precision within 100 iterations.

take mono

NonlinearModelFit: Failed to converge to the requested accuracy or precision within 100 iterations.

take mono

take mono



Mono: chi2 = 2.96309 d = 2.96309 a = 1.08303 t = 0.00307561

Bi: ratio chi2Mono/chi2Bi = 1.00705 d = 0.001 a =
2.66801 a1 = 0.753421 t1 =0.00177641 t2 = 0.0525808

Quantile: Argument {} should be a non−empty list.

Quantile: Argument {} should be a non−empty list.

Quantile: Argument {} should be a non−empty list.

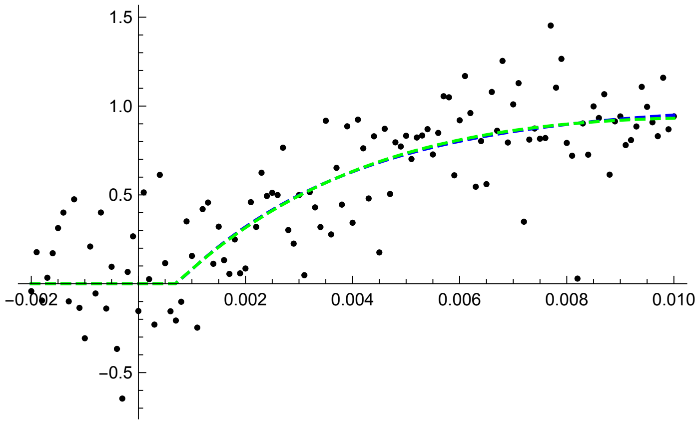
General: Further output of Quantile::empt will be suppressed during this calculation.

-- C10

take mono

take mono

take mono



Mono: chi2 = 7.13307 d = 7.13307 a = 1.01403 t = 0.00337532

Bi: ratio chi2Mono/chi2Bi = 1.00335 d = 0.0007 a =
0.537452 a1 = 2.0822 t1 =0.00508527 t2 = 0.0123731

-- D

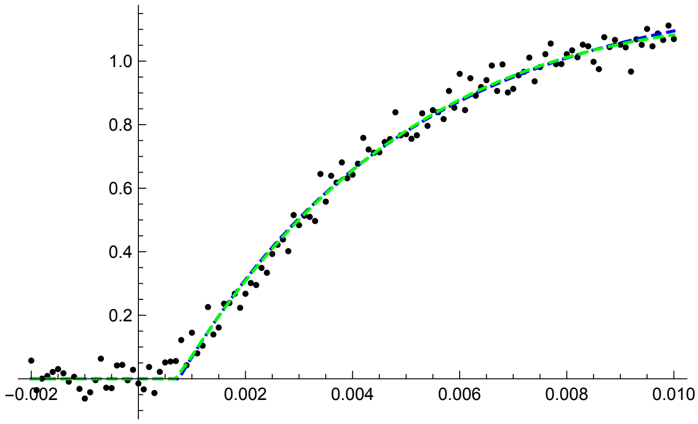
NonlinearModelFit: Failed to converge to the requested accuracy or precision within 100 iterations.

General: Further output of NonlinearModelFit::cvmit will be suppressed during this calculation.

take mono

take mono

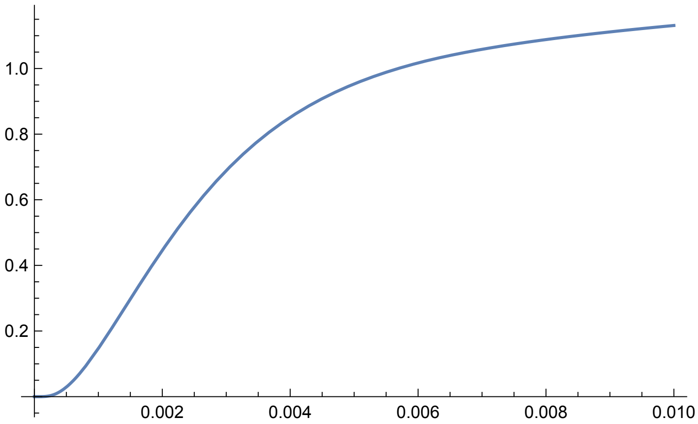
take mono

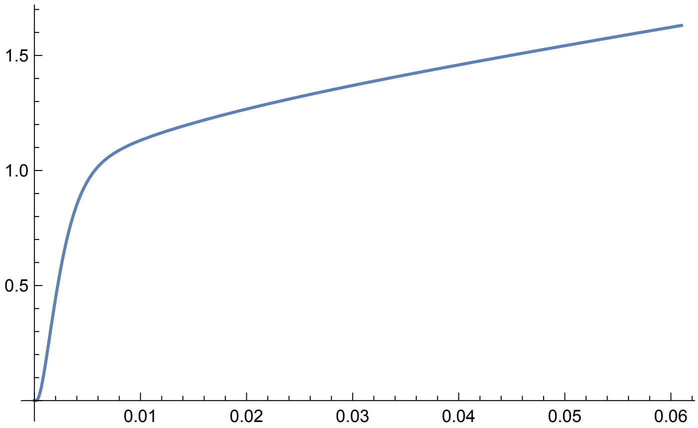


Mono: chi2 = 0.175917 d = 0.175917 a = 1.23782 t = 0.00428099

Bi: ratio chi2Mono/chi2Bi = 1.04785 d = 0.000729688 a =
0.531413 a1 = 3.07318 t1 =0.00683306 t2 = 0.0145241

---------------- Ca = 5.53376
uM --



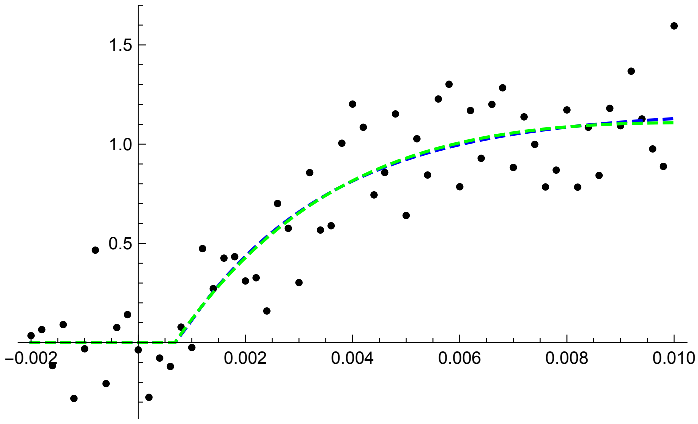


--- C5

take mono

take mono

take mono



Mono: chi2 = 2.48472   d = 2.48472   a = 1.16827   t = 0.00274828

Bi: ratio chi2Mono/chi2Bi = 1.00275   d = 0.000686306   a = 0.918035   a1 = 5.28931   t1 =0.00467827   t2 = 0.00594278

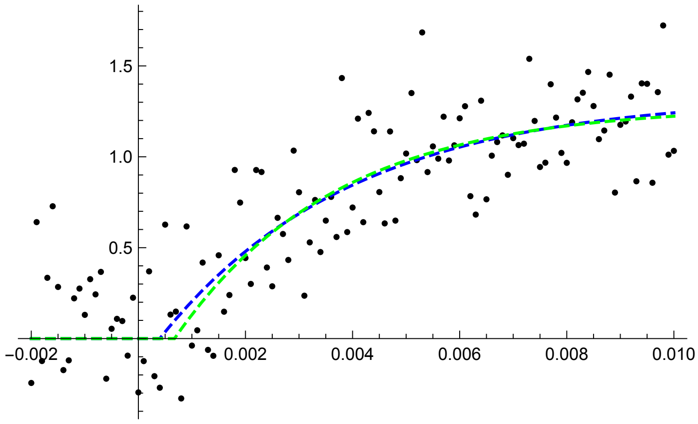
--- C10

take mono

··· NonlinearModelFit: The step size in the search has become less than the tolerance prescribed by the PrecisionGoal option, but the gradient is larger than the tolerance specified by the AccuracyGoal option. There is a possibility that the method has stalled at a point that is not a local minimum.

take mono

take mono



Mono: chi2 = 8.54384   d = 8.54384   a = 1.33406   t = 0.00359141

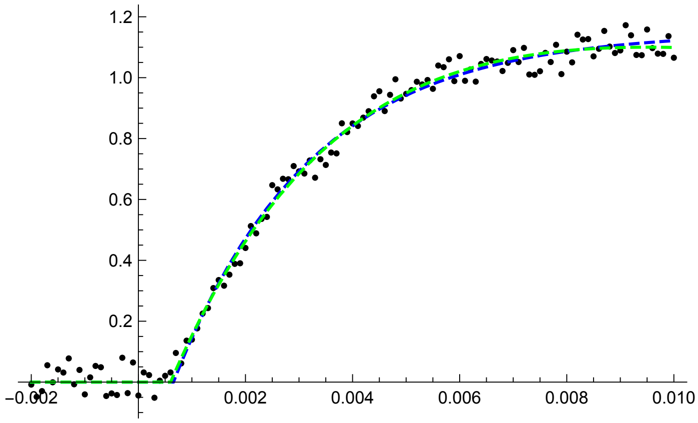
Bi: ratio chi2Mono/chi2Bi = 1.00013   d = 0.000674773   a = 1.2702   a1 = 1.30972   t1 =0.0030363   t2 = 0.00880195

--- D

take mono

take mono

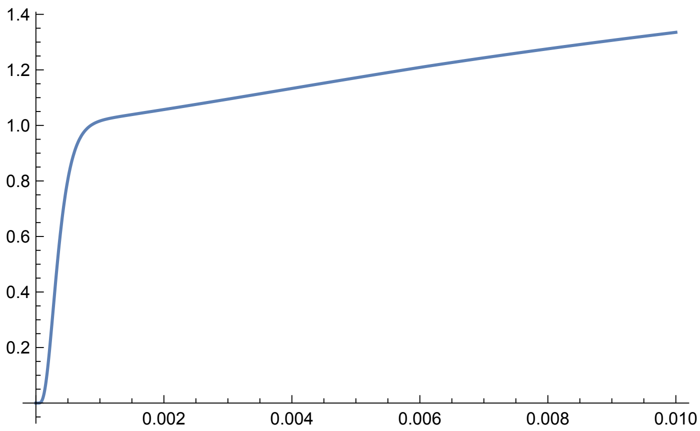
take mono

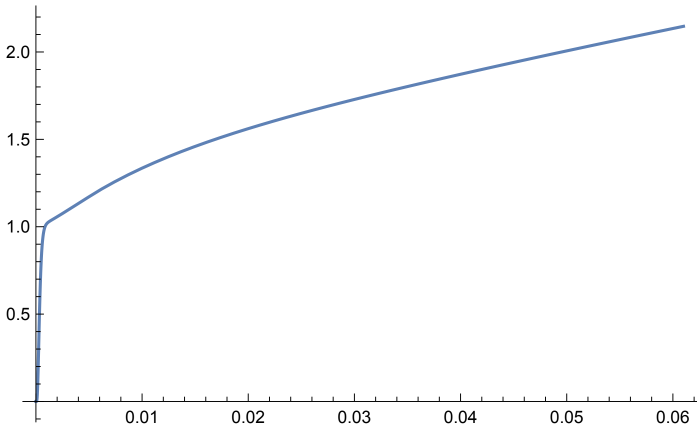


Mono: chi2 = 0.163452   d = 0.163452   a = 1.15075   t = 0.00254525

Bi: ratio chi2Mono/chi2Bi = 1.05765   d = 0.000601563   a = 0.874688   a1 = 3.59971   t1 =0.0043296   t2 = 0.00645371

----------------   Ca = 25.7412 uM   ---





--- C5

NonlinearModelFit: The step size in the search has become less than the tolerance prescribed by the PrecisionGoal option, but the gradient is larger than the tolerance specified by the AccuracyGoal option. There is a possibility that the method has stalled at a point that is not a local minimum.

General: Exp[−736.868] is too small to represent as a normalized machine number; precision may be lost.

General: Exp[−789.463] is too small to represent as a normalized machine number; precision may be lost.

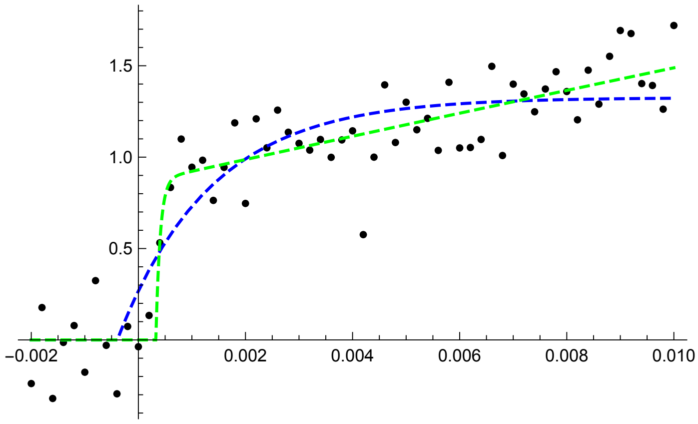
General: Exp[−842.058] is too small to represent as a normalized machine number; precision may be lost.

General: Further output of General::munfl will be suppressed during this calculation.

take bi

take bi

take bi



Mono: chi2 = 2.68841   d = 2.68841   a = 1.32554   t = 0.00173961

Bi: ratio chi2Mono/chi2Bi = 1.50993   d = 0.000326227   a = 13.4845   a1 = 0.879253   t1 =0.0000811097   t2 = 0.195227

-- C10

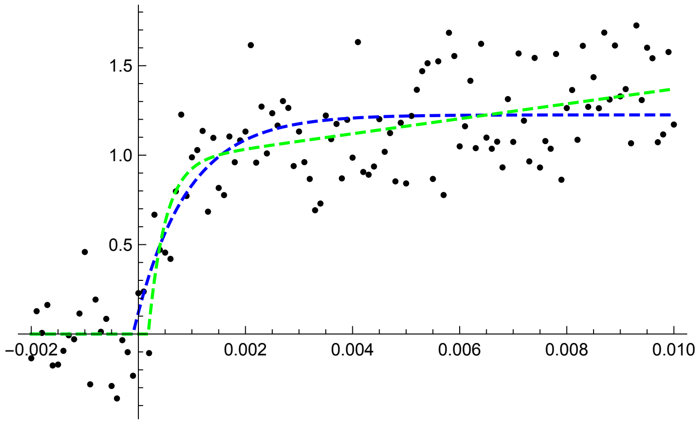
take bi

NonlinearModelFit: The step size in the search has become less than the tolerance prescribed by the PrecisionGoal option, but the gradient is larger than the tolerance specified by the AccuracyGoal option. There is a possibility that the method has stalled at a point that is not a local minimum.

General: Further output of NonlinearModelFit::sszero will be suppressed during this calculation.

take bi

take bi



Mono: chi2 = 6.95516   d = 6.95516   a = 1.2251   t = 0.000967035

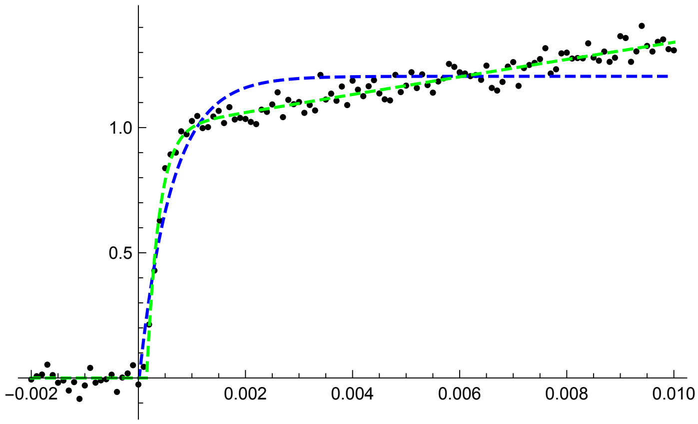
Bi: ratio chi2Mono/chi2Bi = 1.10343   d = 0.000184686   a = 18.6605   a1 = 0.958829   t1 =0.000310163   t2 = 0.417652

-- D

take bi

take bi

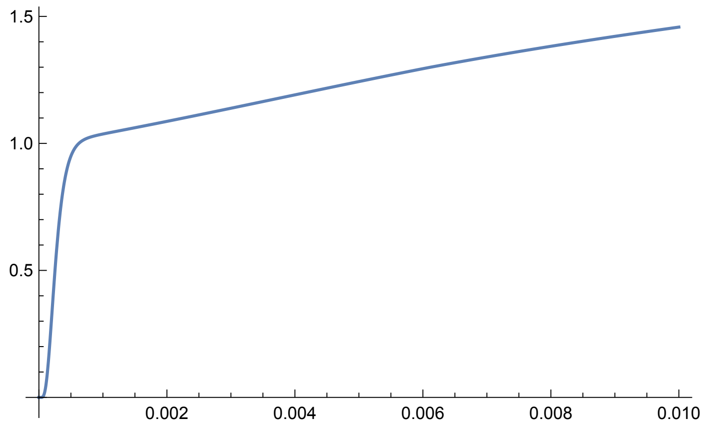
take bi

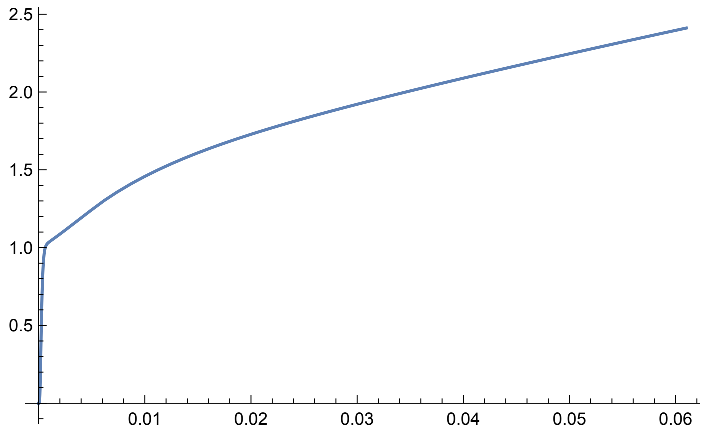


Mono: chi2 = 0.760542 d = 0.760542 a = 1.20466 t = 0.000605332

Bi: ratio chi2Mono/chi2Bi = 5.19154 d = 0.00015316 a = 5.94565 a1 = 0.993437 t1 =0.000228778 t2 = 0.135482

----------------- Ca = 29.9583 uM --



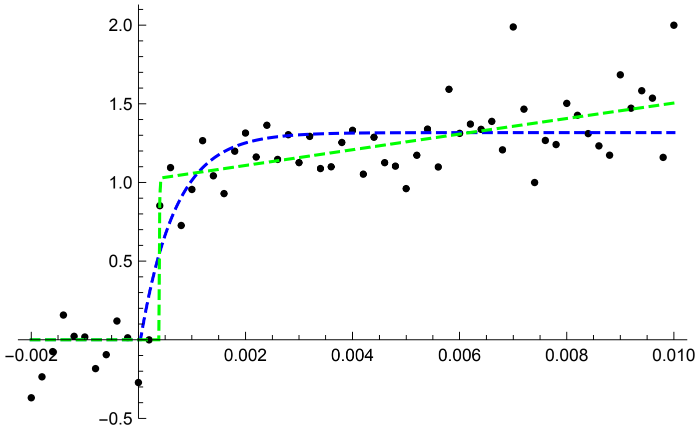


-- C5

take mono

take bi

take bi



Mono: chi2 = 2.8235 d = 2.8235 a = 1.3169 t = 0.000653994

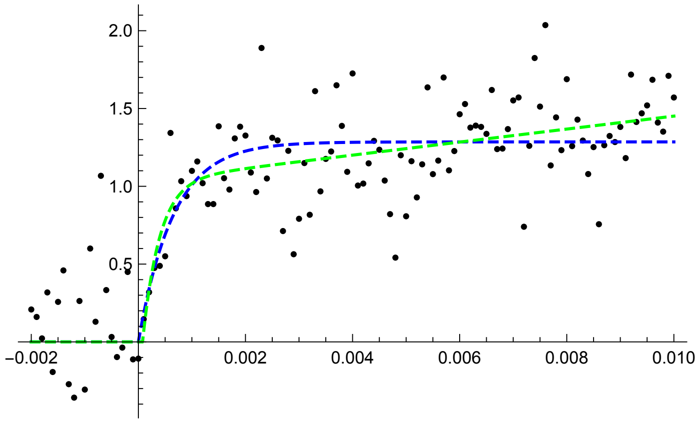
Bi: ratio chi2Mono/chi2Bi = 1.29174 d = 0.000385129 a = 17.9143 a1 = 1.02677 t1 $=8.41823\times10^{-6}$ t2 = 0.33477

