```
(* Mathematica code for Model2 of Eshra et al. eLife 2021 *)
(* Stefan Hallermann and Hartmut Schmidt Aug 2021 *)
```

Import

general

```
(* imported data all Ca in uM, all times in ms,
all ampltiudes and Nv in vesicels *)
CmToVesConversionFactor = (1 / 90.12) * (1 / 70*^-18);(* explained in methods *)
rrp = 10;(* pool of release-ready vesicles per connection *)
dir = NotebookDirectory[];
SetDirectory[dir];
dataFolder = "../data to fit/";
```

tau1 Cm 5kHz

```
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"];
```

```
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

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

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

tau2 Cm 10kHz

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

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

tau2 Deconv

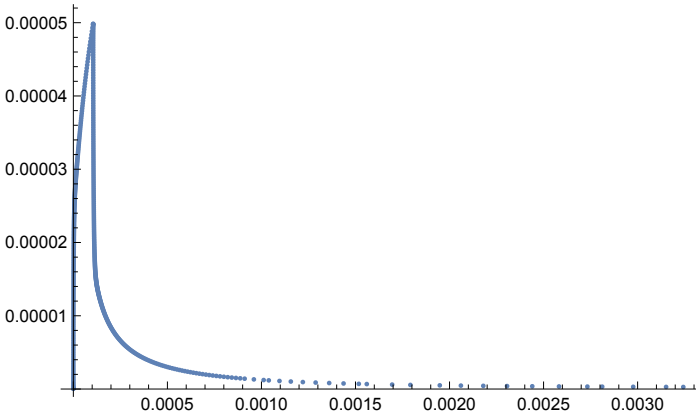
```
data = Import[dataFolder <> "all_t2_v02_D.txt", "Table"];
```

```
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[•]= $3.841510464468773 \times 10^{9}$

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;
```

```
In[•]:= pre_final01
Out[•]= pre_final01
```

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. * 31.4 / 24;]

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. * 31.4 / 24;]

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 ≤ 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*)
        ϒ = 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. * 31.4 / 24;]

        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 ≤ 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] *
```

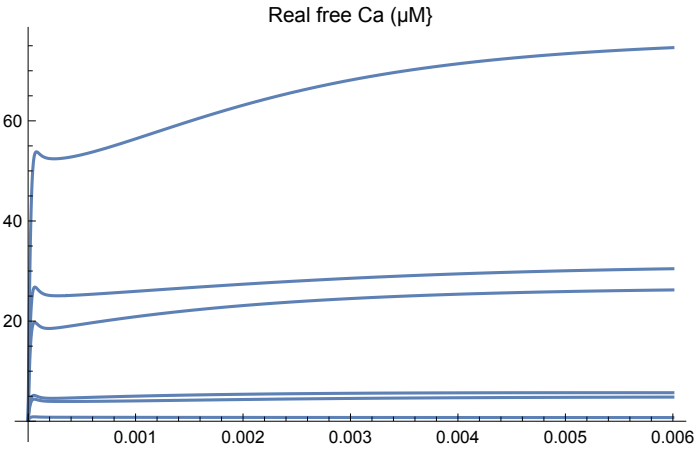
```
          UnitStep[tt - tflash] + 1 / taus * DMs[tt] * UnitStep[tt - tflash])
      + 1 / tauf * MgDMf[tt] * UnitStep[tt - tflash] +
      1 / taus * MgDMs[tt] * UnitStep[tt - tflash],

    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 := {
```

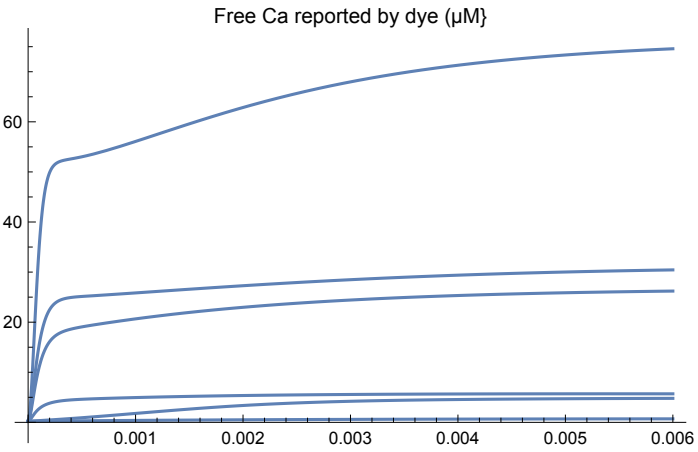
```
    Ca[0] == Ca0,

    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

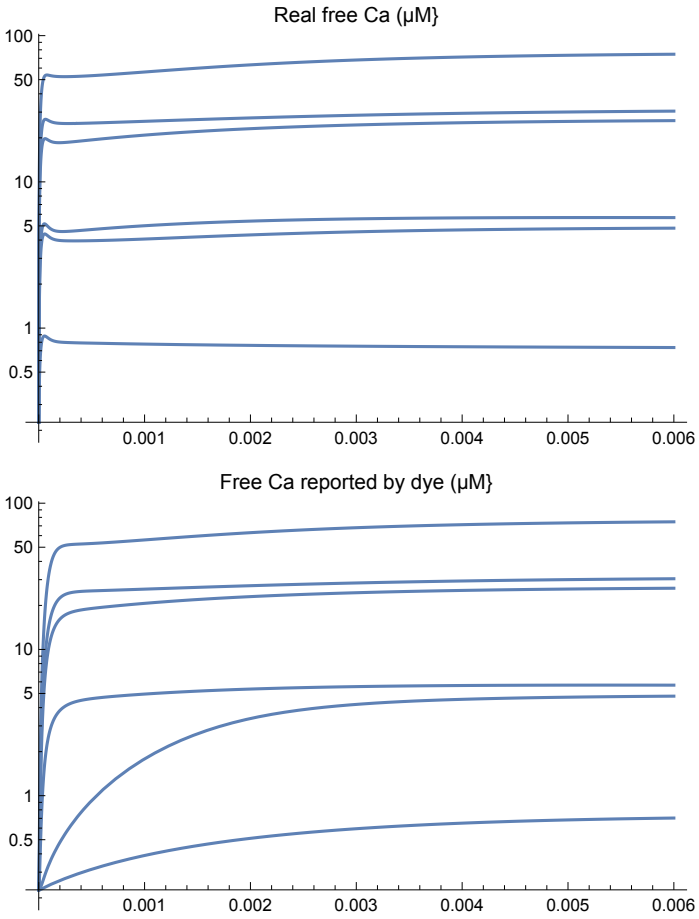
In[•]:=
```
Plot[10^6 * CaListReal, {tt, 0, TimeWindow}, PlotLabel → "Real free Ca (μM}"]
Plot[10^6 * CaListDye, {tt, 0, TimeWindow},
 PlotLabel → "Free Ca reported by dye (μM}"]
LogPlot[10^6 * CaListReal, {tt, 0, TimeWindow}, PlotLabel → "Real free Ca (μM}"]
LogPlot[10^6 * CaListDye, {tt, 0, TimeWindow},
 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[•]=





Out[•]=



Out[•]= $\{7.03073 \times 10^{-7}, 4.79194 \times 10^{-6}, 5.70182 \times 10^{-6},$
$0.0000262105, 0.0000304385, 0.0000745861\}$