-- C10

take mono

take mono

take mono



Mono: chi2 = 10.5048 d = 10.5048 a = 1.28546 t = 0.000650284

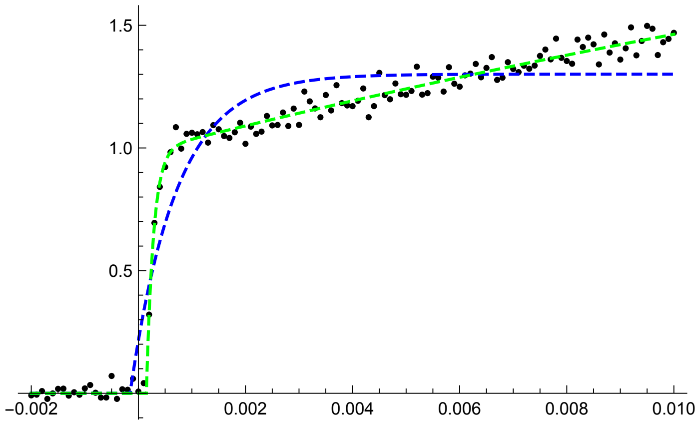
Bi: ratio chi2Mono/chi2Bi = 1.09062 d = 0.0000781728 a = 24.5676 a1 = 1.03494 t1 =0.000316712 t2 = 0.556046

-- D

take bi

take bi

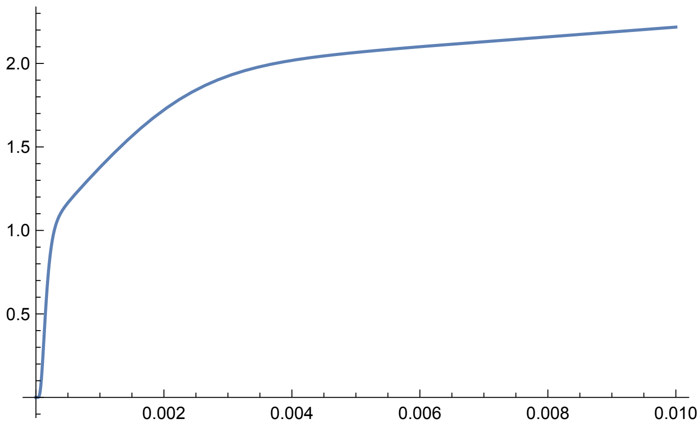
take bi

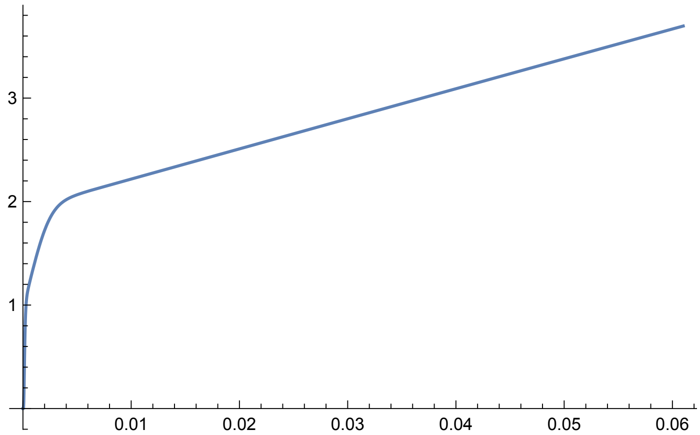


Mono: chi2 = 1.3488   d = 1.3488   a = 1.30082   t = 0.000860891

Bi: ratio chi2Mono/chi2Bi = 8.52178   d = 0.000149284   a = 2.81545   a1 = 0.989814   t1 =0.000130802   t2 = 0.032819

------------------   Ca = 74.0312 uM   ---



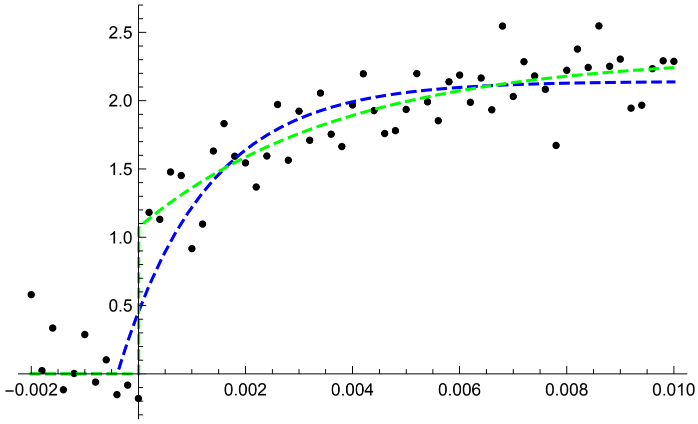


-- C5

take mono

take bi

take bi



Mono: chi2 = 3.78584 d = 3.78584 a = 2.14117 t = 0.00164855

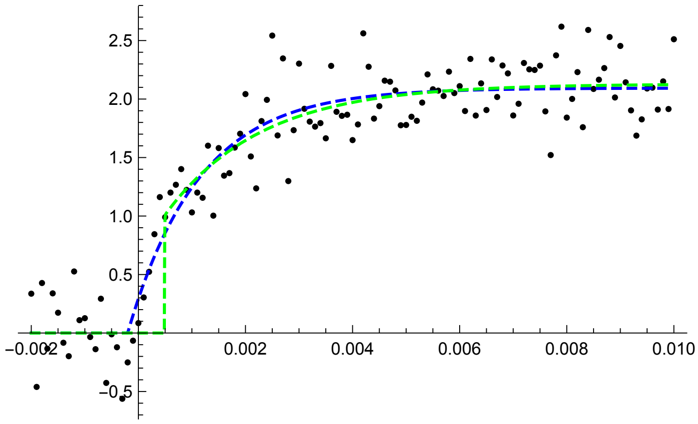
Bi: ratio chi2Mono/chi2Bi = 1.49062 d = 6.60019×10^{-6} a = 2.33322 a1 = 1.07308 t1 =5.41699×10^{-7} t2 = 0.00380677

--- C10

take mono

take bi

take mono



Mono: chi2 = 8.25616 d = 8.25616 a = 2.09344 t = 0.00133118

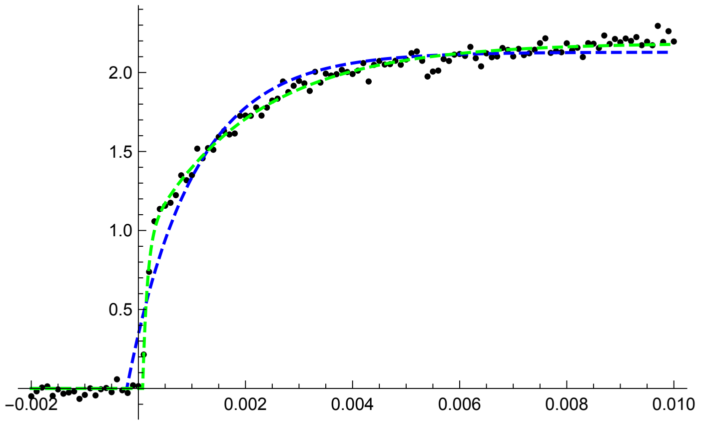
Bi: ratio chi2Mono/chi2Bi = 0.805212 d = 0.000485957 a = 2.12833 a1 = 1.00175 t1 =3.60911×10^{-6} t2 = 0.00178307

--- D

take bi

take bi

take bi




```
Mono: chi2 = 0.89184     d = 0.89184     a = 2.12853     t = 0.00122951
Bi: ratio chi2Mono/chi2Bi = 4.58807     d = 0.000079853     a =
 2.18721     a1 = 0.946759     t1 =0.0000870716     t2 = 0.00202291
```

Plots

In[•]:=
```
caFact = 1*^6;
```

In[•]:=
```
colorA = Green;
colorB = Red;
colorC = Blue;
```

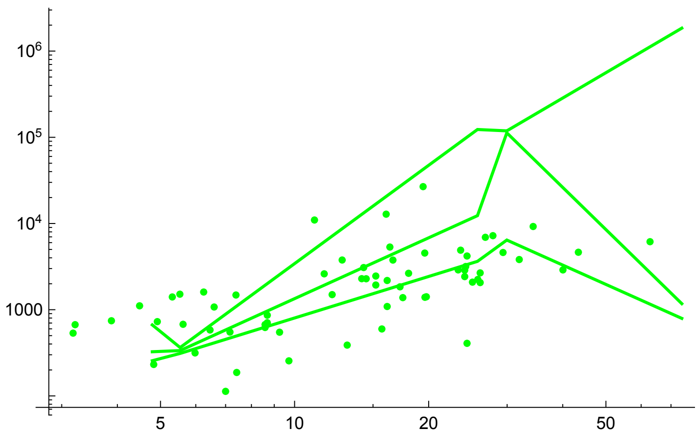
release rate 1/tau1 (merge of mono 1/tau and bi 1/tau1)

In[•]:=
```
simParam = 15;
```

C5

In[∘]:=
```
gr1a = ListLogLogPlot[
   Transpose[{dataT1C5Ca, dataT1C5RelRate}], PlotStyle → {colorA}];
gr2a = ListLogLogPlot[Transpose[{caFact simCaList,
      simParamMedianC5[[simParam, All]]}],
   PlotStyle → {colorA}, Joined → True, PlotRange → All];
gr3a = ListLogLogPlot[Transpose[{caFact simCaList,
      simParamQuantile1C5[[simParam, All]]}],
   PlotStyle → {colorA}, Joined → True, PlotRange → All];
gr4a = ListLogLogPlot[Transpose[{caFact simCaList,
      simParamQuantile2C5[[simParam, All]]}],
   PlotStyle → {colorA}, Joined → True, PlotRange → All];
Show[gr1a, gr2a, gr3a, gr4a, PlotRange → All ]
If[exportYes == 1,
  Export["plot InvTau1 C5 data.txt",
   Transpose[{dataT1C5Ca, dataT1C5RelRate}], "Table"];
  toExport = Transpose[{caFact simCaList, simParamQuantile1C5[[simParam, All]],
     simParamMedianC5[[simParam, All]], simParamQuantile2C5[[simParam, All]]}];
  Export["plot InvTau1 C5 fit - quantiles and median.txt", toExport, "Table"];
 ];
```

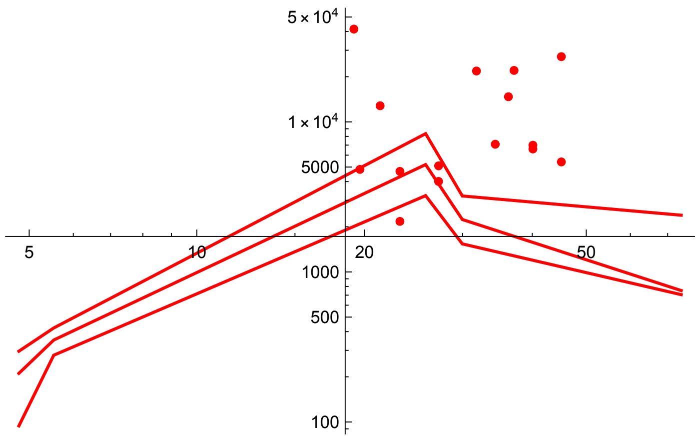
Out[∘]=



C10

In[∘]:=
```
gr1b = ListLogLogPlot[
   Transpose[{dataT1C10Ca, dataT1C10RelRate}], PlotStyle → {colorB}];
gr2b = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamMedianC10[[simParam, All]]}],
   PlotStyle → { colorB}, Joined → True, PlotRange → All];
gr3b = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamQuantile1C10[[simParam, All]]}],
   PlotStyle → { colorB}, Joined → True, PlotRange → All];
gr4b = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamQuantile2C10[[simParam, All]]}],
   PlotStyle → { colorB}, Joined → True, PlotRange → All];
Show[gr1b, gr2b, gr3b, gr4b, PlotRange → All ]
If[exportYes == 1,
  Export["plot InvTau1 C10 data.txt",
   Transpose[{dataT1C10Ca, dataT1C10RelRate}], "Table"];
  toExport = Transpose[{caFact simCaList, simParamQuantile1C10[[simParam, All]],
     simParamMedianC10[[simParam, All]],
     simParamQuantile2C10[[simParam, All]]}];
  Export["plot InvTau1 C10 fit - quantiles and median.txt", toExport, "Table"];
 ];
```

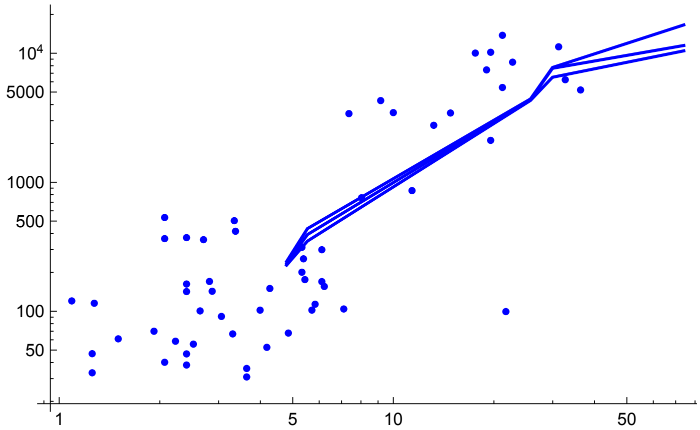
Out[∘]=



D

In[∘]:=
```
gr1c =
  ListLogLogPlot[Transpose[{dataT1DCa, dataT1DRelRate}], PlotStyle → {colorC}];
gr2c = ListLogLogPlot[Transpose[
     {caFact simCaList, simParamMedianD[[simParam, All]]}],
   PlotStyle → {colorC}, Joined → True, PlotRange → All];
gr3c = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamQuantile1D[[simParam, All]]}],
   PlotStyle → {colorC}, Joined → True, PlotRange → All];
gr4c = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamQuantile2D[[simParam, All]]}],
   PlotStyle → {colorC}, Joined → True, PlotRange → All];
Show[gr1c, gr2c, gr3c, gr4c, PlotRange → All]
If[exportYes == 1,
  Export["plot InvTau1 D data.txt",
   Transpose[{dataT1DCa, dataT1DRelRate}], "Table"];
  toExport = Transpose[{caFact simCaList, simParamQuantile1D[[simParam, All]],
     simParamMedianD[[simParam, All]], simParamQuantile2D[[simParam, All]]}];
  Export["plot InvTau1 D fit - quantiles and median.txt", toExport, "Table"];
 ];
```

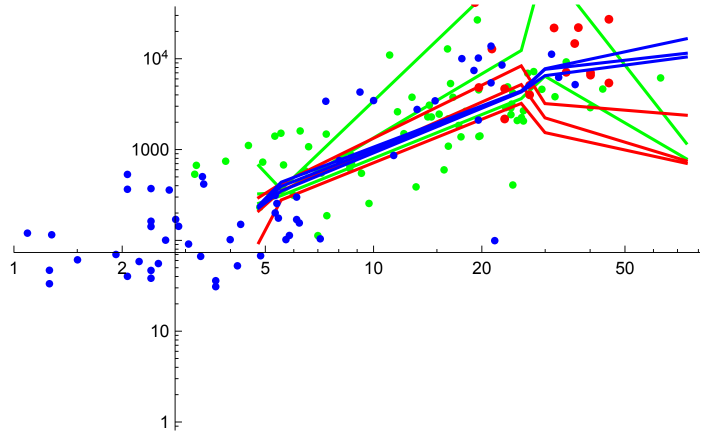
Out[∘]=



C5 and C10 and D

In[•]:=
```
Show[gr1a, gr2a, gr3a, gr4a, gr1b, gr2b, gr3b,
 gr4b, gr1c, gr2c, gr3c, gr4c, PlotRange → {All, {0, 10}}]
```

Out[•]=



In[•]:=
```
Show[gr1a, gr1b, gr1c, PlotRange → {All, {2, 10}}];
```

delay (mono and bi merged)

In[•]:=
```
simParam = 12;
```

In[•]:=
```
Transpose[{caFact simCaList, simParamMedianC5[[simParam, All]]}] // TableForm
```

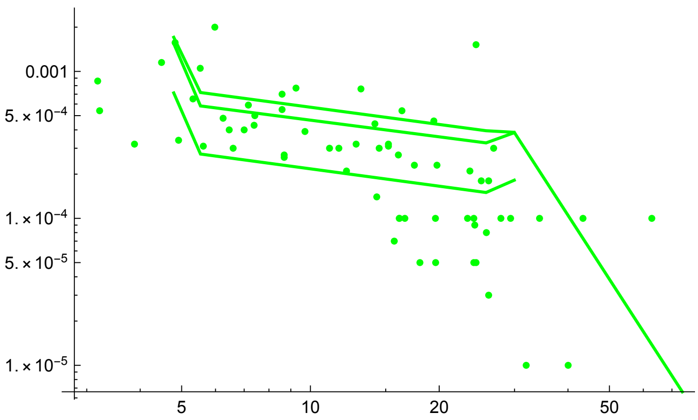
Out[•]//TableForm=

0.703073	
4.79194	0.0015475
5.53376	0.000582079
25.7412	0.000326227
29.9583	0.000382942
74.0312	-0.000187971

C5

In[∘]:=
```
gr1a =
  ListLogLogPlot[Transpose[{dataT1C5Ca, dataT1C5Delay}], PlotStyle → {colorA}];
gr2a = ListLogLogPlot[Transpose[
     {caFact simCaList, simParamMedianC5[[simParam, All]]}],
    PlotStyle → {colorA}, Joined → True, PlotRange → All];
gr3a = ListLogLogPlot[Transpose[{caFact simCaList,
      simParamQuantile1C5[[simParam, All]]}],
    PlotStyle → {colorA}, Joined → True, PlotRange → All];
gr4a = ListLogLogPlot[Transpose[{caFact simCaList,
      simParamQuantile2C5[[simParam, All]]}],
    PlotStyle → {colorA}, Joined → True, PlotRange → All];
Show[gr1a, gr2a, gr3a, gr4a, PlotRange → All ]
If[exportYes == 1,
  Export["plot delay C5 data.txt",
   Transpose[{dataT1C5Ca, dataT1C5Delay}], "Table"];
  toExport = Transpose[{caFact simCaList, simParamQuantile1C5[[simParam, All]],
      simParamMedianC5[[simParam, All]], simParamQuantile2C5[[simParam, All]]}];
  Export["plot delay C5 fit - quantiles and median.txt", toExport, "Table"];
 ];
```

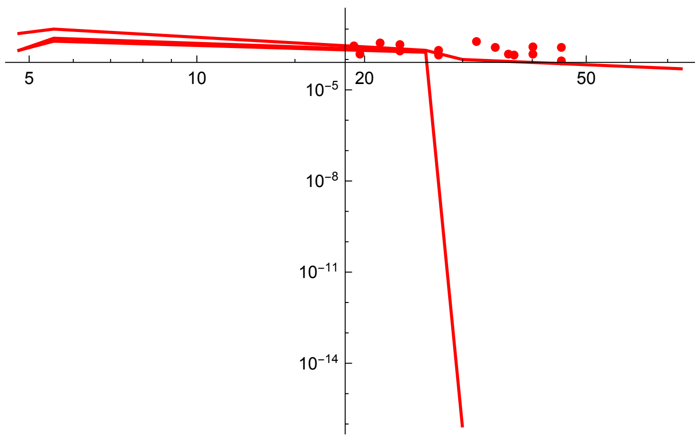
Out[∘]=



C10

In[∘]:=
```
gr1b = ListLogLogPlot[
   Transpose[{dataT1C10Ca, dataT1C10Delay}], PlotStyle → {colorB}];
gr2b = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamMedianC10[[simParam, All]]}],
   PlotStyle → { colorB}, Joined → True, PlotRange → All];
gr3b = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamQuantile1C10[[simParam, All]]}],
   PlotStyle → { colorB}, Joined → True, PlotRange → All];
gr4b = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamQuantile2C10[[simParam, All]]}],
   PlotStyle → { colorB}, Joined → True, PlotRange → All];
Show[gr1b, gr2b, gr3b, gr4b, PlotRange → All ]
If[exportYes == 1,
  Export["plot delay C10 data.txt",
   Transpose[{dataT1C10Ca, dataT1C10Delay}], "Table"];
  toExport = Transpose[{caFact simCaList, simParamQuantile1C10[[simParam, All]],
     simParamMedianC10[[simParam, All]],
     simParamQuantile2C10[[simParam, All]]}];
  Export["plot delay C10 fit - quantiles and median.txt", toExport, "Table"];
 ];
```