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

Release scheme and parameters

Define release scheme

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

forward rates

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

from = 2;
kk = 4 kon;
mat[[from + 1, from]] += kk; mat[[from, from]] += -kk;

from = 3;
kk = 3 kon;
mat[[from + 1, from]] += kk; mat[[from, from]] += -kk;

from = 4;
kk = 2 kon;
mat[[from + 1, from]] += kk; mat[[from, from]] += -kk;

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

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

backwards rates

```
In[•]:= from = 2;(*1 ca bound*)
        kk = koff b^0;
        mat[[from - 1, from]] += kk; mat[[from, from]] += -kk;

        from = 3;
        kk = 2 koff b^1;
        mat[[from - 1, from]] += kk; mat[[from, from]] += -kk;

        from = 4;
        kk = 3 koff b^2;
        mat[[from - 1, from]] += kk; mat[[from, from]] += -kk;

        from = 5;
        kk = 4 koff b^3;
        mat[[from - 1, from]] += kk; mat[[from, from]] += -kk;

        from = 6;(*5 ca bound*)
        kk = 5 koff b^4;
        mat[[from - 1, from]] += kk; mat[[from, from]] += -kk;
```

outflux from matrix

```
In[•]:= mat[[1, 1]] += -kunfill;(* is multiplied by ss1[t] *)
```

influx in matrix

```
In[•]:= mat[[1, 1]] += (1 / ss1[t]) kfill fillStateSS[t];
        (* (1/ss1[t]) is needed to merge the fillState
         Diff Eq. in the matrix format. Note ss1[t] is never 0 *)
```

Matrix

```
In[•]:= mat // TableForm
```

Out[•]//TableForm=

$-5\,\text{kon} - \text{kunfill} + \frac{\text{kfill fillStateSS[t]}}{\text{ss1[t]}}$	koff	0	0
5 kon	$-\text{koff} - 4\,\text{kon}$	2 b koff	0
0	4 kon	$-2\,\text{b koff} - 3\,\text{kon}$	$3\,\text{b}^2\,\text{koff}$
0	0	3 kon	$-3\,\text{b}^2\,\text{koff} - 2\,\text{kc}$
0	0	0	2 kon
0	0	0	0
0	0	0	0

Parameters of release scheme

```
In[•]:= linInterpol[x_, a_, b_] := a + x * (b - a);

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

```
In[•]:= Clear[caFunc, kprimScheme];
(*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*)

affinityFactor = 3.0;
konScheme = linInterpol[sitePlugging[t],
   caFunc[t] Sqrt[affinityFactor] tempFact 1.*^8,
   0.1 caFunc[t] Sqrt[affinityFactor] tempFact 1.*^8
  ];(*M^-1 s^-1*)

koffScheme = linInterpol[sitePlugging[t],
   (1 / Sqrt[affinityFactor]) tempFact 15 000,
   0.4 (1 / Sqrt[affinityFactor]) tempFact 15 000
  ];(*s^-1*)
gammaScheme = tempFact 6000;(*s^-1*)
bScheme = 0.25;

KdPrim = 2.*^-6;
kprimScheme  = 2.5 + 60. (caFunc[t] / (KdPrim + caFunc[t]));
kunprimScheme = 2.5 + 60. (caRest / (KdPrim + caRest));

KdFill = 2.*^-6;
kfillScheme =  100 + 800. (caFunc[t] / (KdFill + caFunc[t]));
kunfillScheme = 100 + 800. (caRest / (KdFill + caRest));


repl = {
   kon → konScheme,
   koff → koffScheme ,
   gamma → gammaScheme ,
    b → bScheme,

   kprim → kprimScheme ,
   kunprim → kunprimScheme ,

   kfill → kfillScheme ,
   kunfill → kunfillScheme

   (*
   kbasal→ kbasalScheme ,
   kunfill→ kunfillScheme
   *)
  };

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

```
In[•]:= siteClearanceTau = 0.04;
```

Initial occupancy

```
In[•]:= (*test initial equilibrium occupancy*)
caFunc[t_] := caTmp;
fillStateSSInitial = kprimScheme / kunprimScheme;
ss0Initial = fillStateSSInitial * kfillScheme / kunfillScheme;
(ss0Initial /. caTmp → 180*^-9)
(ss0Initial /. caTmp → 50*^-9)
(ss0Initial /. caTmp → 30*^-9)
(ss0Initial /. caTmp → 180*^-9) / (ss0Initial /. caTmp → 30*^-9)
(*test initial *)
Plot[kprimScheme / kunprimScheme, {caTmp, 0, 300*^-9}]
Plot[kfillScheme / kunfillScheme, {caTmp, 0, 300*^-9}]
LogLinearPlot[{kprimScheme / kunprimScheme, kfillScheme / kunfillScheme},
 {caTmp, .1*^-9, 30*^-6}, PlotRange → All]
Plot[{kprimScheme / kunprimScheme, kfillScheme / kunfillScheme},
 {caTmp, .1*^-9, 1*^-6}, PlotRange → All]
```

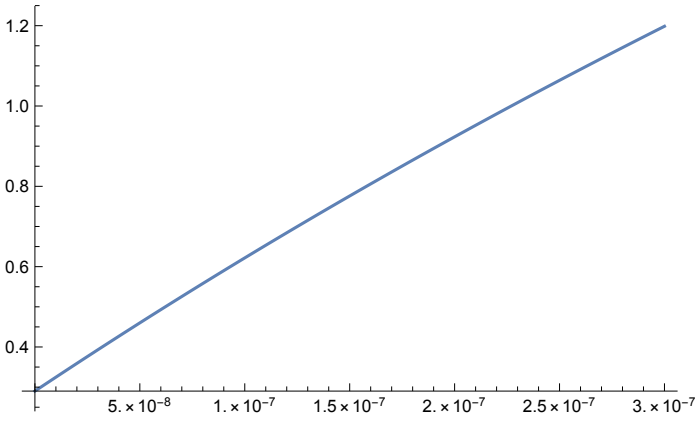
```
Out[•]= 0.791348
```

```
Out[•]= 0.302831
```

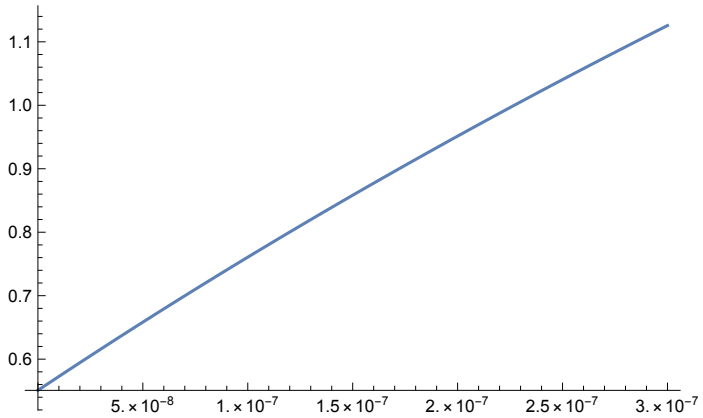
```
Out[•]= 0.242117
```

```
Out[•]= 3.26845
```