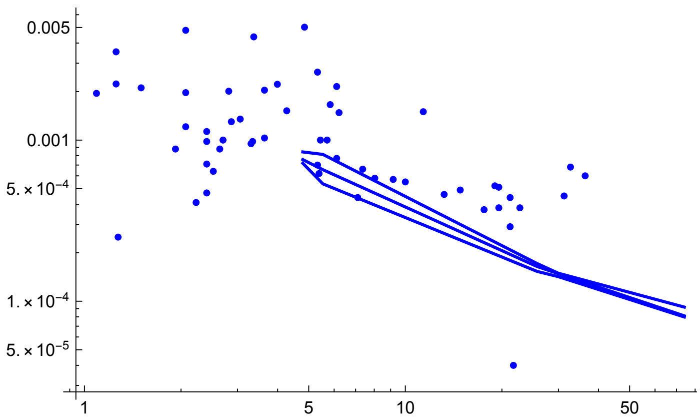
Out[∘]=



D

In[•]:=
```
gr1c =
  ListLogLogPlot[Transpose[{dataT1DCa, dataT1DDelay}], PlotStyle → {colorC}];
gr2c = ListLogLogPlot[Transpose[
     {caFact simCaList, simParamMedianD[[simParam, All]]}],
   PlotStyle → { colorC}, Joined → True, PlotRange → All];
gr3c = ListLogLogPlot[Transpose[{caFact simCaList,
      simParamQuantile1D[[simParam, All]]}],
   PlotStyle → { colorC}, Joined → True, PlotRange → All];
gr4c = ListLogLogPlot[Transpose[{caFact simCaList,
      simParamQuantile2D[[simParam, All]]}],
   PlotStyle → { colorC}, Joined → True, PlotRange → All];
Show[gr1c, gr2c, gr3c, gr4c, PlotRange → All ]
If[exportYes == 1,
  Export["plot delay D data.txt",
   Transpose[{dataT1DCa, dataT1DDelay}], "Table"];
  toExport = Transpose[{caFact simCaList, simParamQuantile1D[[simParam, All]],
      simParamMedianD[[simParam, All]], simParamQuantile2D[[simParam, All]]}];
  Export["plot delay D fit - quantiles and median.txt", toExport, "Table"];
 ];
```

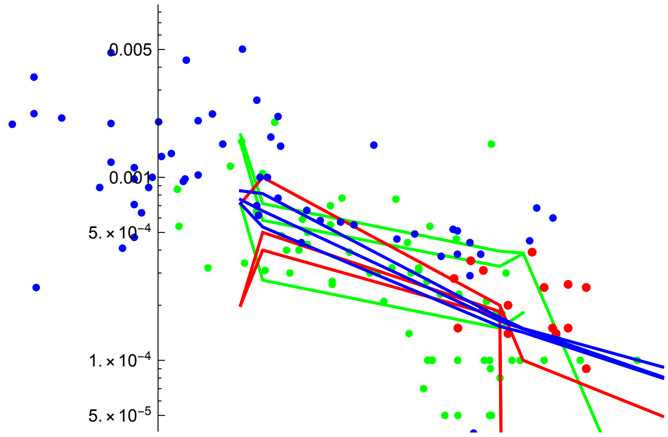
Out[•]=



C5 and C10 and D

In[∘]:=
```
Show[gr1a, gr2a, gr3a, gr4a, gr1b, gr2b, gr3b,
  gr4b, gr1c, gr2c, gr3c, gr4c, PlotRange → {-10, -5}]
```

Out[∘]=



In[∘]:=
```
Show[gr1a, gr1b, gr1c, PlotRange → All];
```

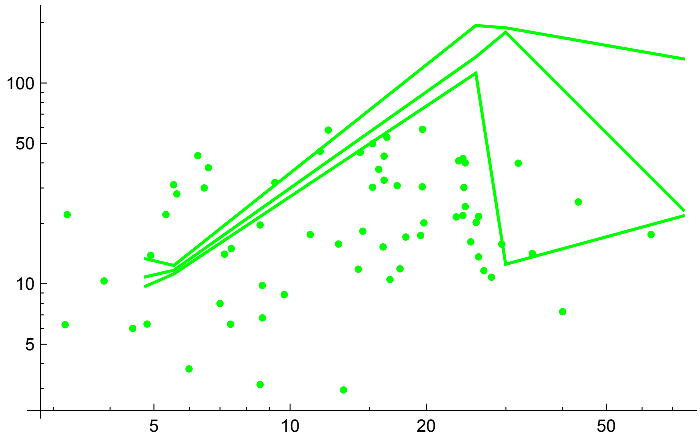
amp (merge of mono amp and bi amp)

In[∘]:=
```
simParam = 13;
```

C5

In[•]:=
```
gr1a = ListLogLogPlot[
   Transpose[{dataT1C5Ca, dataT1C5Amplitude}], PlotStyle → {colorA}];
gr2a = ListLogLogPlot[Transpose[{caFact simCaList,
     rrp simParamMedianC5[[simParam, All]]}],
   PlotStyle → {colorA}, Joined → True, PlotRange → All];
gr3a = ListLogLogPlot[Transpose[{caFact simCaList,
     rrp simParamQuantile1C5[[simParam, All]]}],
   PlotStyle → {colorA}, Joined → True, PlotRange → All];
gr4a = ListLogLogPlot[Transpose[{caFact simCaList,
     rrp simParamQuantile2C5[[simParam, All]]}],
   PlotStyle → {colorA}, Joined → True, PlotRange → All];
Show[gr1a, gr2a, gr3a, gr4a, PlotRange → All ]
```

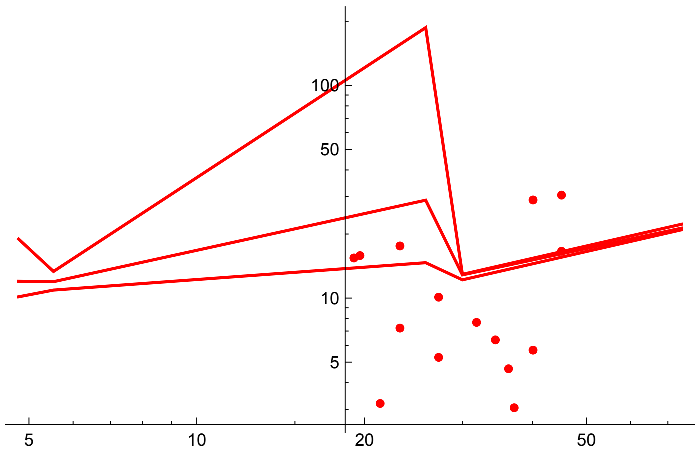
Out[•]=



C10

In[•]:=
```
gr1b = ListLogLogPlot[
   Transpose[{dataT1C10Ca, dataT1C10Amplitude}], PlotStyle → {colorB}];
gr2b = ListLogLogPlot[Transpose[{caFact simCaList,
     rrp simParamMedianC10[[simParam, All]]}],
   PlotStyle → { colorB}, Joined → True, PlotRange → All];
gr3b = ListLogLogPlot[Transpose[{caFact simCaList,
     rrp simParamQuantile1C10[[simParam, All]]}],
   PlotStyle → { colorB}, Joined → True, PlotRange → All];
gr4b = ListLogLogPlot[Transpose[{caFact simCaList,
     rrp simParamQuantile2C10[[simParam, All]]}],
   PlotStyle → { colorB}, Joined → True, PlotRange → All];
Show[gr1b, gr2b, gr3b, gr4b, PlotRange → All ]
```

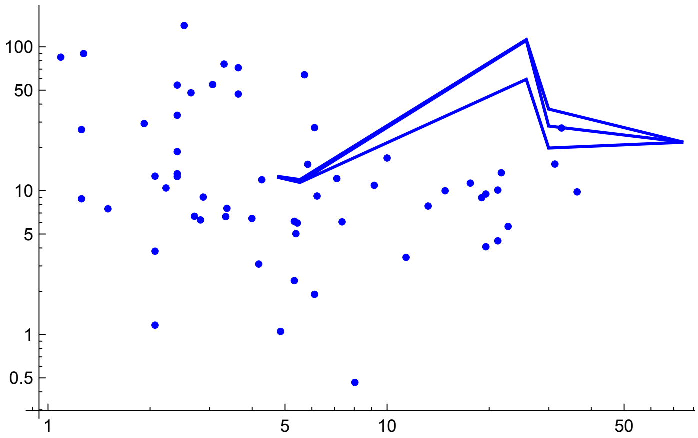
Out[•]=



D

```
In[]:= gr1c = ListLogLogPlot[
         Transpose[{dataT1DCa, dataT1DAmplitude}], PlotStyle → {colorC}];
       gr2c = ListLogLogPlot[Transpose[{caFact simCaList,
           rrp simParamMedianD[[simParam, All]]}],
         PlotStyle → { colorC}, Joined → True, PlotRange → All];
       gr3c = ListLogLogPlot[Transpose[{caFact simCaList,
           rrp simParamQuantile1D[[simParam, All]]}],
         PlotStyle → { colorC}, Joined → True, PlotRange → All];
       gr4c = ListLogLogPlot[Transpose[{caFact simCaList,
           rrp simParamQuantile2D[[simParam, All]]}],
         PlotStyle → { colorC}, Joined → True, PlotRange → All];
       Show[gr1c, gr2c, gr3c, gr4c, PlotRange → All ]
```

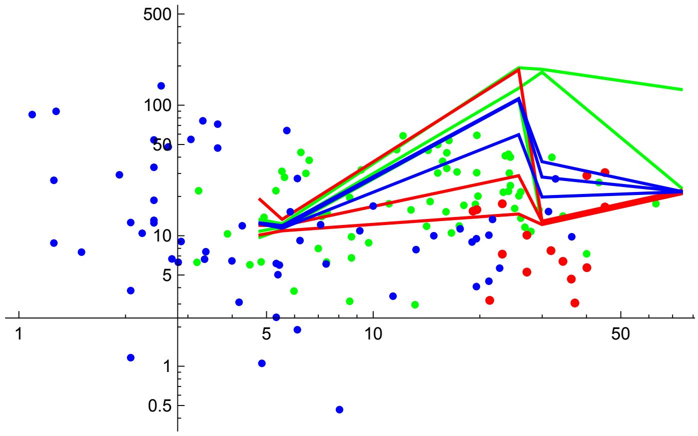
Out[]=



C5 and C10 and D

```
In[]:= Show[gr1a, gr2a, gr3a, gr4a, gr1b, gr2b, gr3b,
        gr4b, gr1c, gr2c, gr3c, gr4c, PlotRange → {-1, 6} ]
```

Out[]=



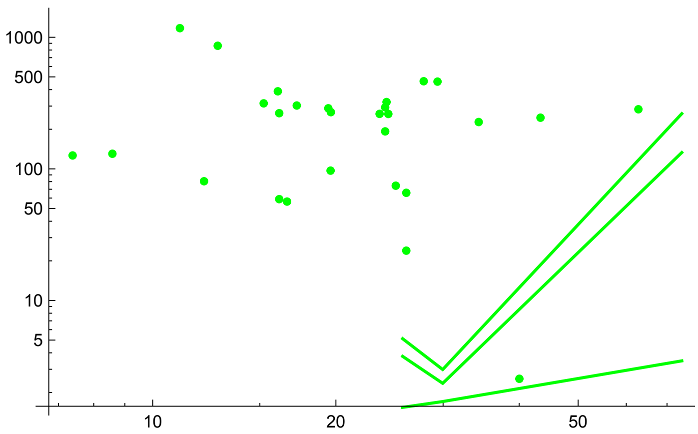
release rate 1/tau2 of bi fits (if bi is justified)

```
In[]:= simParam = 16;
```

C5

In[•]:=
```
gr1a = ListLogLogPlot[
   Transpose[{dataT2C5Ca, dataT2C5RelRate}], PlotStyle → {colorA}];
gr2a = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamMedianC5[[simParam, All]]}],
   PlotStyle → {colorA}, Joined → True, PlotRange → All];
gr3a = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamQuantile1C5[[simParam, All]]}],
   PlotStyle → {colorA}, Joined → True, PlotRange → All];
gr4a = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamQuantile2C5[[simParam, All]]}],
   PlotStyle → {colorA}, Joined → True, PlotRange → All];
Show[gr1a, gr2a, gr3a, gr4a, PlotRange → All ]
If[exportYes == 1,
  Export["plot InvTau2 C5 data.txt",
   Transpose[{dataT2C5Ca, dataT2C5RelRate}], "Table"];
  toExport = Transpose[{caFact simCaList, simParamQuantile1C5[[simParam, All]],
     simParamMedianC5[[simParam, All]], simParamQuantile2C5[[simParam, All]]}];
  Export["plot InvTau2 C5 fit - quantiles and median.txt", toExport, "Table"];
 ];
```

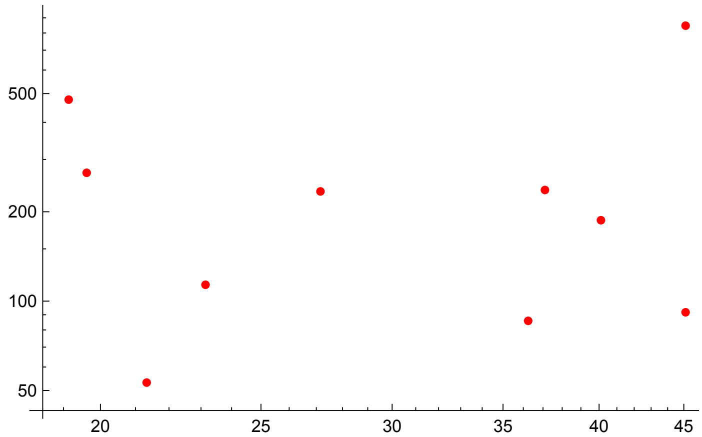
Out[•]=



C10

In[∘]:=
```
gr1b = ListLogLogPlot[
   Transpose[{dataT2C10Ca, dataT2C10RelRate}], PlotStyle → {colorB}];
gr2b = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamMedianC10[[simParam, All]]}],
   PlotStyle → { colorB}, Joined → True, PlotRange → All];
gr3b = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamQuantile1C10[[simParam, All]]}],
   PlotStyle → { colorB}, Joined → True, PlotRange → All];
gr4b = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamQuantile2C10[[simParam, All]]}],
   PlotStyle → { colorB}, Joined → True, PlotRange → All];
Show[gr1b, gr2b, gr3b, gr4b, PlotRange → All ]
If[exportYes == 1,
  Export["plot InvTau2 C10 data.txt",
   Transpose[{dataT2C10Ca, dataT2C10RelRate}], "Table"];
  toExport = Transpose[{caFact simCaList, simParamQuantile1C10[[simParam, All]],
     simParamMedianC10[[simParam, All]],
     simParamQuantile2C10[[simParam, All]]}];
  Export["plot InvTau2 C10 fit - quantiles and median.txt", toExport, "Table"];
 ];
```

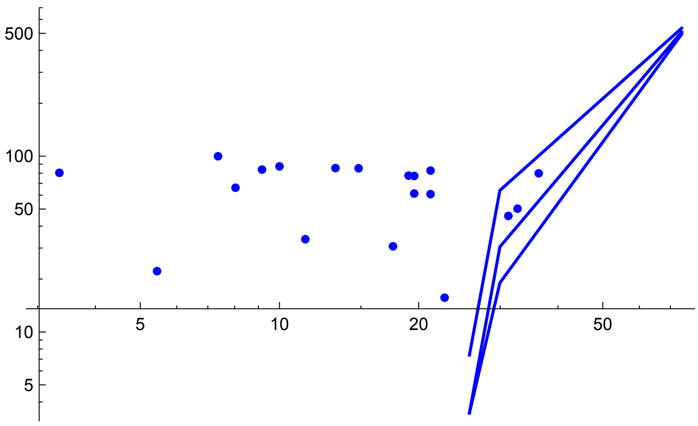
Out[∘]=



D

In[ ]:=
```
gr1c =
  ListLogLogPlot[Transpose[{dataT2DCa, dataT2DRelRate}], PlotStyle → {colorC}];
gr2c = ListLogLogPlot[Transpose[
    {caFact simCaList, simParamMedianD[[simParam, All]]}],
   PlotStyle → { colorC}, Joined → True, PlotRange → All];
gr3c = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamQuantile1D[[simParam, All]]}],
   PlotStyle → { colorC}, Joined → True, PlotRange → All];
gr4c = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamQuantile2D[[simParam, All]]}],
   PlotStyle → { colorC}, Joined → True, PlotRange → All];
Show[gr1c, gr2c, gr3c, gr4c, PlotRange → All ]
If[exportYes == 1,
  Export["plot InvTau2 D data.txt",
   Transpose[{dataT2DCa, dataT2DRelRate}], "Table"];
  toExport = Transpose[{caFact simCaList, simParamQuantile1D[[simParam, All]],
     simParamMedianD[[simParam, All]], simParamQuantile2D[[simParam, All]]}];
  Export["plot InvTau2 D fit - quantiles and median.txt", toExport, "Table"];
 ];
```

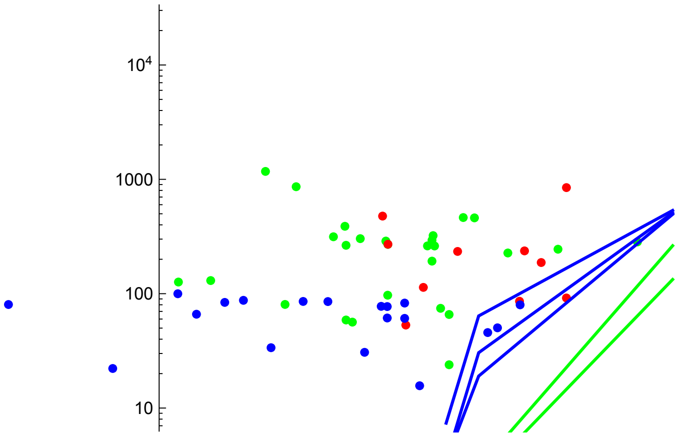
Out[ ]=



C5 and C10 and D

In[•]:=
```
Show[gr1a, gr2a, gr3a, gr4a, gr1b, gr2b, gr3b,
 gr4b, gr1c, gr2c, gr3c, gr4c, PlotRange → {All, {2, 10}}]
```

Out[•]=



In[•]:=
```
Show[gr1a, gr1b, gr1c, PlotRange → All];
```

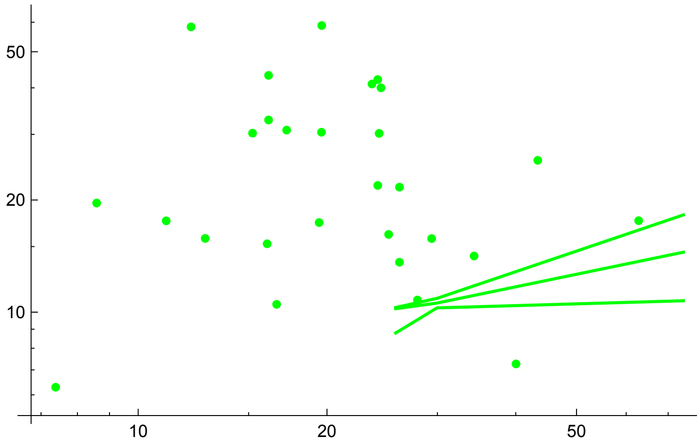
amp1 of bi fits (if bi is justified)

In[•]:=
```
simParam = 14;
```

C5

In[∘]:=
```
gr1a = ListLogLogPlot[
   Transpose[{dataT2C5Ca, dataT2C5Amplitude1}], PlotStyle → {colorA}];
gr2a = ListLogLogPlot[Transpose[{caFact simCaList,
      rrp simParamMedianC5[[simParam, All]]}],
   PlotStyle → {colorA}, Joined → True, PlotRange → All];
gr3a = ListLogLogPlot[Transpose[{caFact simCaList,
      rrp simParamQuantile1C5[[simParam, All]]}],
   PlotStyle → {colorA}, Joined → True, PlotRange → All];
gr4a = ListLogLogPlot[Transpose[{caFact simCaList,
      rrp simParamQuantile2C5[[simParam, All]]}],
   PlotStyle → { colorA}, Joined → True, PlotRange → All];
Show[gr1a, gr2a, gr3a, gr4a, PlotRange → All ]
```

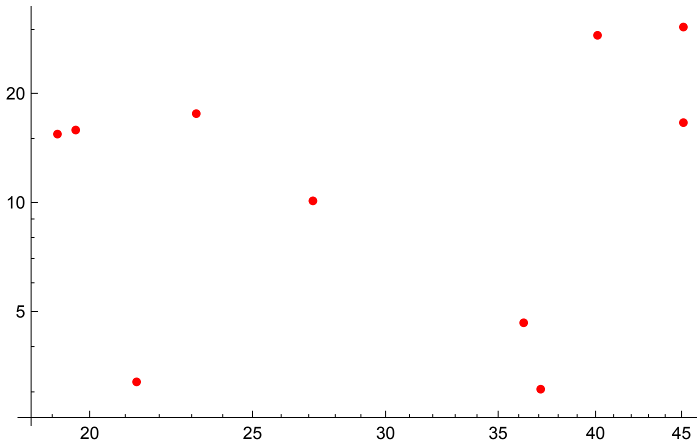
Out[∘]=



C10

In[•]:=
```
gr1b = ListLogLogPlot[
   Transpose[{dataT2C10Ca, dataT2C10Amplitude1}], PlotStyle → {colorB}];
gr2b = ListLogLogPlot[Transpose[{caFact simCaList,
     rrp simParamMedianC10[[simParam, All]]}],
   PlotStyle → { colorB}, Joined → True, PlotRange → All];
gr3b = ListLogLogPlot[Transpose[{caFact simCaList,
     rrp simParamQuantile1C10[[simParam, All]]}],
   PlotStyle → {colorB}, Joined → True, PlotRange → All];
gr4b = ListLogLogPlot[Transpose[{caFact simCaList,
     rrp simParamQuantile2C10[[simParam, All]]}],
   PlotStyle → {colorB}, Joined → True, PlotRange → All];
Show[gr1b, gr2b, gr3b, gr4b, PlotRange → All ]
```

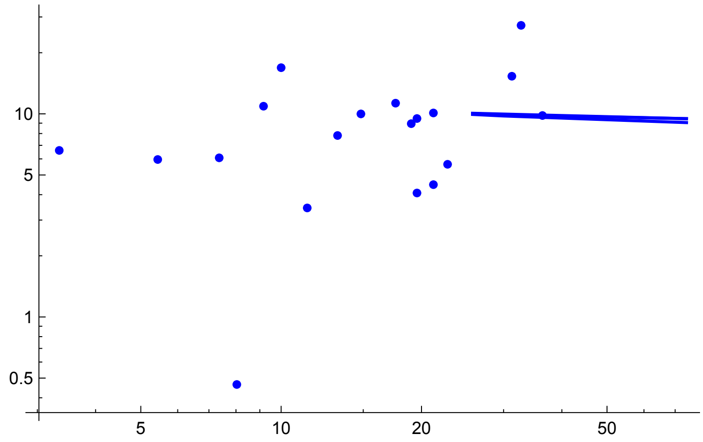
Out[•]=



D

In[•]:=
```
gr1c = ListLogLogPlot[
   Transpose[{dataT2DCa, dataT2DAmplitude1}], PlotStyle → {colorC}];
gr2c = ListLogLogPlot[Transpose[{caFact simCaList,
     rrp simParamMedianD[[simParam, All]]}],
   PlotStyle → {colorC}, Joined → True, PlotRange → All];
gr3c = ListLogLogPlot[Transpose[{caFact simCaList,
     rrp simParamQuantile1D[[simParam, All]]}],
   PlotStyle → {colorC}, Joined → True, PlotRange → All];
gr4c = ListLogLogPlot[Transpose[{caFact simCaList,
     rrp simParamQuantile2D[[simParam, All]]}],
   PlotStyle → {colorC}, Joined → True, PlotRange → All];
Show[gr1c, gr2c, gr3c, gr4c, PlotRange → All ]
```

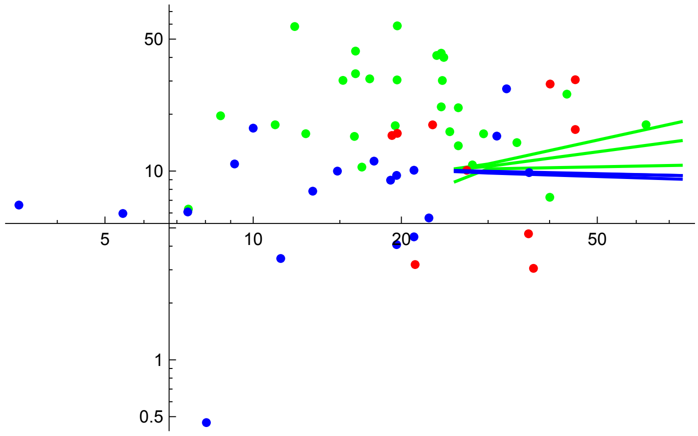
Out[•]=



C5 and C10 and D

In[•]:=
```
Show[gr1a, gr2a, gr3a, gr4a, gr1b, gr2b,
 gr3b, gr4b, gr1c, gr2c, gr3c, gr4c, PlotRange → All]
Show[gr1a, gr1b, gr1c, PlotRange → All];
```

Out[•]=



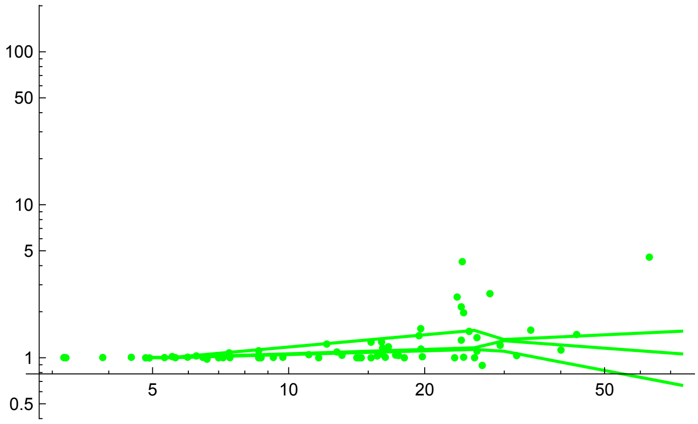
chi2 mono/bi ratio

In[•]:=
```
simParam = 6;
```

C5

In[•]:=
```
gr1a = ListLogLogPlot[
    Transpose[{dataT1C5Ca, dataT1C5ChiRatio}], PlotStyle → {colorA}];
gr2a = ListLogLogPlot[Transpose[{caFact simCaList,
      simParamMedianC5[[simParam, All]]}],
    PlotStyle → {colorA}, Joined → True, PlotRange → All];
gr3a = ListLogLogPlot[Transpose[{caFact simCaList,
      simParamQuantile1C5[[simParam, All]]}],
    PlotStyle → {colorA}, Joined → True, PlotRange → All];
gr4a = ListLogLogPlot[Transpose[{caFact simCaList,
      simParamQuantile2C5[[simParam, All]]}],
    PlotStyle → {colorA}, Joined → True, PlotRange → All];
Show[gr1a, gr2a, gr3a, gr4a, PlotRange → {All, {-0.8, 5}}]
If[exportYes == 1,
   Export["plot chi2Ratio C5 data.txt",
    Transpose[{dataT1C5Ca, dataT1C5ChiRatio}], "Table"];
   toExport = Transpose[{caFact simCaList, simParamQuantile1C5[[simParam, All]],
      simParamMedianC5[[simParam, All]], simParamQuantile2C5[[simParam, All]]}];
   Export["plot chi2Ratio C5 fit - quantiles and median.txt",
    toExport, "Table"];
 ];
```

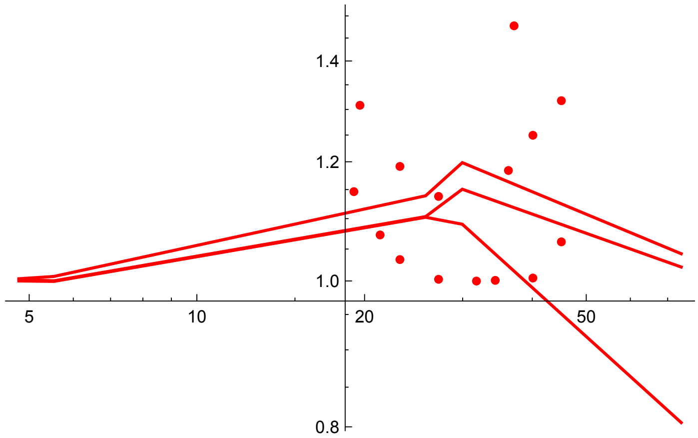
Out[•]=



C10

In[∘]:=
```
gr1b = ListLogLogPlot[
   Transpose[{dataT1C10Ca, dataT1C10ChiRatio}], PlotStyle → {colorB}];
gr2b = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamMedianC10[[simParam, All]]}],
   PlotStyle → {colorB}, Joined → True, PlotRange → All];
gr3b = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamQuantile1C10[[simParam, All]]}],
   PlotStyle → {colorB}, Joined → True, PlotRange → All];
gr4b = ListLogLogPlot[Transpose[{caFact simCaList,
     simParamQuantile2C10[[simParam, All]]}],
   PlotStyle → {colorB}, Joined → True, PlotRange → All];
Show[gr1b, gr2b, gr3b, gr4b, PlotRange → All ]
If[exportYes == 1,
  Export["plot chi2Ratio C10 data.txt",
   Transpose[{dataT1C10Ca, dataT1C10ChiRatio}], "Table"];
  toExport = Transpose[{caFact simCaList, simParamQuantile1C10[[simParam, All]],
     simParamMedianC10[[simParam, All]],
     simParamQuantile2C10[[simParam, All]]}];
  Export["plot chi2Ratio C10 fit - quantiles and median.txt",
   toExport, "Table"];
 ];
```

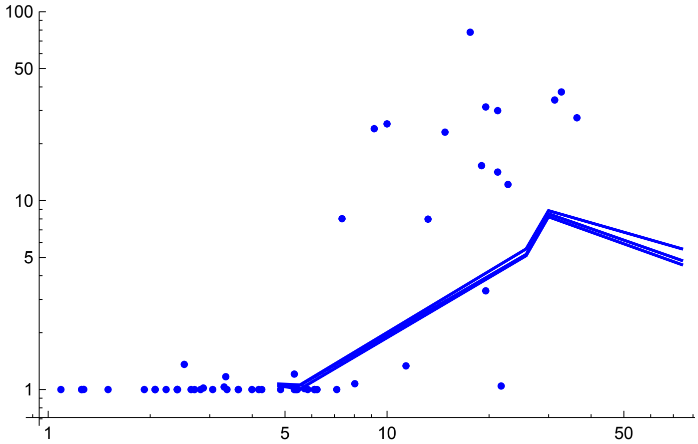
Out[∘]=



D

In[•]:=
```
gr1c =
  ListLogLogPlot[Transpose[{dataT1DCa, dataT1DChiRatio}], PlotStyle → {colorC}];
gr2c = ListLogLogPlot[Transpose[
     {caFact simCaList, simParamMedianD[[simParam, All]]}],
   PlotStyle → {colorC}, Joined → True, PlotRange → All];
gr3c = ListLogLogPlot[Transpose[{caFact simCaList,
      simParamQuantile1D[[simParam, All]]}],
   PlotStyle → {colorC}, Joined → True, PlotRange → All];
gr4c = ListLogLogPlot[Transpose[{caFact simCaList,
      simParamQuantile2D[[simParam, All]]}],
   PlotStyle → {colorC}, Joined → True, PlotRange → All];
Show[gr1c, gr2c, gr3c, gr4c, PlotRange → All ]
If[exportYes == 1,
  Export["plot chi2Ratio D data.txt",
   Transpose[{dataT1DCa, dataT1DChiRatio}], "Table"];
  toExport = Transpose[{caFact simCaList, simParamQuantile1D[[simParam, All]],
      simParamMedianD[[simParam, All]], simParamQuantile2D[[simParam, All]]}];
  Export["plot chi2Ratio D fit - quantiles and median.txt", toExport, "Table"];
 ];
```

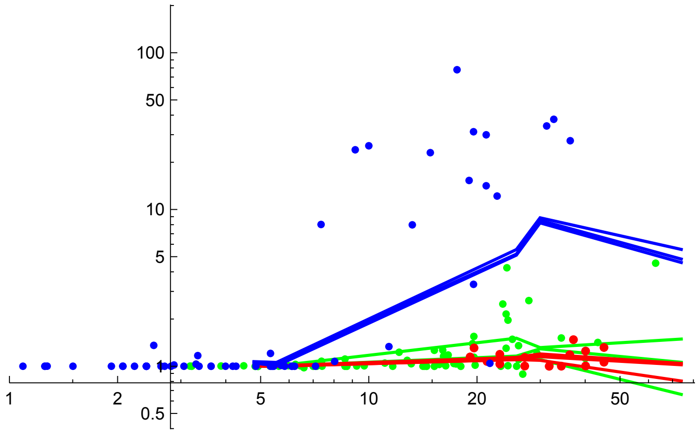
Out[•]=



C5 and C10 and D

In[∘]:=
```
Show[gr1a, gr2a, gr3a, gr4a, gr1b, gr2b, gr3b,
  gr4b, gr1c, gr2c, gr3c, gr4c, PlotRange → {All, {-0.8, 5}}]
Show[gr1a, gr1b, gr1c, PlotRange → All];
```

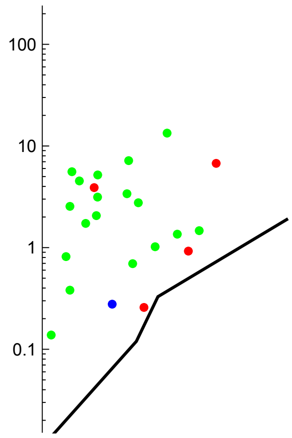
Out[∘]=



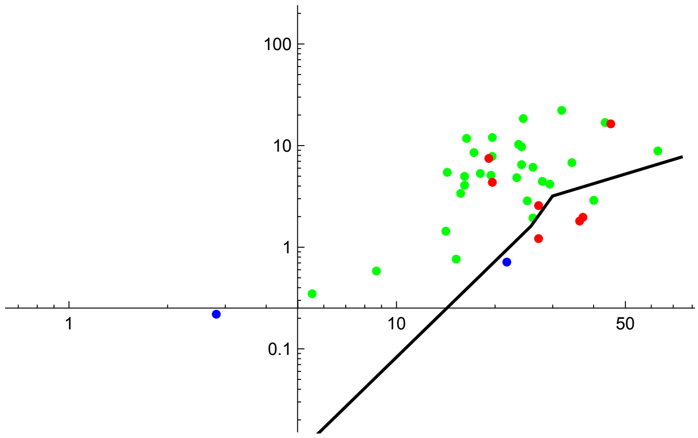
Nv

In[∘]:=
```
For[NvCount = 1, NvCount ≤ 7, NvCount += 1,
  Print[" time for Nv (ms) = ", 1000 * timeOfNv[[NvCount]]];
  gr1a = ListLogLogPlot[
    Transpose[{dataT1C5Ca, dataT1C5Nv[[NvCount]]}], PlotStyle → {colorA}];
  gr1b = ListLogLogPlot[Transpose[{dataT1C10Ca, dataT1C10Nv[[NvCount]]}],
    PlotStyle → {colorB}];
  gr1c = ListLogLogPlot[Transpose[{dataT1DCa, dataT1DNv[[NvCount]]}],
    PlotStyle → {colorC}];
  gr2 = ListLogLogPlot[Transpose[{caFact simCaList, rrp simParamNv[[NvCount,
        All]]}], PlotStyle → { Black}, Joined → True, PlotRange → All];
  Show[gr1a, gr1b, gr1c, gr2, PlotRange → {All, {-4, 5}} ] // Print;
 ];
```

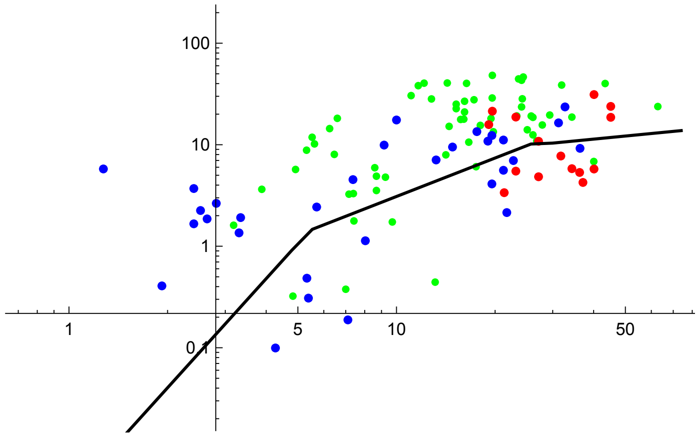
time for Nv (ms) = 0.1



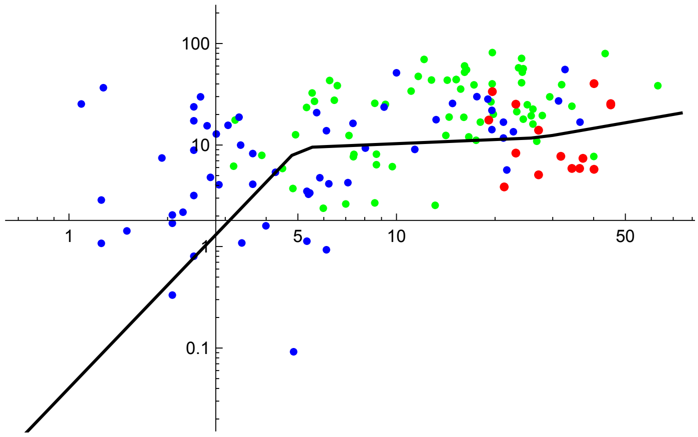
time for Nv (ms) = 0.2



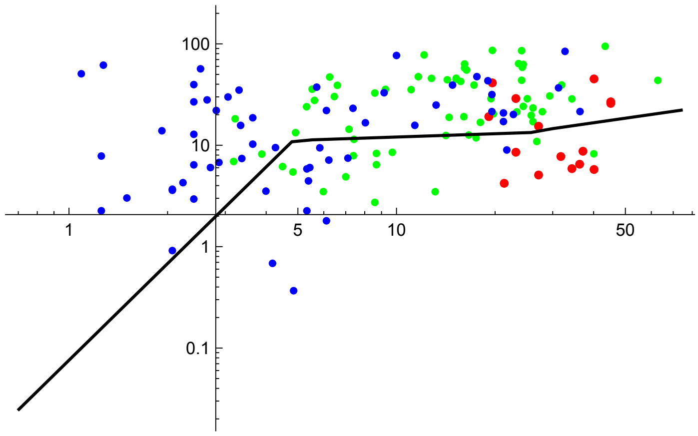

```
time for Nv (ms) = 1.
```



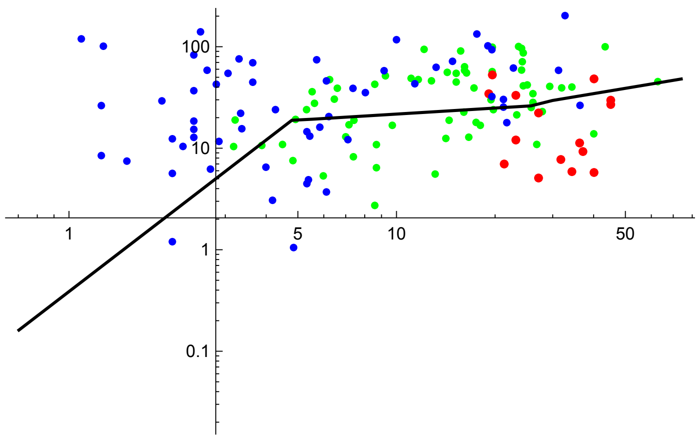

```
time for Nv (ms) = 5.
```



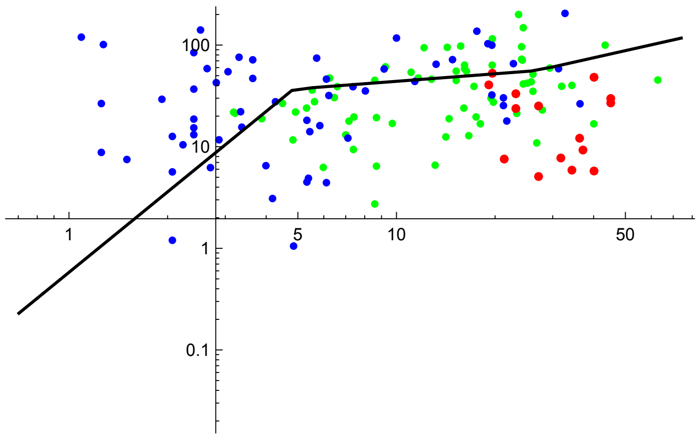

```
time for Nv (ms) = 10.
```



time for Nv (ms) = 100.