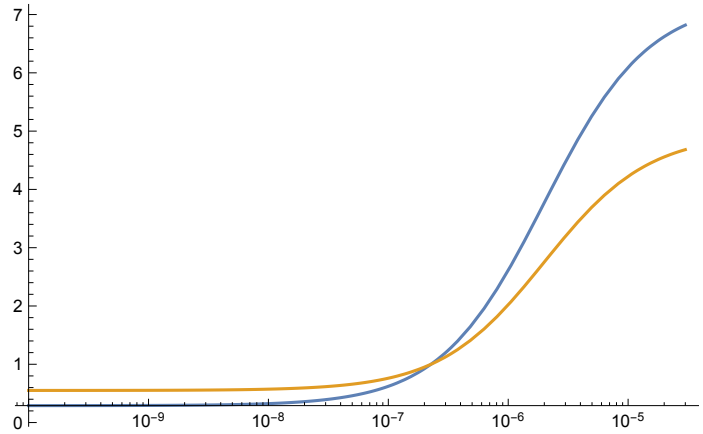
```
Out[•]=
```



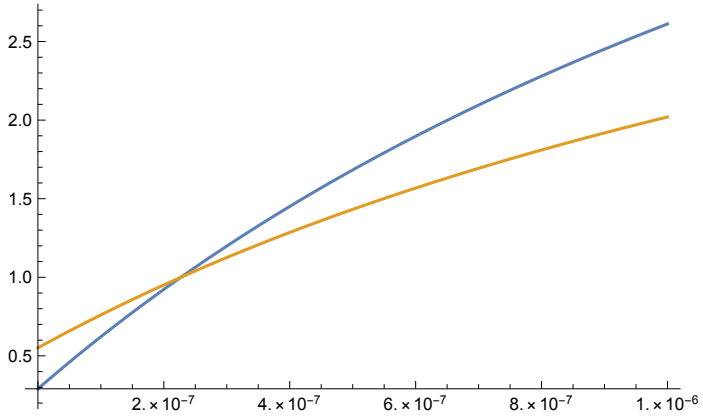
Out[•]=


1.1
1.0
0.9
0.8
0.7
0.6
5. × 10^-8
1. × 10^-7
1.5 × 10^-7
2. × 10^-7
2.5 × 10^-7
3. × 10^-7

Out[•]=


7
6
5
4
3
2
1
0
10^-9
10^-8
10^-7
10^-6
10^-5

Out[•]=


2.5
2.0
1.5
1.0
0.5
2. × 10^-7
4. × 10^-7
6. × 10^-7
8. × 10^-7
1. × 10^-6

```
In[•]:= (*calualte initial equilibrium occupancy*)
caFunc[t_] := caRest;
kprimScheme
kunprimScheme
fillStateSSInitial = kprimScheme / kunprimScheme
kfillScheme
kunfillScheme
ss0Initial = fillStateSSInitial * kfillScheme / kunfillScheme
```

```
Out[•]= 8.61585
```

```
Out[•]= 8.61585
```

```
Out[•]= 1.
```

```
Out[•]= 181.545
```

```
Out[•]= 181.545
```

```
Out[•]= 1.
```

```
In[•]:= caRest
```