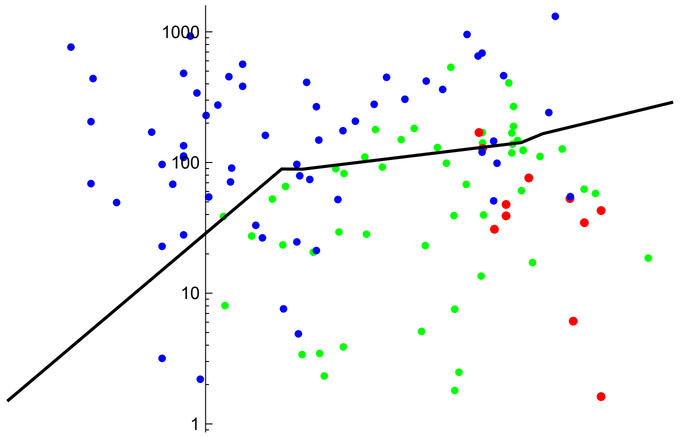


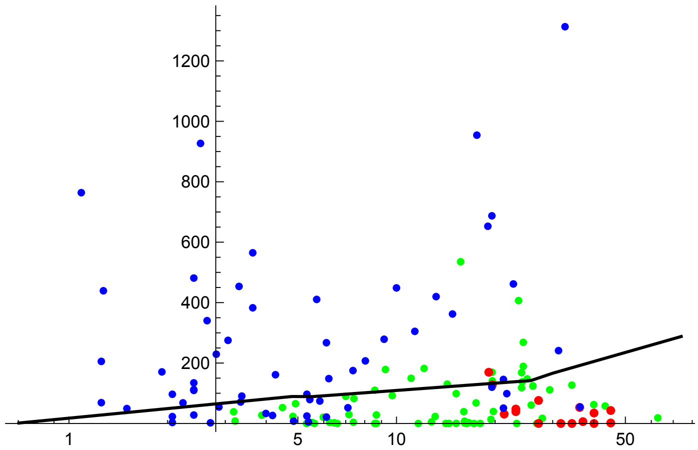
time for Nv (ms) = 400.



sustained release 10 to 100 ms

In[•]:=
```
ttt1 = Transpose[{dataT1C5Ca, (dataT1C5Nv[[6]] - dataT1C5Nv[[5]]) / 0.09}];
ttt2 = Transpose[{dataT1C10Ca, (dataT1C10Nv[[6]] - dataT1C10Nv[[5]]) / 0.09}];
ttt3 = Transpose[{dataT1DCa, (dataT1DNv[[6]] - dataT1DNv[[5]]) / 0.09}];
ttt4 = Transpose[
    {caFact simCaList, rrp (simParamNv[[6, All]] - simParamNv[[5, All]]) / 0.09}];

gr1a = ListLogLogPlot[ttt1, PlotStyle → {colorA}];
gr1b = ListLogLogPlot[ttt2, PlotStyle → {colorB}];
gr1c = ListLogLogPlot[ttt3, PlotStyle → {colorC}];
gr2 = ListLogLogPlot[ttt4, PlotStyle → { Black}, Joined → True, PlotRange → All];
Show[gr1a, gr1b, gr1c, gr2, PlotRange → {All, {0, 7}} ] // Print;

gr1a = ListLogLinearPlot[ttt1, PlotStyle → {colorA}];
gr1b = ListLogLinearPlot[ttt2, PlotStyle → {colorB}];
gr1c = ListLogLinearPlot[ttt3, PlotStyle → {colorC}];
gr2 = ListLogLinearPlot[ttt4,
   PlotStyle → { Black}, Joined → True, PlotRange → All];
Show[gr1a, gr1b, gr1c, gr2, PlotRange → {All, All} ] // Print;
```





In[•]:=
```
If[exportYes == 1,
  Export["plot sustained release Cm5 data.txt", ttt1, "Table"];
  Export["plot sustained release Cm10 data.txt", ttt2, "Table"];
  Export["plot sustained release D data.txt", ttt3, "Table"];
  Export["plot sustained release sim.txt", ttt4, "Table"]
 ];
```

Nv normalize to the value at 5 ms

In[•]:=
```
timeOfNv
nromPos = 5;
1000 * timeOfNv[[nromPos]]
```

Out[•]= {0.0001, 0.0002, 0.001, 0.005, 0.01, 0.1, 0.4}

Out[•]= 10.

In[•]:=
```
dataT1DNvNorm = Transpose[Transpose[dataT1DNv] / dataT1DNv[[nromPos]]];
dataT1C10NvNorm = Transpose[Transpose[dataT1C10Nv] / dataT1C10Nv[[nromPos]]];
dataT1C5NvNorm = Transpose[Transpose[dataT1C5Nv] / dataT1C5Nv[[nromPos]]];
```

In[•]:=
```
dataT1C5NvNorm // TableForm
```

Out[•]//TableForm=

0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.
0.20362	0.265687	0.42823	0.550679	0.804146	0.862111	0.657036
0.719618	0.914034	0.947765	0.995367	0.999983	0.961307	0.985586
1.	1.	1.	1.	1.	1.	1.
1.97523	1.00693	1.44309	1.00001	1.	1.38099	1.17479
1.98885	1.00693	1.64647	1.00001	1.	1.52805	1.34343

In[•]:=
```
simParamNv // TableForm
simParamNvNorm = Transpose[Transpose[simParamNv] / simParamNv[[nromPos]]];
simParamNvNorm // TableForm
```

Out[•]//TableForm=

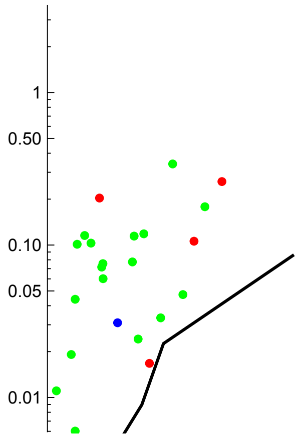
1.15733×10^{-8}	0.0000223798	0.0000466775	0.0119635	0.0329205	0.188836
4.0267×10^{-7}	0.000687618	0.00129019	0.160989	0.318746	0.769426
0.0000966667	0.0913074	0.14679	1.01594	1.03695	1.37729
0.00123275	0.791128	0.952943	1.17143	1.24341	2.06583
0.00248568	1.08299	1.13109	1.33522	1.45782	2.21762
0.0161613	1.8861	1.93094	2.61354	2.95342	4.80775
0.0229784	3.57675	3.80516	5.55074	6.08559	11.7107

Out[•]//TableForm=

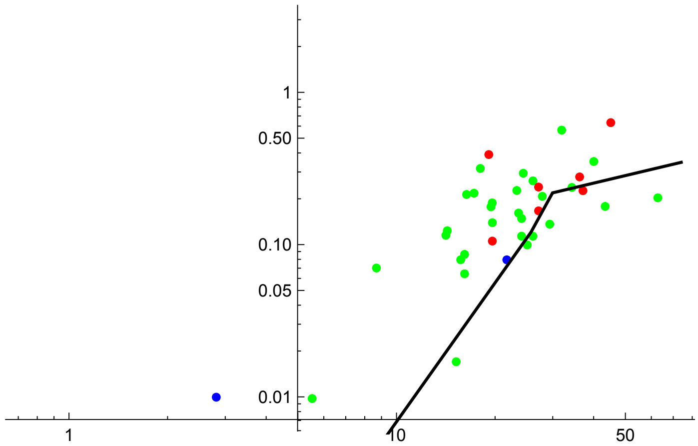
4.656×10^{-6}	0.0000206648	0.0000412679	0.00895994	0.022582	0.0851525
0.000161996	0.000634927	0.00114066	0.120571	0.218646	0.34696
0.0388895	0.0843107	0.129778	0.76088	0.711303	0.621066
0.495941	0.730505	0.842503	0.877331	0.852926	0.931553
1.	1.	1.	1.	1.	1.
6.50175	1.74157	1.70715	1.95738	2.02591	2.16797
9.24432	3.30267	3.36416	4.15717	4.17445	5.28075

```
In[•]:= For[NvCount = 1, NvCount ≤ 7, NvCount += 1,
  Print[" time for Nv (ms) = ", 1000 * timeOfNv[[NvCount]]];
  gr1a = ListLogLogPlot[
    Transpose[{dataT1C5Ca, dataT1C5NvNorm[[NvCount]]}], PlotStyle → {colorA}];
  gr1b = ListLogLogPlot[Transpose[{dataT1C10Ca, dataT1C10NvNorm[[NvCount]]}],
    PlotStyle → {colorB}];
  gr1c = ListLogLogPlot[Transpose[{dataT1DCa, dataT1DNvNorm[[NvCount]]}],
    PlotStyle → {colorC}];
  gr2 = ListLogLogPlot[Transpose[{caFact simCaList, simParamNvNorm[[NvCount,
       All]]}], PlotStyle → { Black}, Joined → True, PlotRange → All];
  Show[gr1a, gr1b, gr1c, gr2, PlotRange → {All, {-5, 1}} ] // Print;
 ];
```

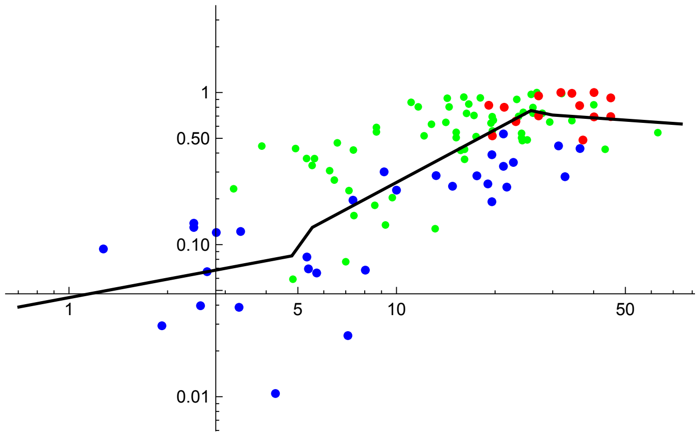
```
time for Nv (ms) = 0.1
```



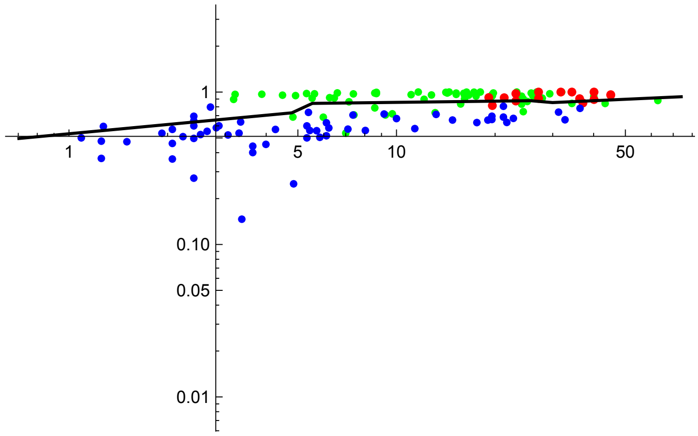

```
time for Nv (ms) = 0.2
```



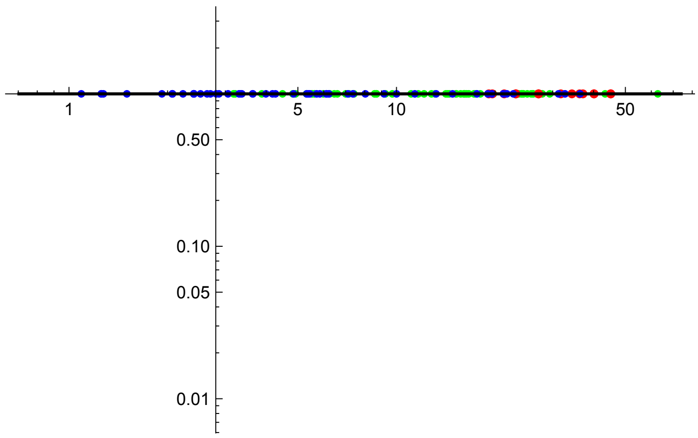

```
time for Nv (ms) = 1.
```



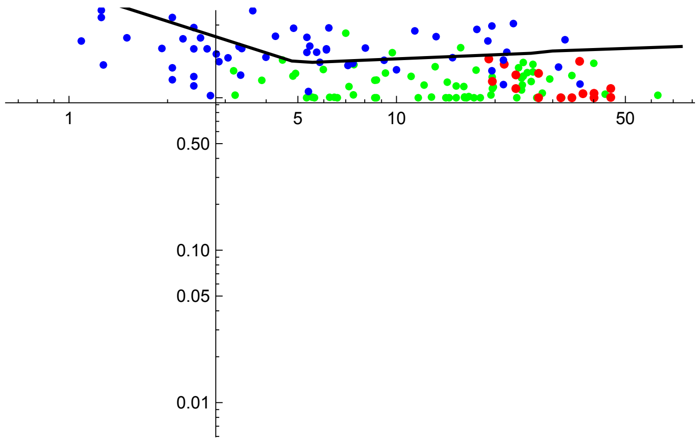
time for Nv (ms) = 5.



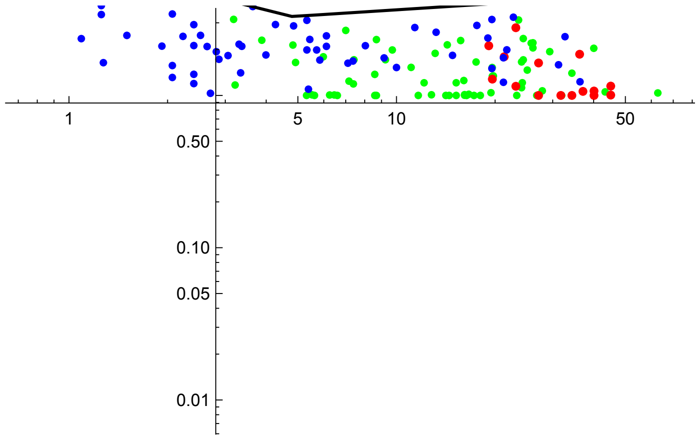
time for Nv (ms) = 10.



time for Nv (ms) = 100.




```
time for Nv (ms) = 400.
```



Export Nv

In[•]:=
```
If[exportYes == 1,
  Export["Nv export Ca,0.0001,0.0002,0.001,0.005,0.01,0.1,0.4.txt",
    Transpose[Prepend[simParamNv, caFact simCaList]], "Table"];
 ];
```

Print some values

C5

In[ ]:=
```
Transpose[simParamMedianC5] // TableForm
Transpose[simParamQuantile1C5] // TableForm
Transpose[simParamQuantile2C5] // TableForm
```

Out[ ]//TableForm=

0	2.5573	0.0015475	1.08303	325.139	1.00194	0.001514	0.
0	2.5008	0.000582079	1.16827	335.43	1.00275	0.000578415	0.
0	2.56658	0.000365502	1.22178	6534.34	1.16104	0.000326227	13
0	3.23888	0.00018182	1.25086	5808.03	1.29174	0.000382942	18
0	3.27303	-0.000273013	2.14117	792.583	1.05974	6.60019×10^{-6}	2.

Out[ ]//TableForm=

0	2.51606	0.000713634	0.971947	255.849	1.00121	0.001
0	2.48472	0.000274045	1.11993	310.874	1.00001	0.00035898
0	2.47291	-0.000393024	1.16734	574.84	1.13236	0.000149975
0	2.8235	0.0000393307	1.24239	1529.06	1.10292	0.000199128
0	2.85223	-0.000391853	2.10841	606.595	0.661049	-0.000187971

Out[ ]//TableForm=

0	2.96309	0.00170589	1.33129	665.904	1.00705	0.00168772	2.66
0	2.7496	0.000718402	1.23395	363.863	1.00424	0.000686306	14.8
0	2.68841	0.00039793	1.32554	262973.	1.50993	0.000394757	19.3
0	3.38616	0.000198884	1.3169	6425.4	1.3151	0.000385129	29.4
0	3.78584	-0.00025997	2.1781	816.066	1.49062	0.000588808	13.1

C10

In[•]:=
```
Transpose[simParamMedianC10] // TableForm
Transpose[simParamQuantile1C10] // TableForm
Transpose[simParamQuantile2C10] // TableForm
```

Out[•]//TableForm=

0	7.67019	0.0002	1.20006	210.934	1.00267	0.0002	1
0	7.82497	0.0005	1.19405	351.536	1.00013	0.000674773	1
0	8.81587	-0.0001	1.2251	1265.03	1.10343	0.000184686	2
0	8.45286	8.70425×10^{-17}	1.28546	2234.91	1.15064	0.0000781728	2
0	8.25616	-0.000305627	2.1229	705.538	1.02153	0.000141093	2

Out[•]//TableForm=

0	7.13307	0.0002	1.01403	94.1928	1.	0.0002
0	7.72649	0.0004	1.09039	278.443	0.999308	0.0005
0	6.95516	-0.000120999	1.21723	1034.09	1.10233	0.000172789
0	7.6985	-0.0000252389	1.21827	1537.79	1.09062	0.0000404502
0	6.55323	-0.000363606	2.09344	692.184	0.805212	0.0000494741

Out[•]//TableForm=

0	9.18484	0.000718121	1.89261	296.268	1.00335	0.0007	1.88
0	8.54384	0.001	1.33406	422.948	1.00688	0.0011	2.66
0	10.2249	0.0002	1.23662	2119.65	1.13915	0.0002	18.6
0	10.5048	0.0001	1.29292	3207.42	1.19839	0.00016319	33.3
0	10.0342	-0.0002	2.13017	751.212	1.04254	0.000485957	2.22

D

In[•]:=
```
Transpose[simParamMedianD] // TableForm
Transpose[simParamQuantile1D] // TableForm
Transpose[simParamQuantile2D] // TableForm
```

Out[•]//TableForm=

0	0.182211	0.000756457	1.23782	233.591	1.06421	0.000729688	0
0	0.160482	0.000655076	1.15075	392.888	1.05382	0.000601563	0
0	0.760542	0.0000509594	1.20625	1852.55	5.19154	0.0001642	1
0	1.25495	-0.000127826	1.29544	1218.95	8.52178	0.000148444	2
0	0.89184	-0.0002	2.12744	829.215	4.82759	0.0000811523	2

Out[•]//TableForm=

0	0.175917	0.00071855	1.23443	227.397	1.04785	0.000679706	0.
0	0.150673	0.00053524	1.14136	349.506	1.00562	0.000474348	0.
0	0.746446	0.000015852	1.20466	1651.99	5.11262	0.00015316	5.
0	1.24511	-0.000155294	1.28948	1161.59	8.21177	0.000142152	1.
0	0.812283	-0.00021586	2.12151	813.33	4.58807	0.000079853	2.

Out[•]//TableForm=

0	0.192411	0.000844248	1.2548	242.66	1.07016	0.000816872
0	0.163452	0.000814924	1.19442	436.992	1.05765	0.000785753
0	0.773428	0.0000836901	1.20636	1926.33	5.55399	0.000171345
0	1.3488	-0.0000966015	1.30082	1316.79	8.85177	0.000149284
0	0.92757	-0.0002	2.12853	835.937	5.56303	0.0000919688

Nv

In[•]:=
```
Transpose[simParamNv] // TableForm
```

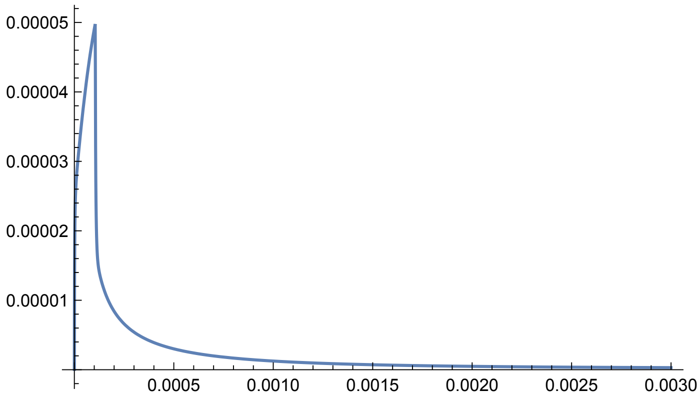
Out[•]//TableForm=

1.15733×10^{-8}	4.0267×10^{-7}	0.0000966667	0.00123275	0.00248568	0.01616
0.0000223798	0.000687618	0.0913074	0.791128	1.08299	1.8861
0.0000466775	0.00129019	0.14679	0.952943	1.13109	1.93094
0.0119635	0.160989	1.01594	1.17143	1.33522	2.61354
0.0329205	0.318746	1.03695	1.24341	1.45782	2.95342
0.188836	0.769426	1.37729	2.06583	2.21762	4.80775

EPSC with different caRest

In[•]:=
```
locaCa = Transpose[{dataLocalCaTime, dataLocalCa}];
locaCaWithoutdublictes = Mean /@ GatherBy[locaCa, First];
interpolFunc = Interpolation[locaCaWithoutdublictes, InterpolationOrder → 1];
caFunc[t_] := interpolFunc[t];
Plot[caFunc[t], {t, 0.00, 0.003}, PlotRange → All]
```

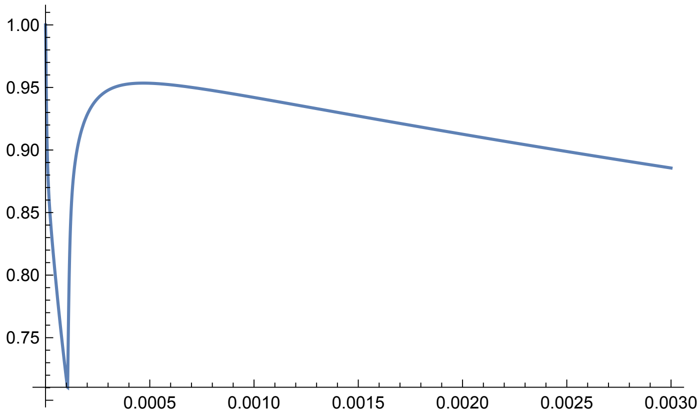
Out[•]=



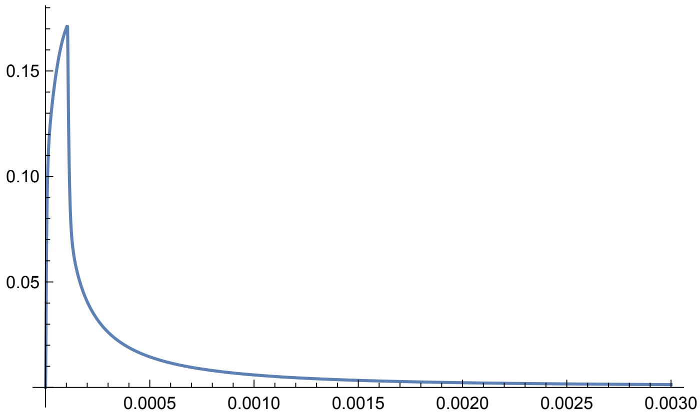
NDSolve

```
In[•]:= timeStartForPLot = 0.0;
timeEndForPLot = 0.003;
myNDSolveResults = NDSolve[eq, {ss1A, ss2A, ss3A, ss4A, ss5A, ss6A,
    ss7A, ss1B, ss2B, ss3B, ss4B, ss5B, ss6B, ss7B}, {t, 0, 0.003}];
Plot[(ss1A[t] /. myNDSolveResults), {t, timeStartForPLot, timeEndForPLot},
 PlotRange → All]
Plot[(ss2A[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss3A[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss4A[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss5A[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss6A[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss7A[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss1B[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss2B[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss3B[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss4B[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss5B[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss6B[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss7B[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
```

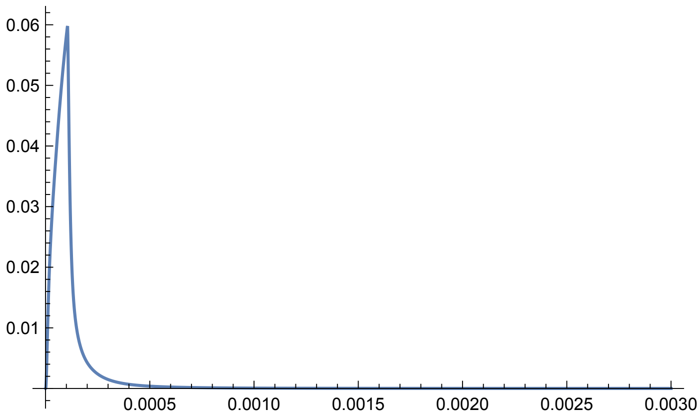
Out[•]=



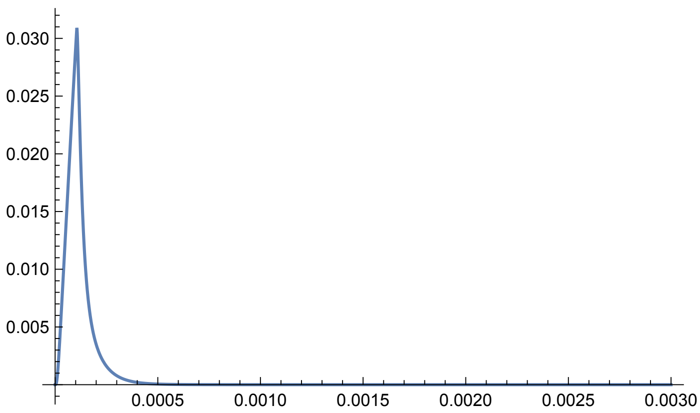
Out[•]=



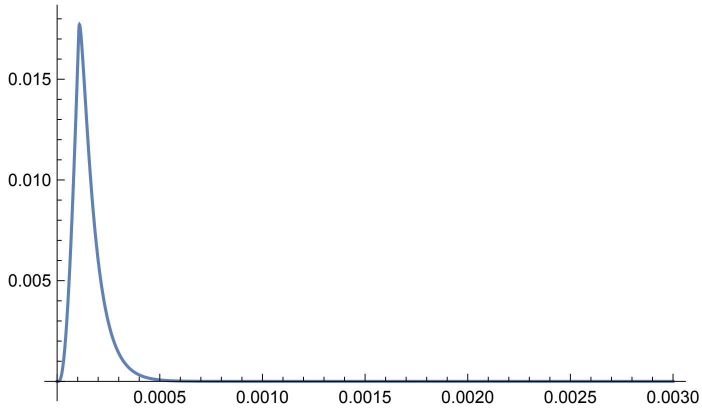
Out[•]=



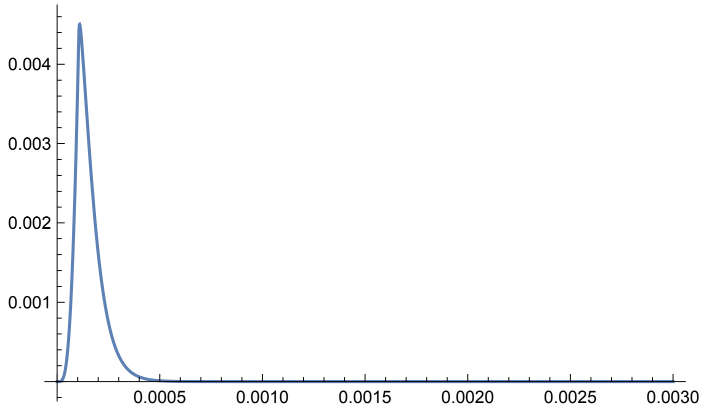
Out[•]=



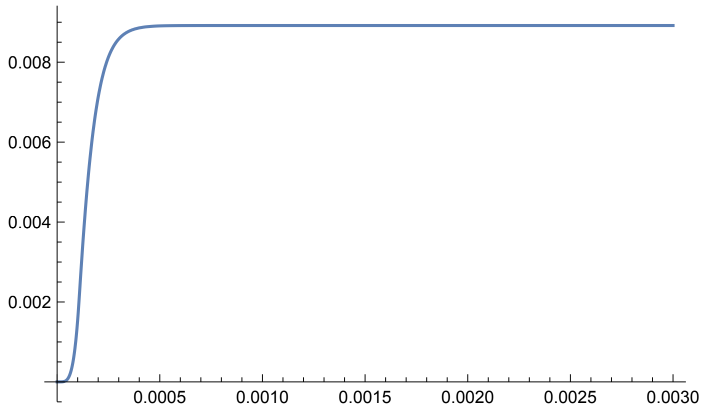
Out[∘]=



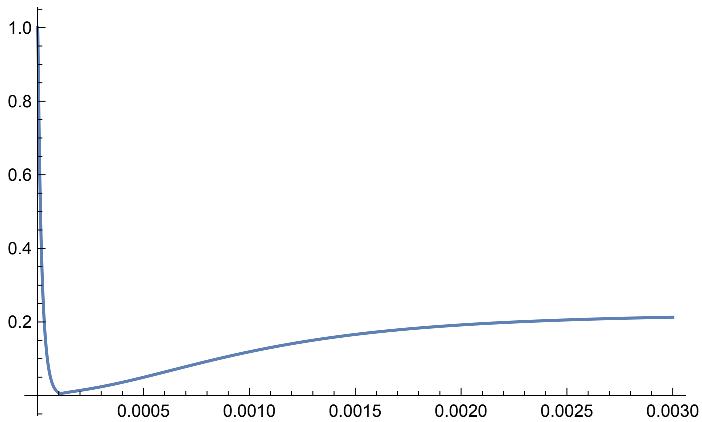
Out[∘]=

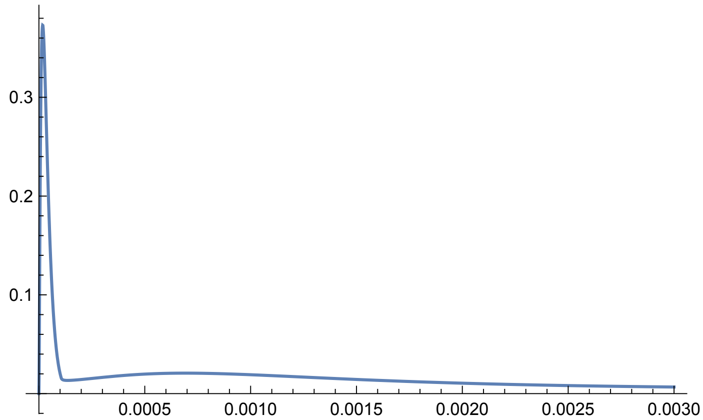


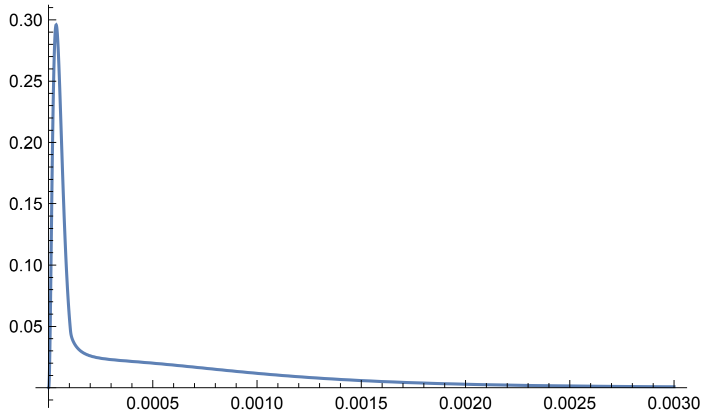
Out[∘]=

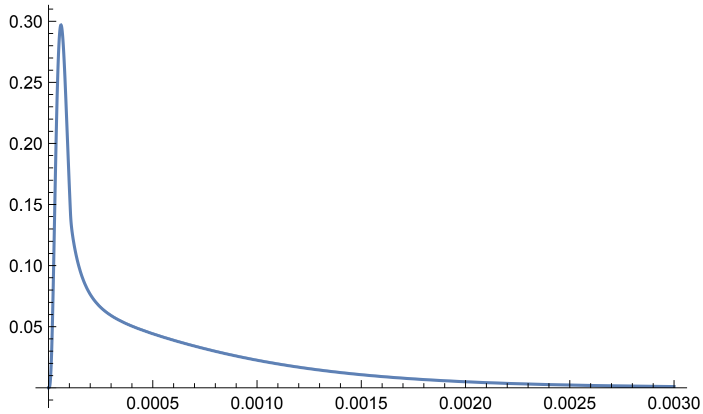


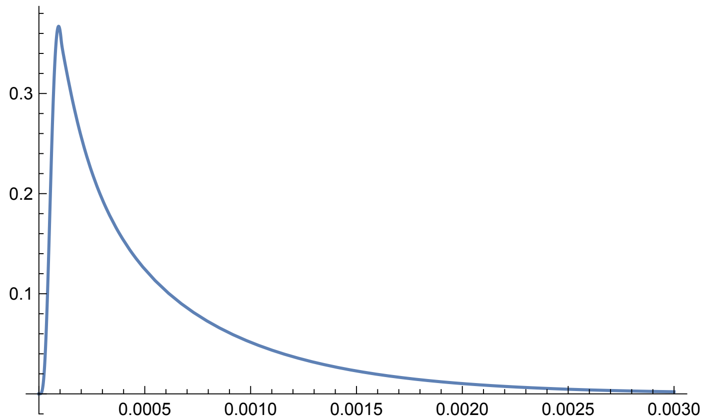
Out[∘]=



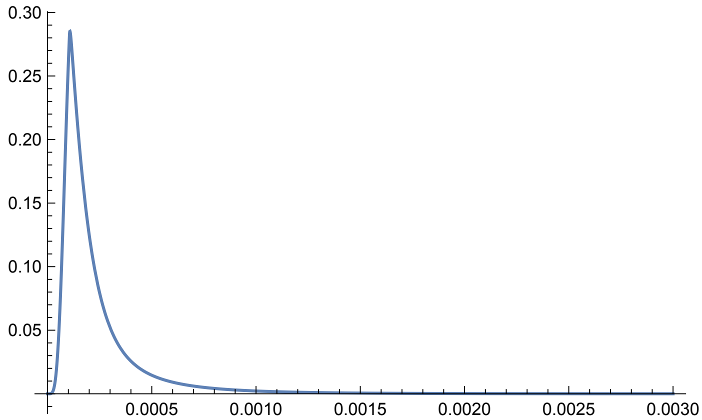
Out[ ]=


Out[ ]=


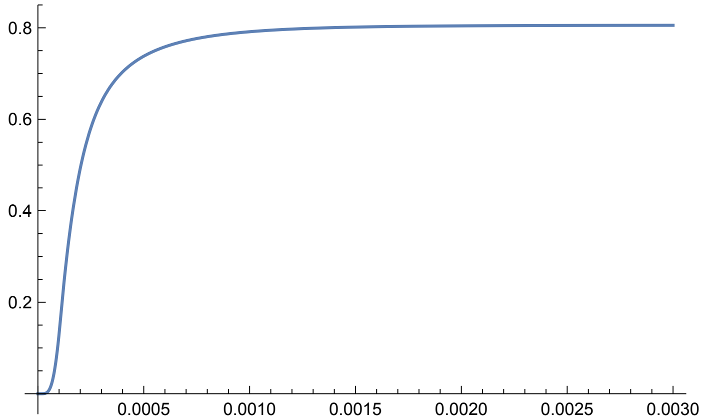
Out[ ]=


Out[ ]=


Out[ ]=



Out[ ]=



different caRest

In[ ]:=
```
caRestLow = 30*^-9;
caRestHigh = 180*^-9;
```

Low Ca

Initial occupancy

In[ ]:=
```
(*calualte initial equilibrium occupancy*)
caFunc[t_] := caRestLow;
ss0AInitial = kprimSchemeA / kunprimSchemeA
ss0BInitial = (kprimSchemeA / kunprimSchemeA) * (kprimSchemeB / kunprimSchemeB)
```

Out[ ]= 1.

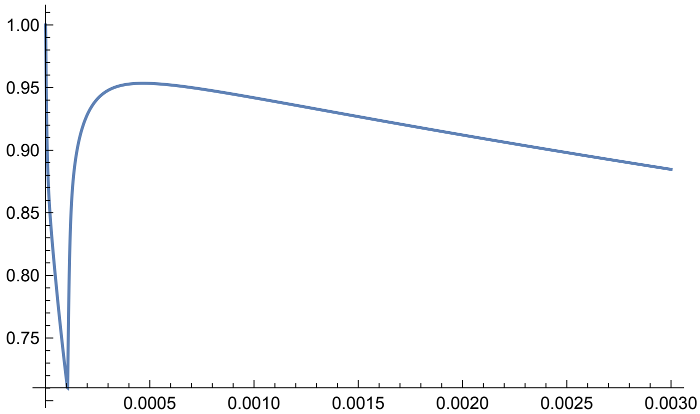
Out[ ]= 0.26514

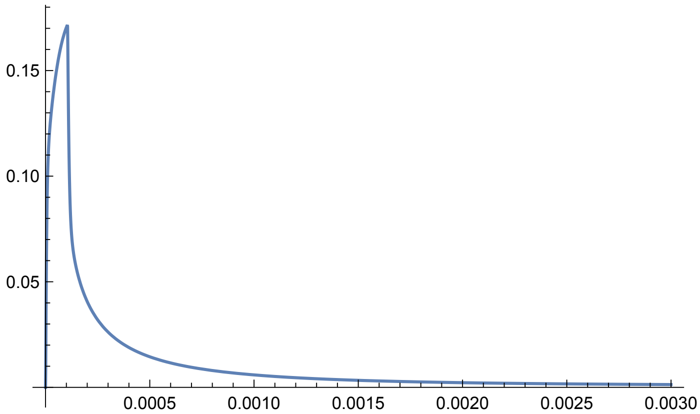
Diff eq.