$$\text{Out}[\bullet]= 2.27 \times 10^{-7}$$

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*)
ss[t_] = {ss1[t], ss2[t], ss3[t], ss4[t], ss5[t], ss6[t], ss7[t]};
eq = {ss'[t] == (mat /. repl).ss[t],
   ss[0] == {ss0Initial, 0., 0., 0., 0., 0., 0.},
   (fillStateSS'[t] == kprim - kunprim fillStateSS[t] -
       kfill fillStateSS[t] + kunfill ss1[t]) /. repl,
   fillStateSS[0] == fillStateSSInitial,
   sitePlugging'[t] ==
    (1 - sitePlugging[t]) ss7'[t] - siteClearanceTau sitePlugging[t],
   sitePlugging[0] == 0
  };
```

Print diff eq.

In[•]:= **mat**

Out[•]=
$$\left\{\left\{-5\,\text{kon}-\text{kunfill}+\frac{\text{kfill}\,\text{fillStateSS}[t]}{\text{ss1}[t]},\ \text{koff},\ 0,\ 0,\ 0,\ 0,\ 0\right\},\right.$$
$$\{5\,\text{kon},\ -\text{koff}-4\,\text{kon},\ 2\,b\,\text{koff},\ 0,\ 0,\ 0,\ 0\},$$
$$\{0,\ 4\,\text{kon},\ -2\,b\,\text{koff}-3\,\text{kon},\ 3\,b^2\,\text{koff},\ 0,\ 0,\ 0\},$$
$$\{0,\ 0,\ 3\,\text{kon},\ -3\,b^2\,\text{koff}-2\,\text{kon},\ 4\,b^3\,\text{koff},\ 0,\ 0\},$$
$$\{0,\ 0,\ 0,\ 2\,\text{kon},\ -4\,b^3\,\text{koff}-\text{kon},\ 5\,b^4\,\text{koff},\ 0\},$$
$$\left.\{0,\ 0,\ 0,\ 0,\ \text{kon},\ -\text{gamma}-5\,b^4\,\text{koff},\ 0\},\ \{0,\ 0,\ 0,\ 0,\ 0,\ \text{gamma},\ 0\}\right\}$$

In[•]:= **repl**

Out[•]=
$$\{\text{kon}\to 5.11454\times 10^8\,\text{caFunc}[t]-4.60309\times 10^8\,\text{caFunc}[t]\times\text{sitePlugging}[t],$$
$$\text{koff}\to 25\,572.7-15\,343.6\,\text{sitePlugging}[t],\ \text{gamma}\to 17\,717.3,$$
$$b\to 0.25,\ \text{kprim}\to 2.5+\frac{60.\,\text{caFunc}[t]}{2.\times 10^{-6}+\text{caFunc}[t]},\ \text{kunprim}\to 8.61585,$$
$$\text{kfill}\to 100+\frac{800.\,\text{caFunc}[t]}{2.\times 10^{-6}+\text{caFunc}[t]},\ \text{kunfill}\to 181.545\}$$

In[•]:= **mat /. repl**

Out[•]=
$$\left\{\left\{-181.545-5\left(5.11454\times 10^8\,\text{caFunc}[t]-4.60309\times 10^8\,\text{caFunc}[t]\times\text{sitePlugging}[t]\right)+\right.\right.$$
$$\left.\frac{\left(100+\frac{800.\,\text{caFunc}[t]}{2.\times 10^{-6}+\text{caFunc}[t]}\right)\text{fillStateSS}[t]}{\text{ss1}[t]},\right.$$
$$25\,572.7-15\,343.6\,\text{sitePlugging}[t],\ 0,\ 0,\ 0,\ 0,\ 0\},$$
$$\{5\left(5.11454\times 10^8\,\text{caFunc}[t]-4.60309\times 10^8\,\text{caFunc}[t]\times\text{sitePlugging}[t]\right),$$
$$-25\,572.7+15\,343.6\,\text{sitePlugging}[t]-$$
$$4\left(5.11454\times 10^8\,\text{caFunc}[t]-4.60309\times 10^8\,\text{caFunc}[t]\times\text{sitePlugging}[t]\right),$$
$$0.5\left(25\,572.7-15\,343.6\,\text{sitePlugging}[t]\right),\ 0,\ 0,\ 0,\ 0\},$$
$$\{0,\ 4\left(5.11454\times 10^8\,\text{caFunc}[t]-4.60309\times 10^8\,\text{caFunc}[t]\times\text{sitePlugging}[t]\right),$$
$$-0.5\left(25\,572.7-15\,343.6\,\text{sitePlugging}[t]\right)-$$
$$3\left(5.11454\times 10^8\,\text{caFunc}[t]-4.60309\times 10^8\,\text{caFunc}[t]\times\text{sitePlugging}[t]\right),$$
$$0.1875\left(25\,572.7-15\,343.6\,\text{sitePlugging}[t]\right),\ 0,\ 0,\ 0\},$$
$$\{0,\ 0,\ 3\left(5.11454\times 10^8\,\text{caFunc}[t]-4.60309\times 10^8\,\text{caFunc}[t]\times\text{sitePlugging}[t]\right),$$
$$-0.1875\left(25\,572.7-15\,343.6\,\text{sitePlugging}[t]\right)-$$
$$2\left(5.11454\times 10^8\,\text{caFunc}[t]-4.60309\times 10^8\,\text{caFunc}[t]\times\text{sitePlugging}[t]\right),$$
$$0.0625\left(25\,572.7-15\,343.6\,\text{sitePlugging}[t]\right),\ 0,\ 0\},$$
$$\{0,\ 0,\ 0,\ 2\left(5.11454\times 10^8\,\text{caFunc}[t]-4.60309\times 10^8\,\text{caFunc}[t]\times\text{sitePlugging}[t]\right),$$
$$-5.11454\times 10^8\,\text{caFunc}[t]-0.0625\left(25\,572.7-15\,343.6\,\text{sitePlugging}[t]\right)+$$
$$4.60309\times 10^8\,\text{caFunc}[t]\times\text{sitePlugging}[t],$$
$$0.0195313\left(25\,572.7-15\,343.6\,\text{sitePlugging}[t]\right),\ 0\},$$
$$\{0,\ 0,\ 0,\ 0,\ 5.11454\times 10^8\,\text{caFunc}[t]-4.60309\times 10^8\,\text{caFunc}[t]\times\text{sitePlugging}[t],$$
$$-17\,717.3-0.0195313\left(25\,572.7-15\,343.6\,\text{sitePlugging}[t]\right),\ 0\},$$
$$\{0,\ 0,\ 0,\ 0,\ 0,\ 17\,717.3,\ 0\}\}$$

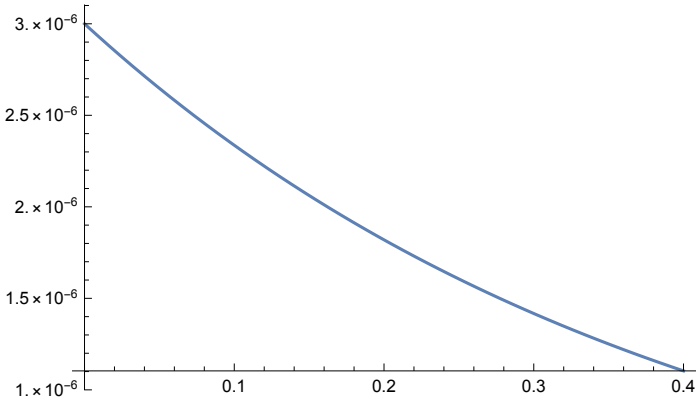
In[•]:= **eq**

Out[•]=
$$\Big\{\{ss1'[t], ss2'[t], ss3'[t], ss4'[t], ss5'[t], ss6'[t], ss7'[t]\} ==$$
$$\Big\{\Big(-181.545 - 5\left(5.11454\times10^{8}\,caFunc[t] - 4.60309\times10^{8}\,caFunc[t]\times sitePlugging[t]\right) + \frac{\left(100+\frac{800.\,caFunc[t]}{2.\times10^{-6}+caFunc[t]}\right) fillStateSS[t]}{ss1[t]}\Big)\,ss1[t] +$$
$$\left(25\,572.7 - 15\,343.6\,sitePlugging[t]\right) ss2[t],$$
$$5\left(5.11454\times10^{8}\,caFunc[t] - 4.60309\times10^{8}\,caFunc[t]\times sitePlugging[t]\right) ss1[t] +$$
$$\left(-25\,572.7 + 15\,343.6\,sitePlugging[t] - 4\left(5.11454\times10^{8}\,caFunc[t] - 4.60309\times10^{8}\,caFunc[t]\times sitePlugging[t]\right)\right) ss2[t] + 0.5\left(25\,572.7 - 15\,343.6\,sitePlugging[t]\right) ss3[t],$$
$$4\left(5.11454\times10^{8}\,caFunc[t] - 4.60309\times10^{8}\,caFunc[t]\times sitePlugging[t]\right) ss2[t] +$$
$$\left(-0.5\left(25\,572.7 - 15\,343.6\,sitePlugging[t]\right) - 3\left(5.11454\times10^{8}\,caFunc[t] - 4.60309\times10^{8}\,caFunc[t]\times sitePlugging[t]\right)\right) ss3[t] + 0.1875\left(25\,572.7 - 15\,343.6\,sitePlugging[t]\right) ss4[t],$$
$$3\left(5.11454\times10^{8}\,caFunc[t] - 4.60309\times10^{8}\,caFunc[t]\times sitePlugging[t]\right) ss3[t] +$$
$$\left(-0.1875\left(25\,572.7 - 15\,343.6\,sitePlugging[t]\right) - 2\left(5.11454\times10^{8}\,caFunc[t] - 4.60309\times10^{8}\,caFunc[t]\times sitePlugging[t]\right)\right) ss4[t] + 0.0625\left(25\,572.7 - 15\,343.6\,sitePlugging[t]\right) ss5[t],$$
$$2\left(5.11454\times10^{8}\,caFunc[t] - 4.60309\times10^{8}\,caFunc[t]\times sitePlugging[t]\right) ss4[t] +$$
$$\left(-5.11454\times10^{8}\,caFunc[t] - 0.0625\left(25\,572.7 - 15\,343.6\,sitePlugging[t]\right) + 4.60309\times10^{8}\,caFunc[t]\times sitePlugging[t]\right) ss5[t] + 0.0195313\left(25\,572.7 - 15\,343.6\,sitePlugging[t]\right) ss6[t],$$
$$\left(5.11454\times10^{8}\,caFunc[t] - 4.60309\times10^{8}\,caFunc[t]\times sitePlugging[t]\right) ss5[t] +$$
$$\left(-17\,717.3 - 0.0195313\left(25\,572.7 - 15\,343.6\,sitePlugging[t]\right)\right) ss6[t],$$
$$17\,717.3\,ss6[t]\Big\}, \{ss1[0], ss2[0], ss3[0], ss4[0], ss5[0], ss6[0], ss7[0]\} == \{1., 0., 0., 0., 0., 0., 0.\},$$
$$fillStateSS'[t] == 2.5 + \frac{60.\,caFunc[t]}{2.\times10^{-6}+caFunc[t]} - 8.61585\,fillStateSS[t] - \left(100+\frac{800.\,caFunc[t]}{2.\times10^{-6}+caFunc[t]}\right) fillStateSS[t] + 181.545\,ss1[t],$$
$$fillStateSS[0] == 1., sitePlugging'[t] == -0.04\,sitePlugging[t] + \left(1 - sitePlugging[t]\right) ss7'[t],$$
$$sitePlugging[0] == 0\Big\}$$

Solve all states

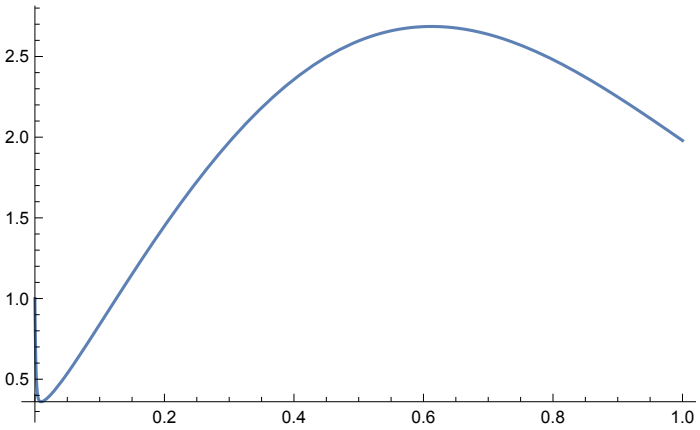
In[•]:=
```
caFunc[t_] := 3*^-6 * Exp[-t / tauOfDecayOfUncagedCa];;
Plot[caFunc[t], {t, 0, 0.4}, PlotRange → All]
```

Out[•]=

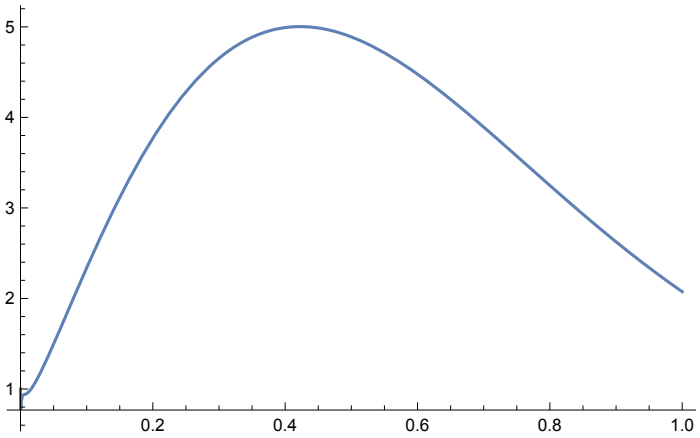


In[•]:=
```
myNDSolveResults = NDSolve[eq,
   {fillStateSS, ss1, ss2, ss3, ss4, ss5, ss6, ss7, sitePlugging}, {t, 0, 10.}];
tEndForPlot = 1.;
Plot[(fillStateSS[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
Plot[(ss1[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
Plot[(ss2[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
Plot[(ss3[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
Plot[(ss4[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
Plot[(ss5[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
Plot[(ss6[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
Plot[(ss7[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
Plot[(sitePlugging[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange → All]
```

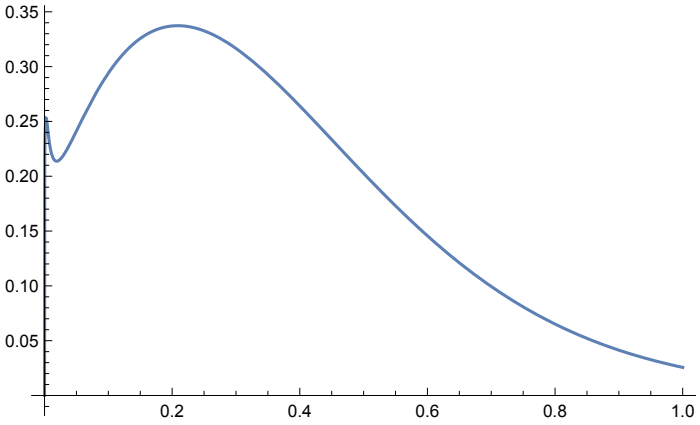
Out[•]=



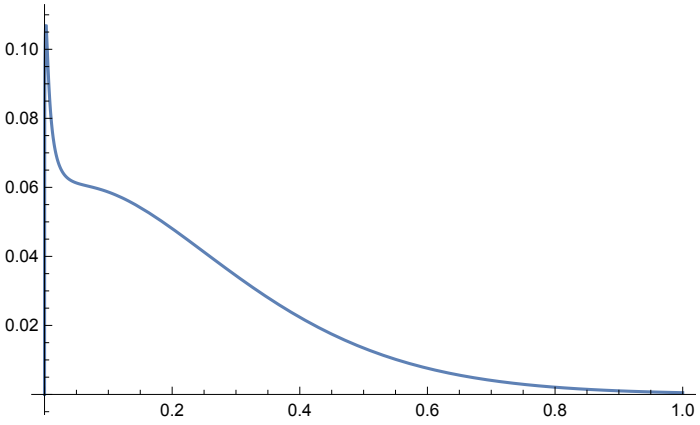
Out[•]=



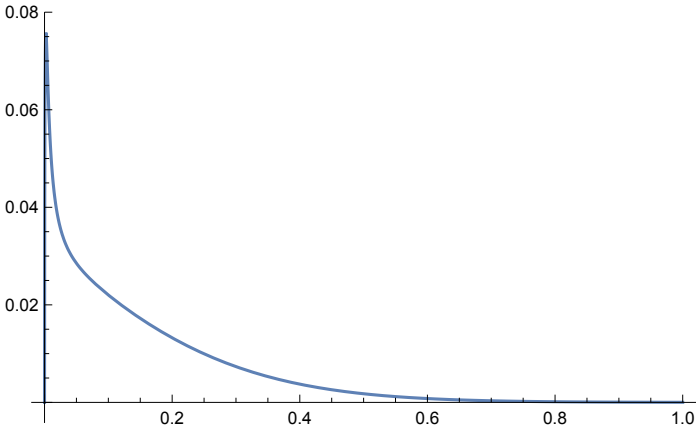
Out[•]=



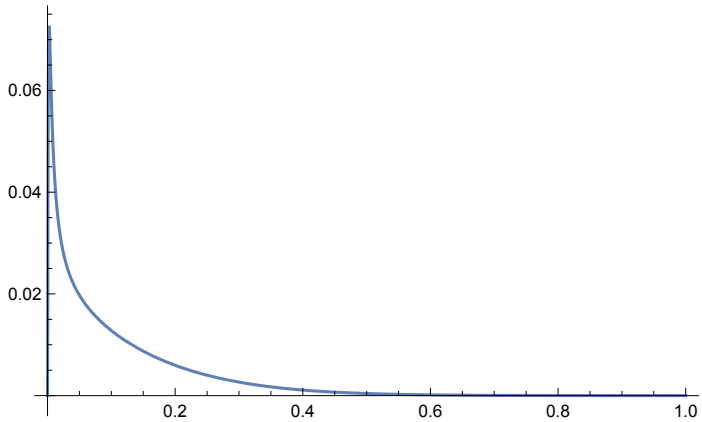
Out[•]=



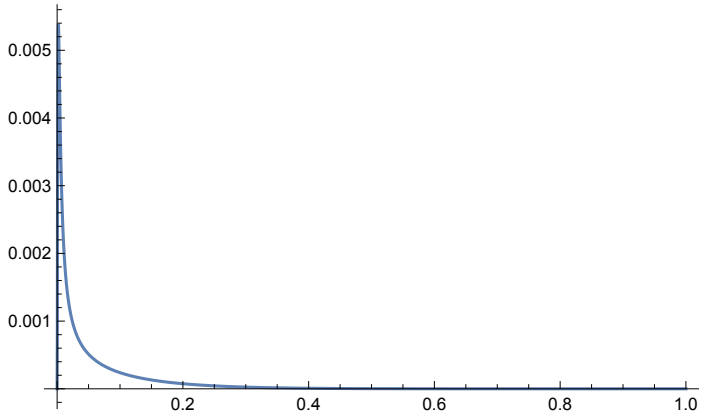
Out[•]=



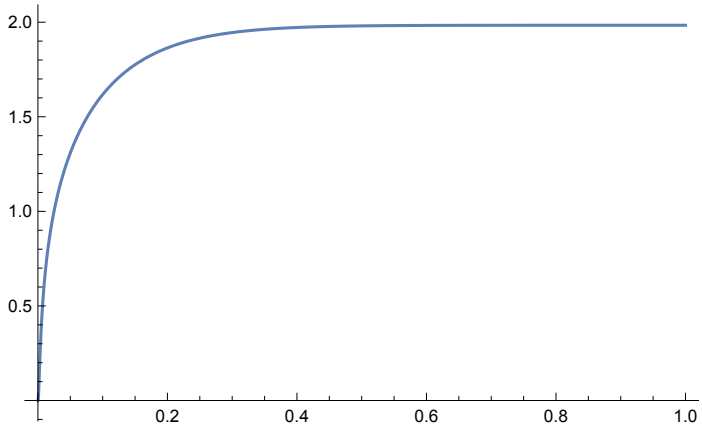
Out[•]=



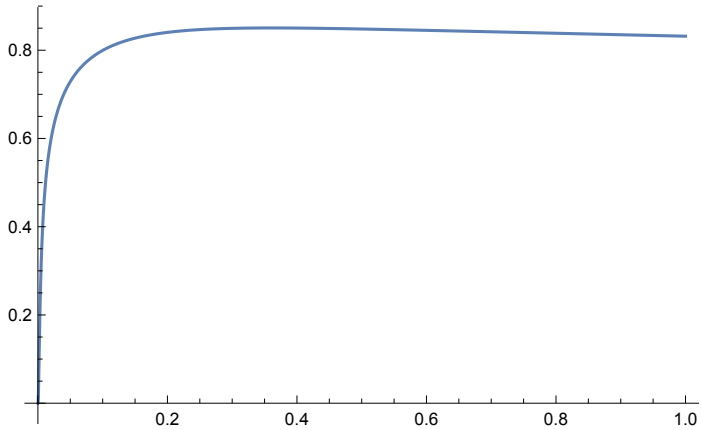
Out[•]=



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}];
        (*Lists for saving data within loop*)
        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}];
        (*Lists for saving data within loop*)
        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}];
        (*Lists for saving data within loop*)
        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, ss7, {t, 0, 0.4}];
(* plot results: *)
fused[t_] := ss7[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"];
];
```