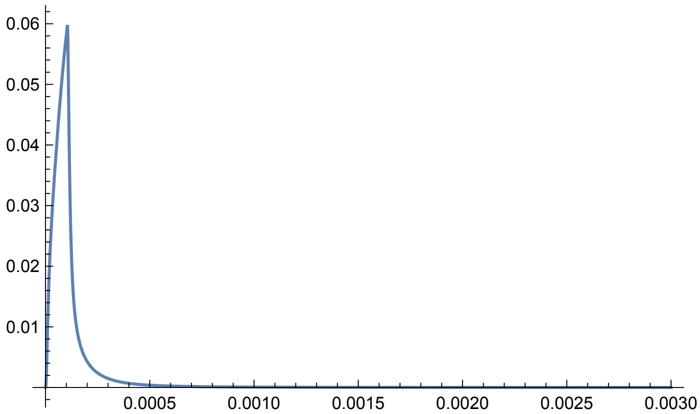
In[•]:=
```
Clear[caFunc, eq];(*Clear is needed if the cell is exectued for a 2nd time
 when caFunc is already set to a value or an Interpolationfunction*)
caFunc[t_] := interpolFunc[t];
ssA[t_] = {ss1A[t], ss2A[t], ss3A[t], ss4A[t], ss5A[t], ss6A[t], ss7A[t]};
ssB[t_] = {ss1B[t], ss2B[t], ss3B[t], ss4B[t], ss5B[t], ss6B[t], ss7B[t]};
eq = {
   ssA'[t] == (matA /. repl).ssA[t],
   ssA[0] == {ss0AInitial, 0, 0, 0, 0, 0, 0},

   ssB'[t] == (matB /. repl).ssB[t],
   ssB[0] == {ss0BInitial, 0, 0, 0, 0, 0, 0}
  };
```

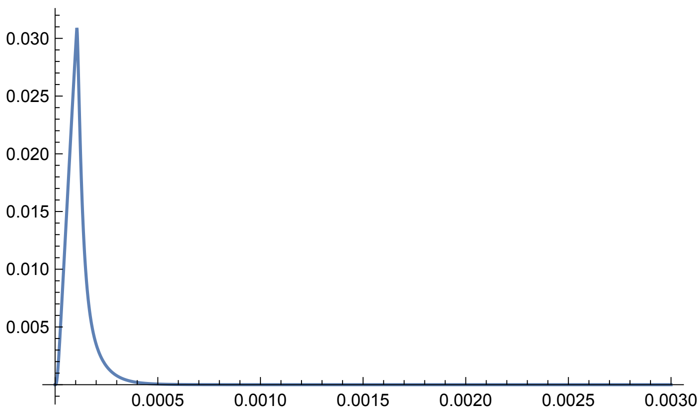
NDSolve

```
In[•]:= myNDSolveResults = NDSolve[eq, {ss1A, ss2A, ss3A, ss4A, ss5A, ss6A,
        ss7A, ss1B, ss2B, ss3B, ss4B, ss5B, ss6B, ss7B}, {t, 0, 0.003}];
      Plot[(ss1A[t] /. myNDSolveResults), {t, timeStartForPLot, timeEndForPLot},
        PlotRange → All]
      Plot[(ss2A[t] /. myNDSolveResults),
        {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
      Plot[(ss3A[t] /. myNDSolveResults),
        {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
      Plot[(ss4A[t] /. myNDSolveResults),
        {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
      Plot[(ss5A[t] /. myNDSolveResults),
        {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
      Plot[(ss6A[t] /. myNDSolveResults),
        {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
      Plot[(ss7A[t] /. myNDSolveResults),
        {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
      Print["-----------------------------------------------------------------"];
      Plot[(ss1B[t] /. myNDSolveResults),
        {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
      Plot[(ss2B[t] /. myNDSolveResults),
        {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
      Plot[(ss3B[t] /. myNDSolveResults),
        {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
      Plot[(ss4B[t] /. myNDSolveResults),
        {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
      Plot[(ss5B[t] /. myNDSolveResults),
        {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
      Plot[(ss6B[t] /. myNDSolveResults),
        {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
      Plot[(ss7B[t] /. myNDSolveResults),
        {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
```

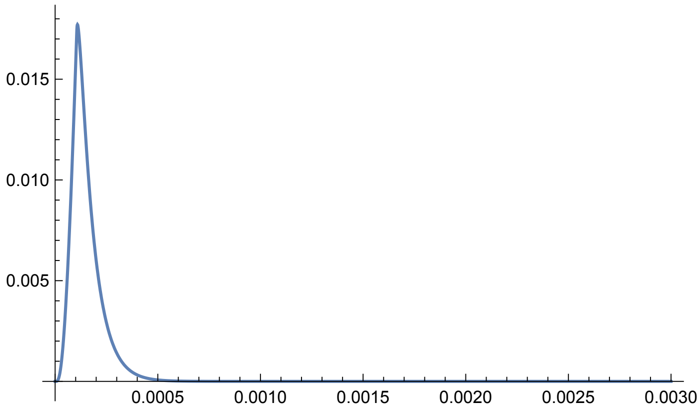
Out[∘]=


Out[∘]=


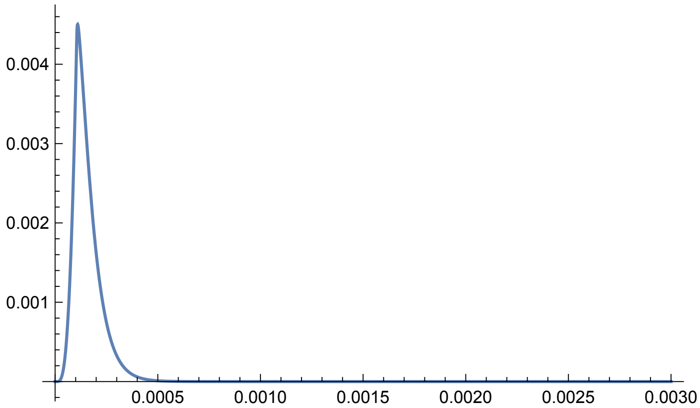
Out[∘]=


Out[∘]=


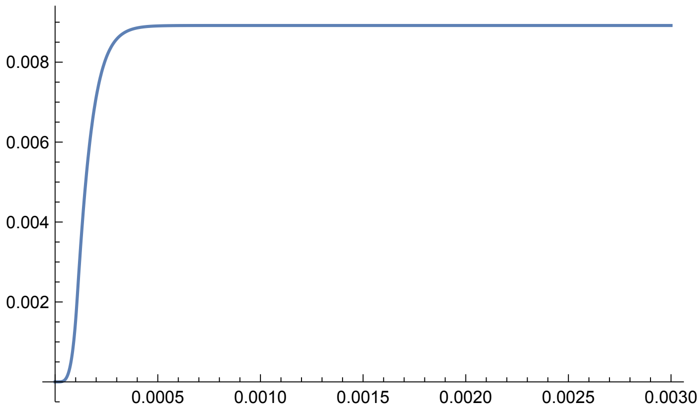
Out[•]=



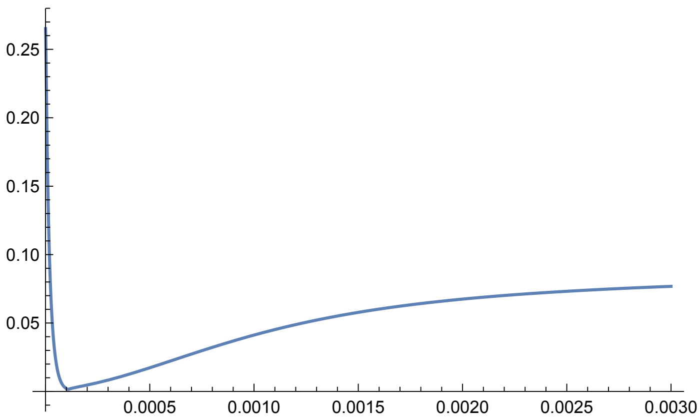
Out[•]=



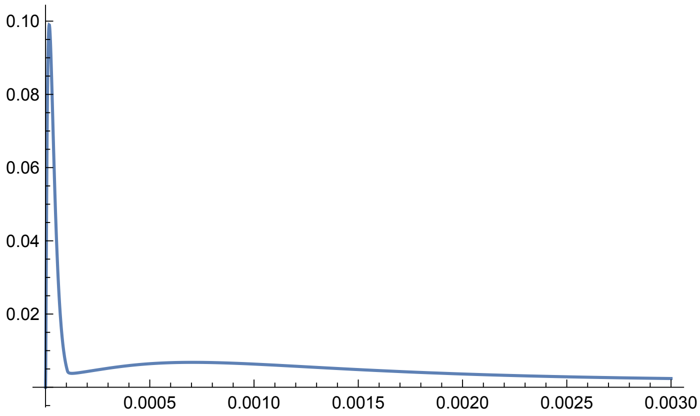
Out[•]=



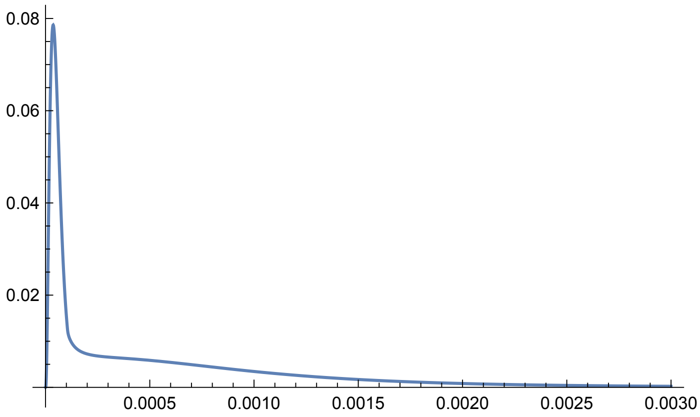
Out[•]=



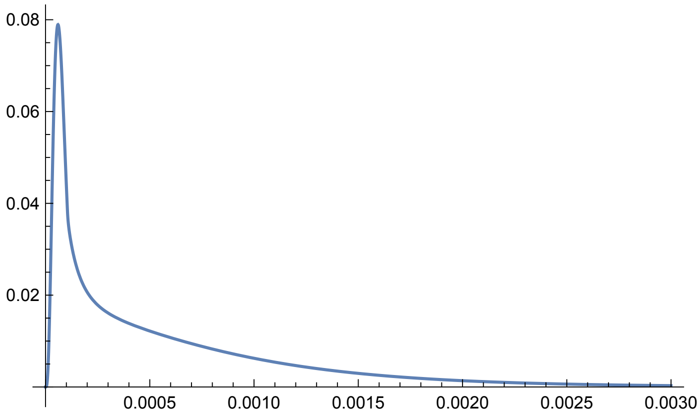
Out[•]=



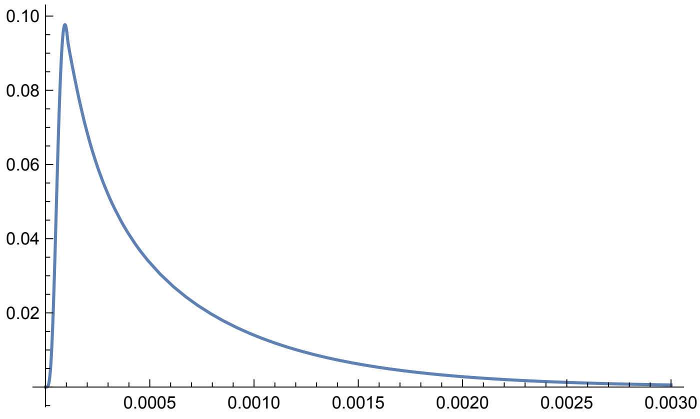
Out[•]=

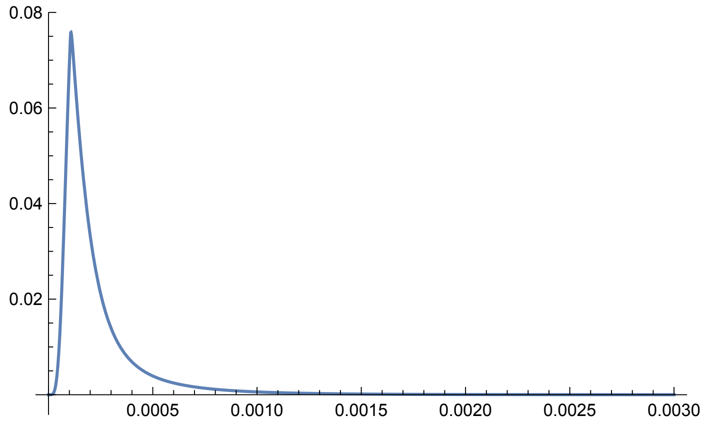


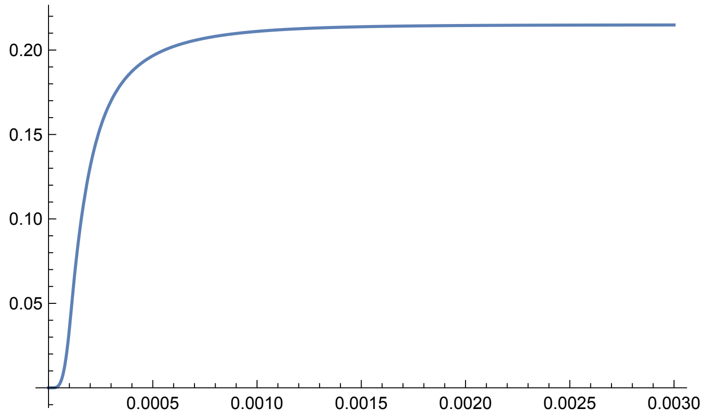
Out[•]=



Out[•]=

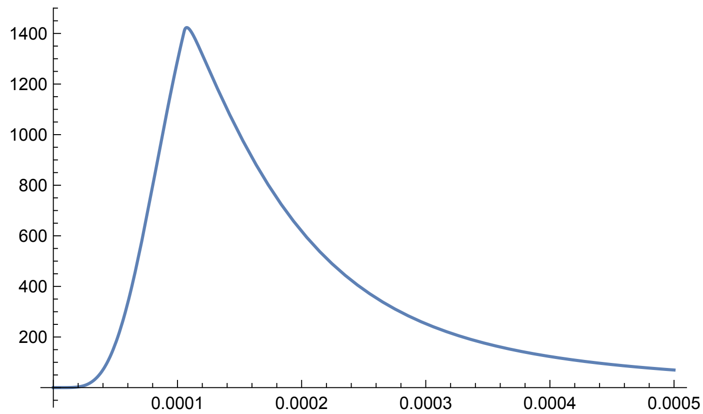


Out[•]=


Out[•]=


Plot EPSC

In[•]:=
```
epscLowCa = D[((ss7A[t] + ss7B[t]) /. myNDSolveResults), t];
Plot[epscLowCa, {t, 0, 2.*^-3}, PlotRange → All];
Plot[epscLowCa, {t, 0, 0.5*^-3}, PlotRange → All]
```

Out[•]=


High Ca

Initial occupancy

In[•]:=
```
(*calualte initial equilibrium occupancy*)
caFunc[t_] := caRestHigh;
ss0AInitial = kprimSchemeA / kunprimSchemeA
ss0BInitial = (kprimSchemeA / kunprimSchemeA) * (kprimSchemeB / kunprimSchemeB)
```

Out[•]= 1.

Out[•]= 0.836742

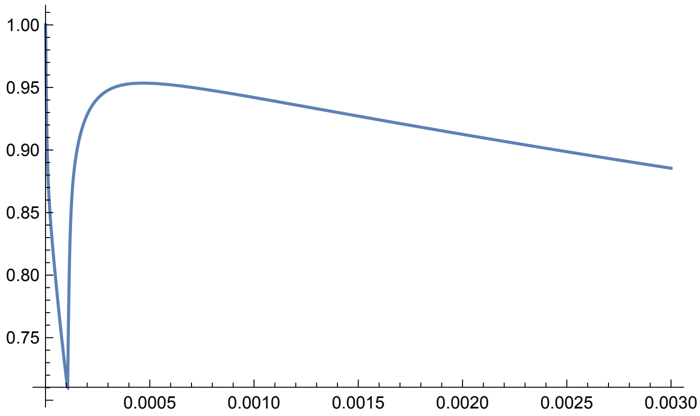
Diff eq.

In[•]:=
```
Clear[caFunc, eq];(*Clear is needed if the cell is exectued for a 2nd time
 when caFunc is already set to a value or an Interpolationfunction*)
caFunc[t_] := interpolFunc[t];
ssA[t_] = {ss1A[t], ss2A[t], ss3A[t], ss4A[t], ss5A[t], ss6A[t], ss7A[t]};
ssB[t_] = {ss1B[t], ss2B[t], ss3B[t], ss4B[t], ss5B[t], ss6B[t], ss7B[t]};
eq = {
   ssA'[t] == (matA /. repl).ssA[t],
   ssA[0] == {ss0AInitial, 0, 0, 0, 0, 0, 0},

   ssB'[t] == (matB /. repl).ssB[t],
   ssB[0] == {ss0BInitial, 0, 0, 0, 0, 0, 0}
  };
```

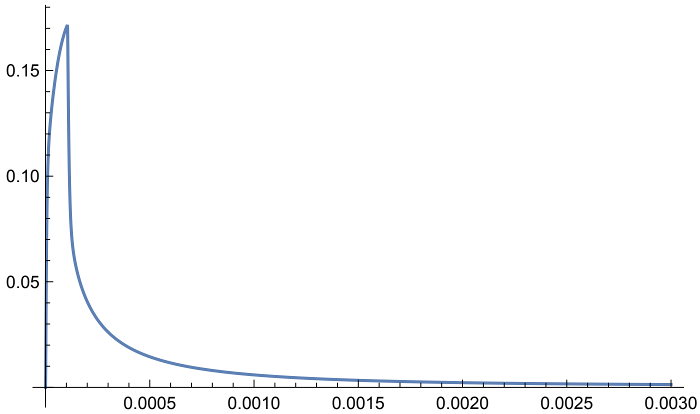
NDSolve

In[•]:=
```
myNDSolveResults = NDSolve[eq, {ss1A, ss2A, ss3A, ss4A, ss5A, ss6A,
    ss7A, ss1B, ss2B, ss3B, ss4B, ss5B, ss6B, ss7B}, {t, 0, 0.003}];
Plot[(ss1A[t] /. myNDSolveResults), {t, timeStartForPLot, timeEndForPLot},
 PlotRange → All]
Plot[(ss2A[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss3A[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss4A[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss5A[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss6A[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss7A[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Print["----------------------------------------------------------------"];
Plot[(ss1B[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss2B[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss3B[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss4B[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss5B[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss6B[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss7B[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
```