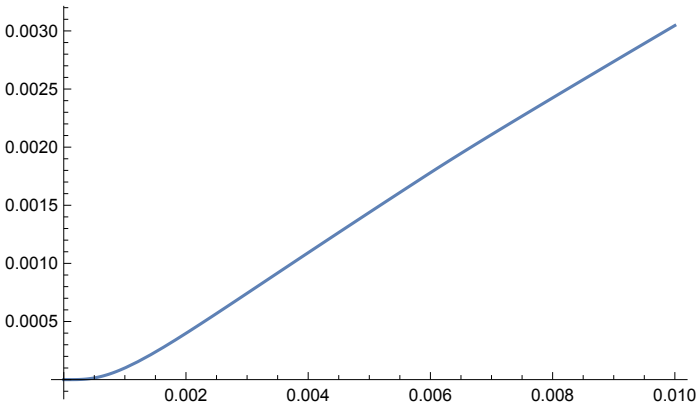
```
noiseN = noiseRepeats; (*fit parameter of last noisetrace*)
Print["Mono: chi2 = ", simParamNoiseD[[2, noiseN]],
 "     d = ", simParamNoiseD[[2, noiseN]], "     a = ",
 simParamNoiseD[[4, noiseN]], "     t = ", 1 / simParamNoiseD[[5, noiseN]]];
Print["Bi: ratio chi2Mono/chi2Bi = ", simParamNoiseD[[6, 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]];
```

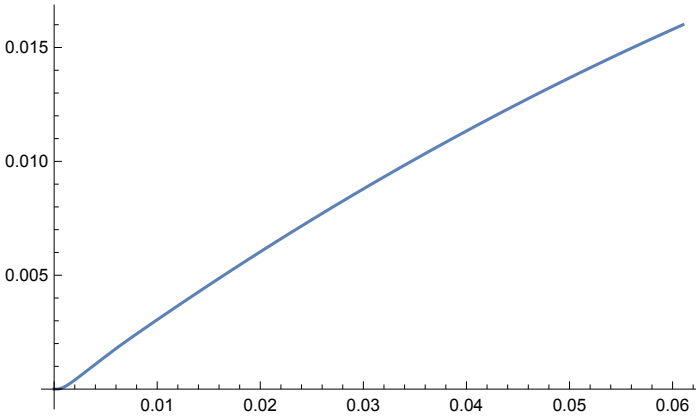
```
 simParamNoiseD[[8, noiseN]] = ((amp /. fitResultLongBi))[[1]];
 simParamNoiseD[[9, noiseN]] = ((amp amp1 /. fitResultLongBi))[[1]];
 (*relative amp1*)
 simParamNoiseD[[10, noiseN]] = (1 / (tau1 /. fitResultLongBi))[[1]];
 simParamNoiseD[[11, noiseN]] = (1 / (tau2 /. fitResultLongBi))[[1]];
 (*merge*)
 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 ----------





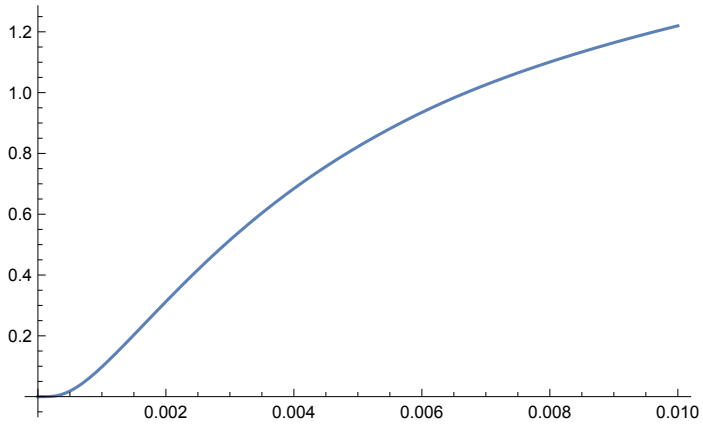
---------- C5

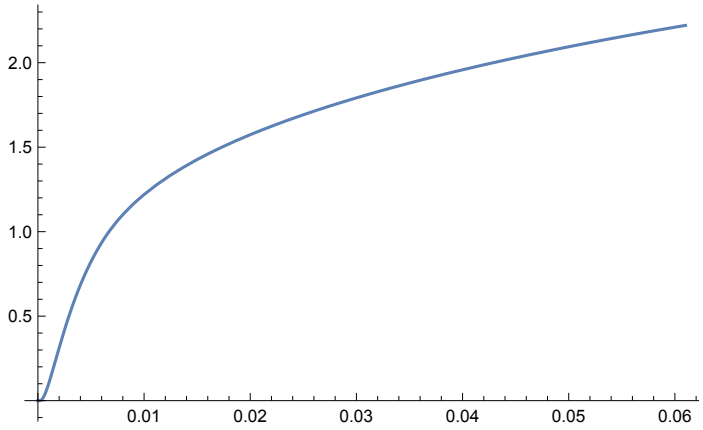
---------- C10

---------- D

---------- Ca = 4.79194

uM ----------





---------- C5

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

take mono

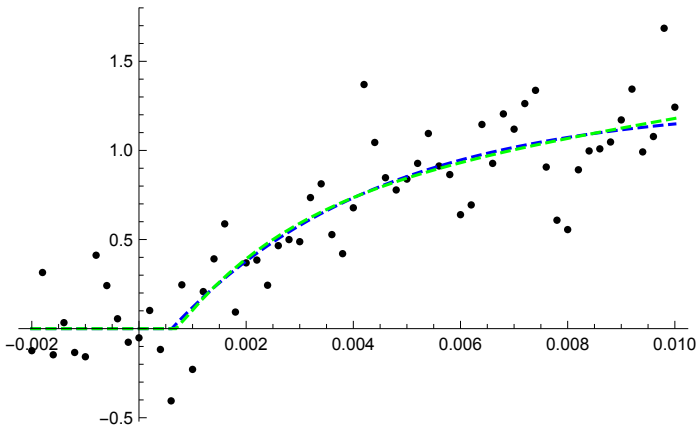
NonlinearModelFit: Failed to converge to the requested accuracy or precision within 100 iterations.

take mono

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



Mono: chi2 = 2.96481 d = 2.96481 a = 1.26086 t = 0.0038824

Bi: ratio chi2Mono/chi2Bi = 1.00554 d = 0.000716352 a = 21.5738 a1 = 0.736358 t1 =0.00217229 t2 = 0.421949

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

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

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

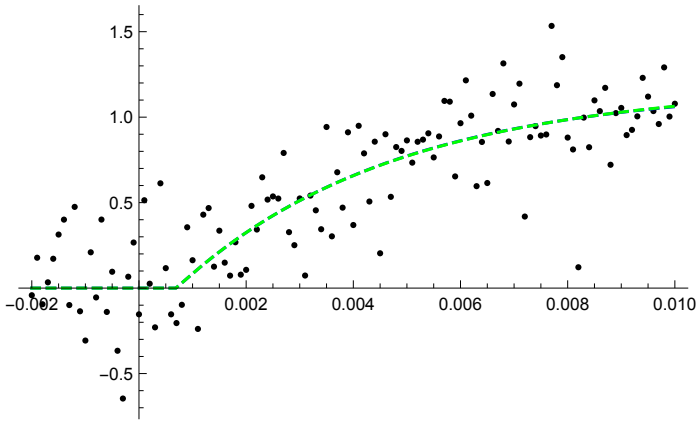
General: Further output of Quantile::empt will be suppressed during this calculation.

---------- C10

take mono

take mono

take mono



Mono: chi2 = 7.08688 d = 7.08688 a = 1.17872 t = 0.00403296

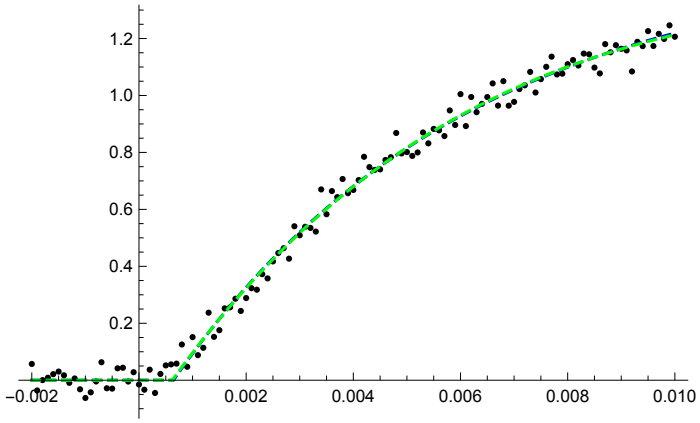
Bi: ratio chi2Mono/chi2Bi = 0.999794 d = 0.0007 a =
1.44509 a1 = 1.12956 t1 =0.00391193 t2 = 0.070724

---------- D

take mono