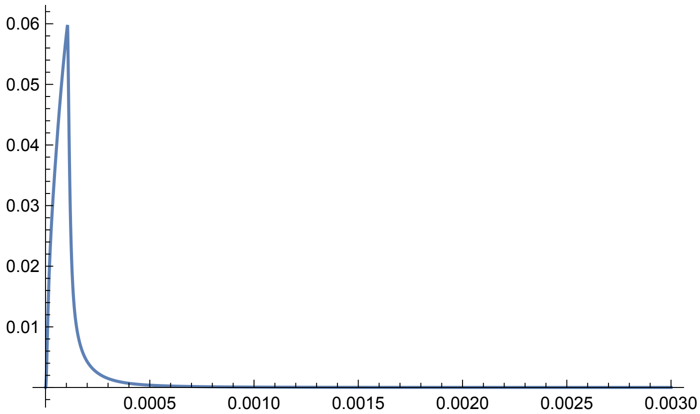
Out[•]=



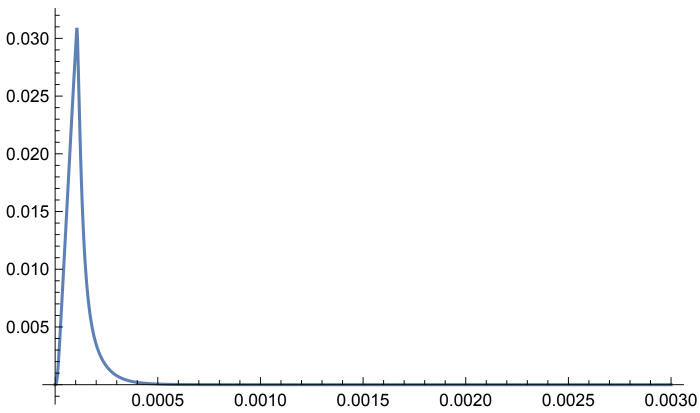
Out[•]=



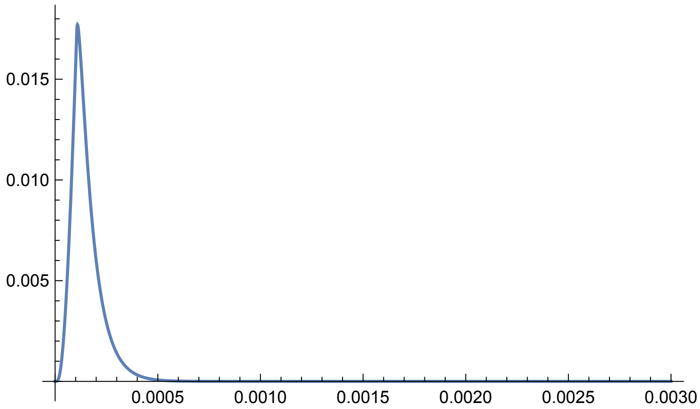
Out[•]=



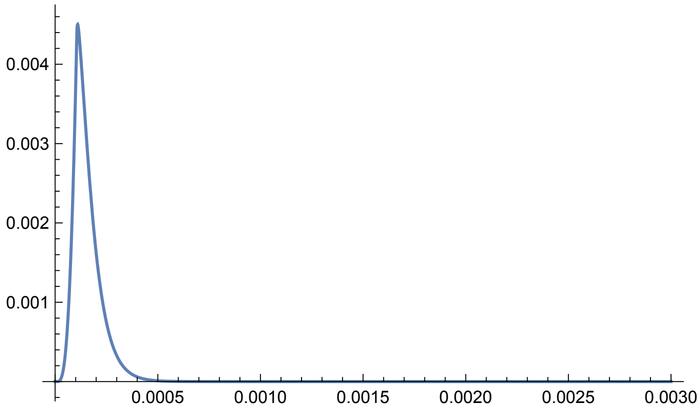
Out[•]=



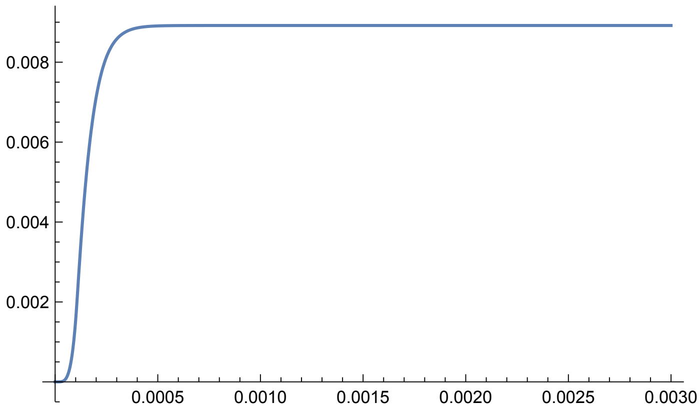
Out[•]=



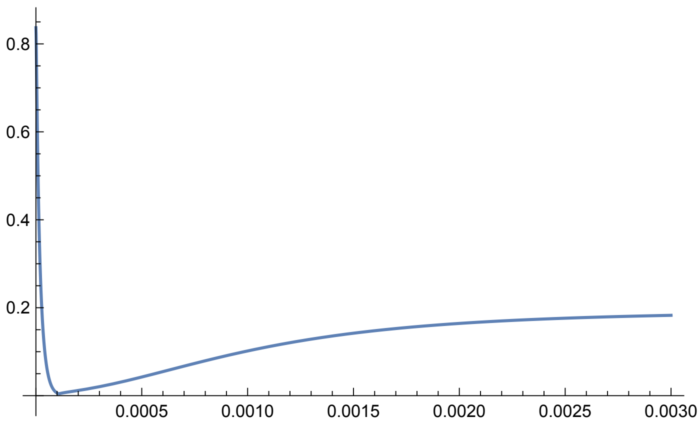
Out[•]=



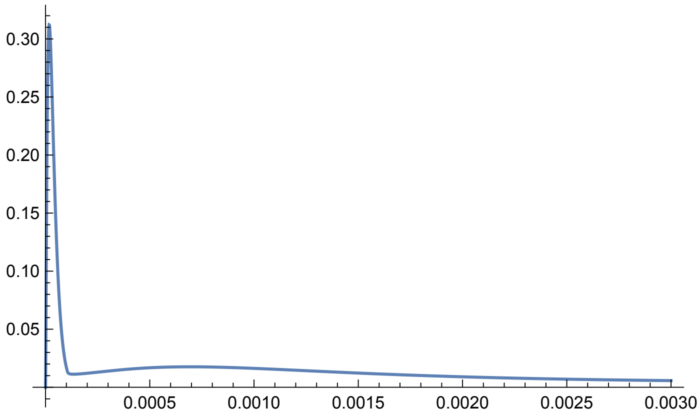
Out[•]=



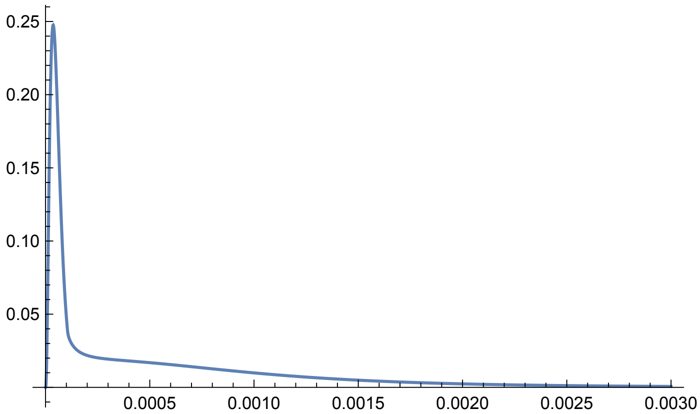
Out[•]=



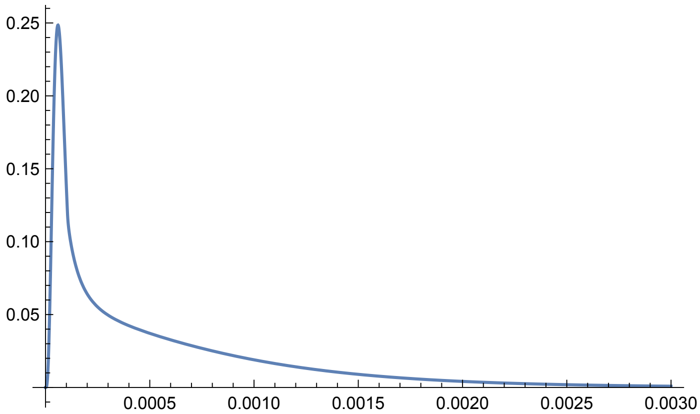
Out[•]=



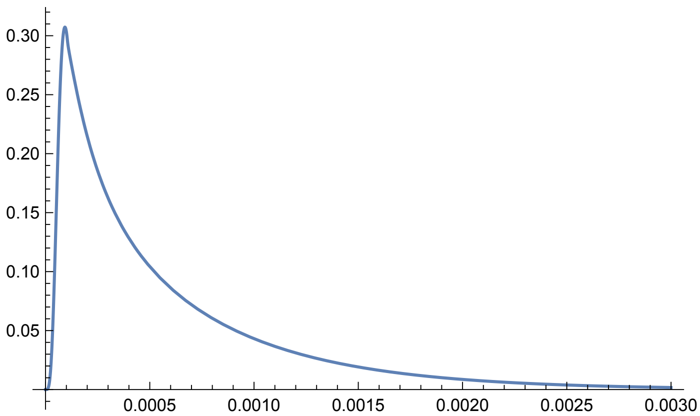
Out[•]=

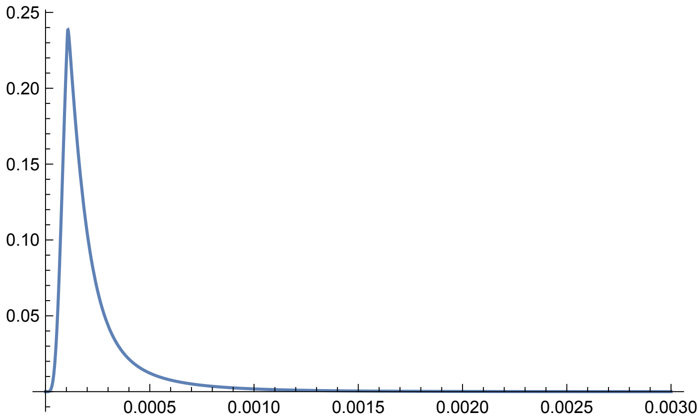


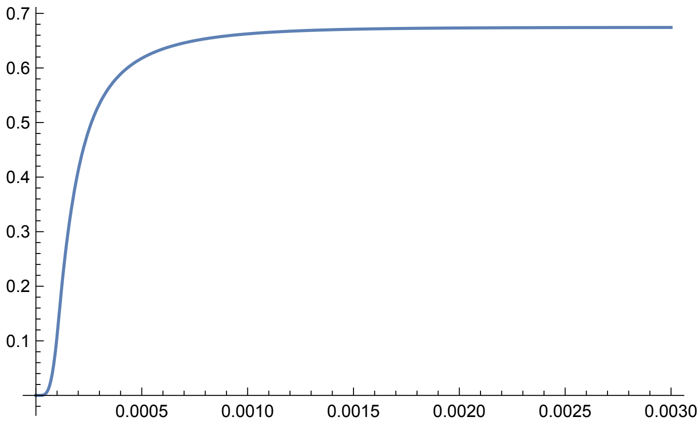
Out[•]=



Out[•]=

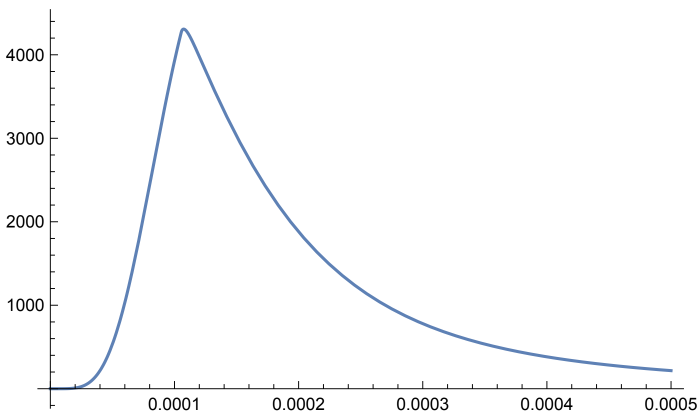


Out[ ]=


Out[ ]=


Plot EPSC

In[ ]:=
```
epscHighCa = D[((ss7A[t] + ss7B[t]) /. myNDSolveResults), t];
Plot[epscHighCa, {t, 0, 2*^-3}, PlotRange → All];
Plot[epscHighCa, {t, 0, 0.5*^-3}, PlotRange → All]
```

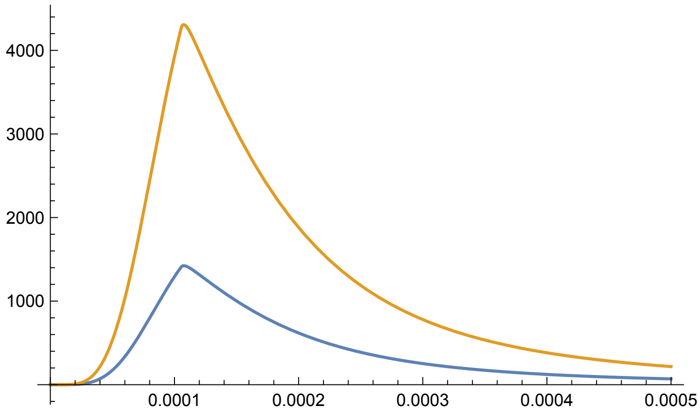
Out[ ]=


Compare

Plot both release rates

In[∘]:=
```
Plot[{epscLowCa, epscHighCa}, {t, 0, 0.5*^-3}, PlotRange → All]
```

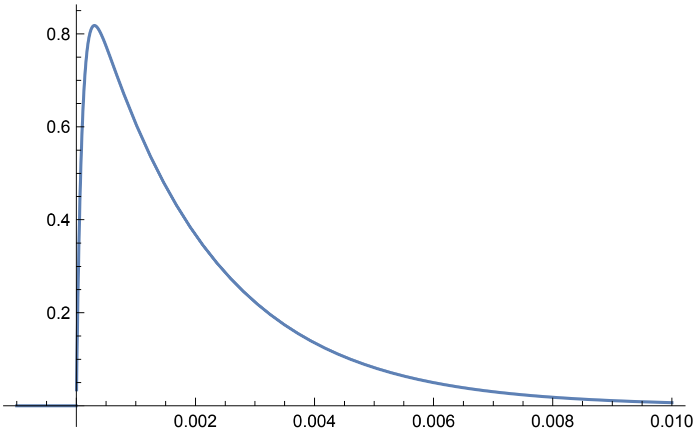
Out[∘]=



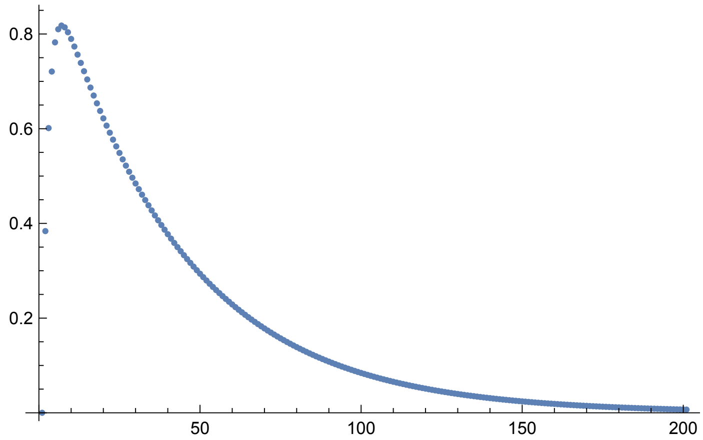
Convolution RelRate => EPSC

In[∘]:=
```
miniKernel[t_] := If[t ≤ 0, 0, (1 - Exp[-t / 0.0001]) * Exp[-t / 0.002]];
Plot[miniKernel[t], {t, -.001, .01}]
dtForConvolve = 0.00005;
tEndConv = 0.01;
miniKernelList = Table[miniKernel[t], {t, 0.0, tEndConv, dtForConvolve}];
epscHighCaList = {Table[0, {t, 0, tEndConv, dtForConvolve}],
   Table[epscHighCa, {t, 0.0, 0.0025, dtForConvolve}],
   Table[0, {t, 0, tEndConv, dtForConvolve}]} // Flatten;
epscLowCaList = {Table[0, {t, 0, tEndConv, dtForConvolve}],
   Table[epscLowCa, {t, 0.0, 0.0025, dtForConvolve}],
   Table[0, {t, 0, tEndConv, dtForConvolve}]} // Flatten;
ListPlot[miniKernelList]
ListPlot[epscLowCaList, PlotRange → All]
```

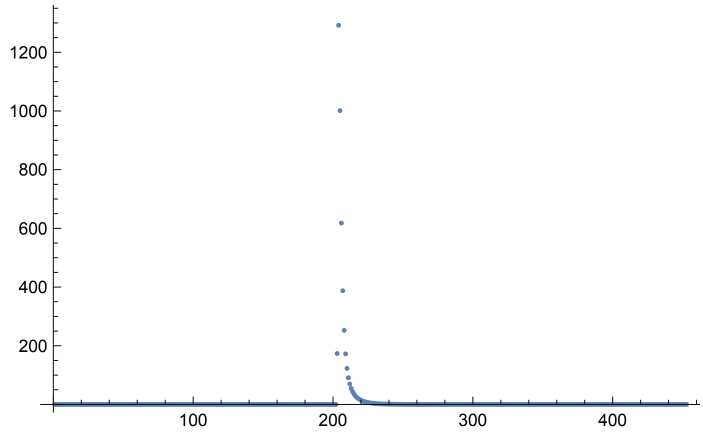
Out[∘]=



Out[ ]=



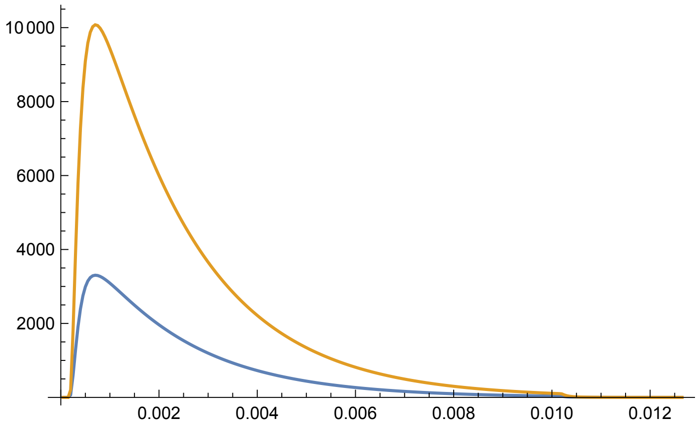
Out[ ]=



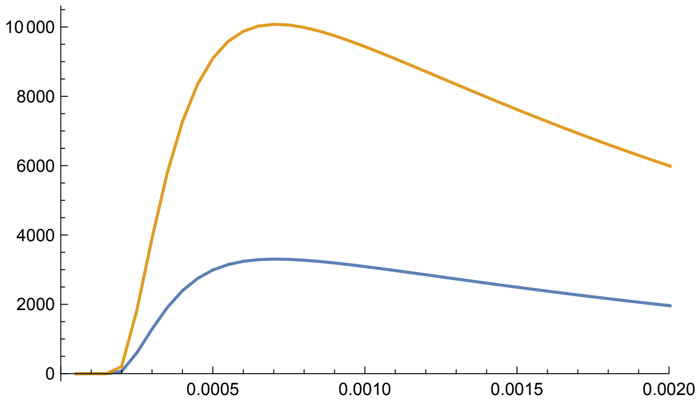
In[ ]:=

```
epscLowCaCurrentList = ListConvolve[miniKernelList, epscLowCaList];
epscHighCaCurrentList = ListConvolve[miniKernelList, epscHighCaList];
timeConv = Table[t * dtForConvolve, {t, Length[epscLowCaCurrentList]}];
ListPlot[{Transpose[{timeConv, epscLowCaCurrentList}],
  Transpose[{timeConv, epscHighCaCurrentList}]}, Joined → True, PlotRange → All]
ListPlot[{Transpose[{timeConv, epscLowCaCurrentList}],
  Transpose[{timeConv, epscHighCaCurrentList}]},
 Joined → True, PlotRange → {{0, 0.002}, All}]
maxLow = Max[epscLowCaCurrentList]
maxHigh = Max[epscHighCaCurrentList]
Print["maxHigh/maxLow = ", maxHigh / maxLow];
ListPlot[{Transpose[{timeConv, (1 / maxLow) * epscLowCaCurrentList}],
  Transpose[{timeConv, (1 / maxHigh) * epscHighCaCurrentList}]},
 Joined → True, PlotRange → All]
ListPlot[{Transpose[{timeConv, (1 / maxLow) * epscLowCaCurrentList}],
  Transpose[{timeConv, (1 / maxHigh) * epscHighCaCurrentList}]},
 Joined → True, PlotRange → {{0, 0.002}, All}]
If[exportYes == 1,
  toExport = Transpose[{timeConv, epscLowCaCurrentList, epscHighCaCurrentList,
     (1 / maxLow) * epscLowCaCurrentList, (1 / maxHigh) * epscHighCaCurrentList}];
  Export["plot EPSC - low and high - abs and norm.txt", toExport, "Table"];
 ];
```

Out[•]=



Out[•]=

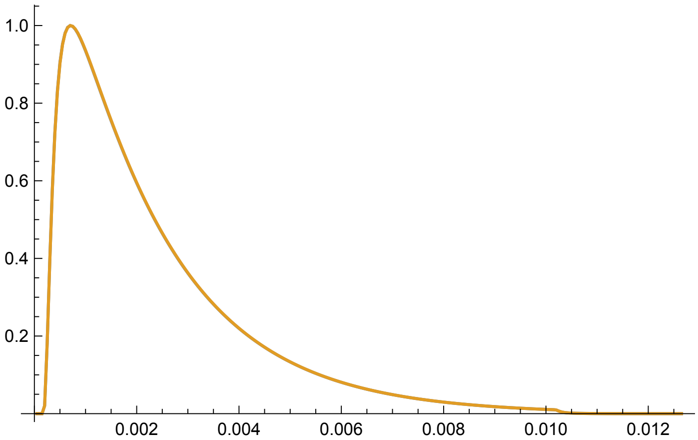


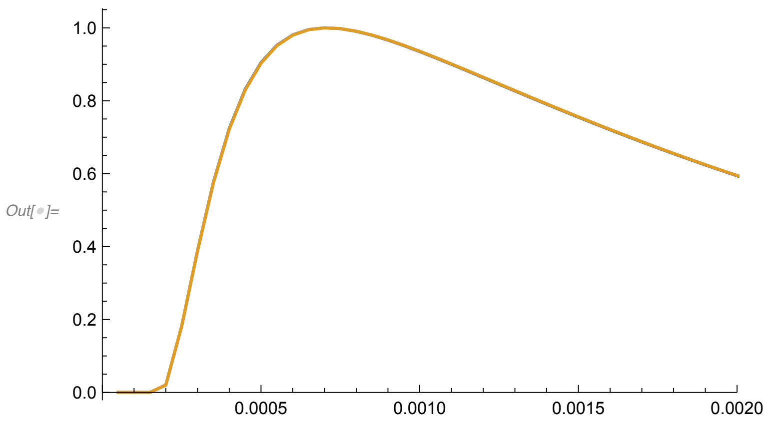
Out[•]= 3304.39

Out[•]= 10 076.8

maxHigh/maxLow = 3.0495

Out[•]=





Timing

In[•]:=
```
timeEnd = AbsoluteTime[]
(timeEnd - timeStart) / 60.(*time of calculation in min*)
```

Out[•]= $3.839313758445628 \times 10^9$

Out[•]= 0.298439