take mono

take mono

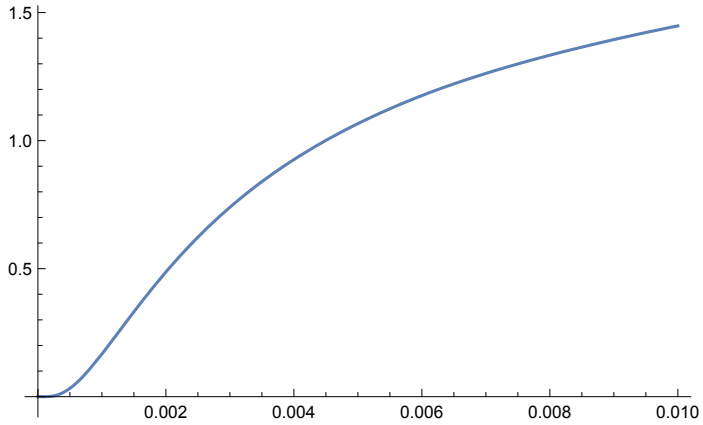


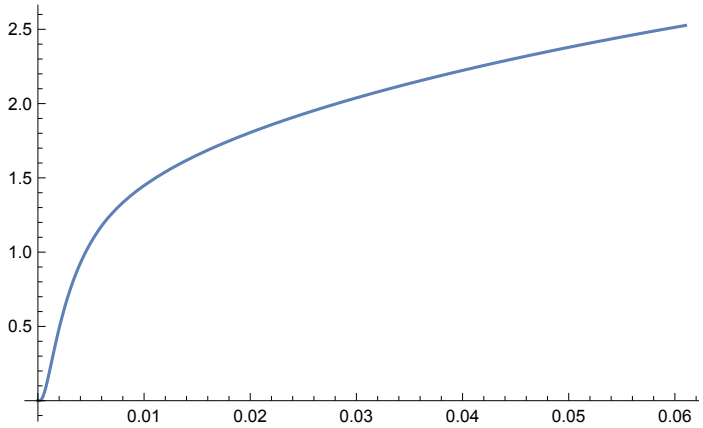
Mono: chi2 = 0.158654 d = 0.158654 a = 1.49144 t = 0.00550646

Bi: ratio chi2Mono/chi2Bi = 1.01159 d = 0.000623127 a =
0.86273 a1 = 2.87366 t1 =0.00768285 t2 = 0.0181125

---------- Ca = 5.70182

uM ----------



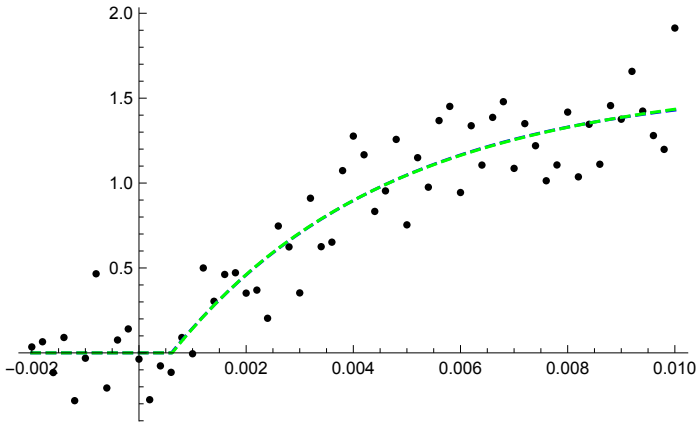


---------- C5

take mono

take mono

take mono



Mono: chi2 = 2.45802 d = 2.45802 a = 1.58885 t = 0.00407806

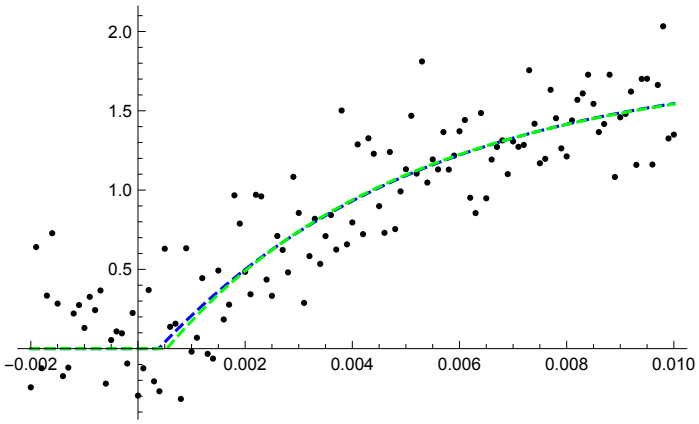
Bi: ratio chi2Mono/chi2Bi = 1.00011 d = 0.000620321 a =
5.15402 a1 = 1.44204 t1 =0.00374182 t2 = 0.311006

---------- C10

take mono

take mono

take mono



Mono: chi2 = 8.51897 d = 8.51897 a = 1.80949 t = 0.00499477

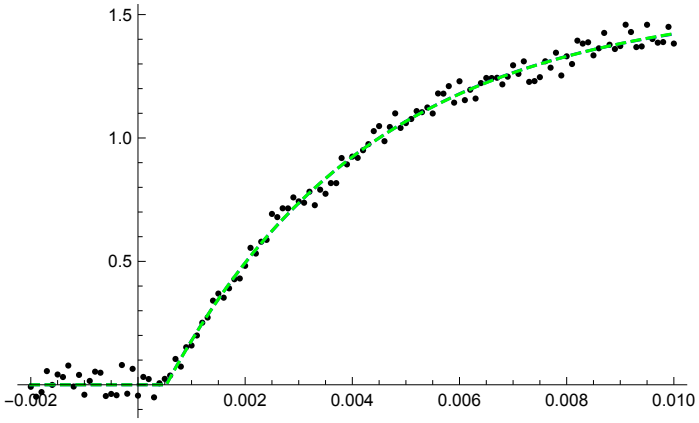
Bi: ratio chi2Mono/chi2Bi = 0.996193 d = 0.000540053 a =
1.82242 a1 = 0.244523 t1 =0.00218104 t2 = 0.00543857

---------- D

take mono

take mono

take mono

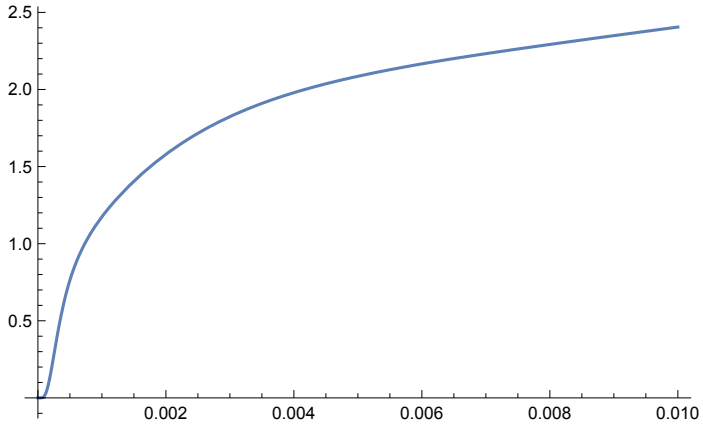


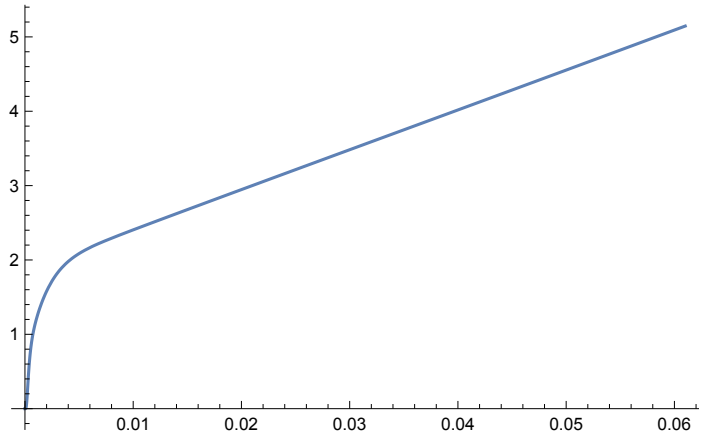
Mono: chi2 = 0.150768 d = 0.150768 a = 1.55454 t = 0.00385334

Bi: ratio chi2Mono/chi2Bi = 1.00016 d = 0.000529527
a = 2.15068 a1 = 1.5176 t1 =0.0037703 t2 = 0.207261

---------- Ca = 26.2105

uM ----------



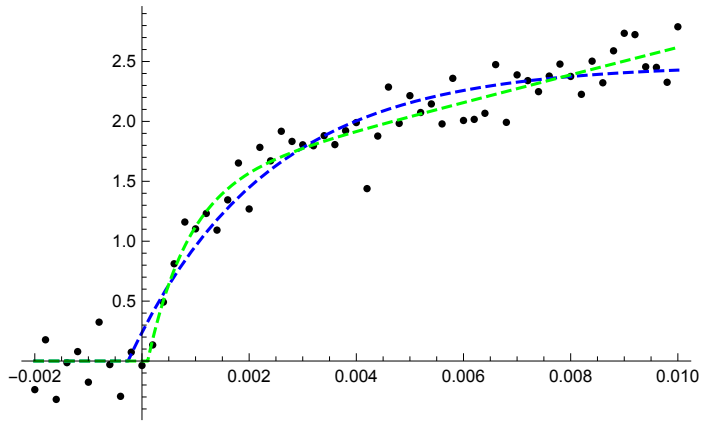


---------- C5

take bi

take bi

take bi



Mono: chi2 = 2.34076 d = 2.34076 a = 2.47322 t = 0.00256242

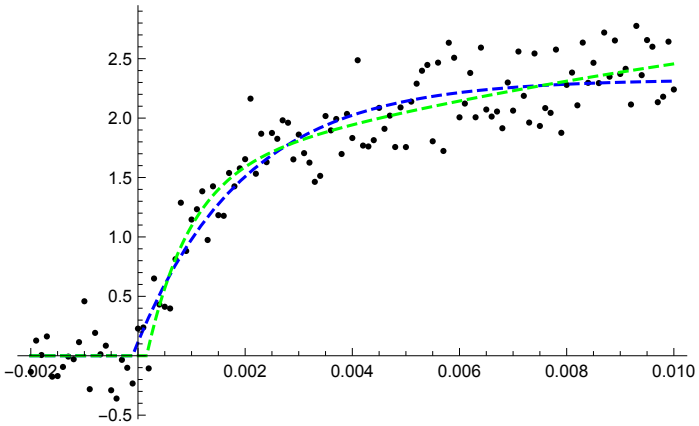
Bi: ratio chi2Mono/chi2Bi = 1.35958 d = 0.000107103 a = 25.382 a1 = 1.46219 t1 =0.000755749 t2 = 0.199792

---------- C10

take bi

take bi

take bi



Mono: chi2 = 6.85199 d = 6.85199 a = 2.32637 t = 0.00200985

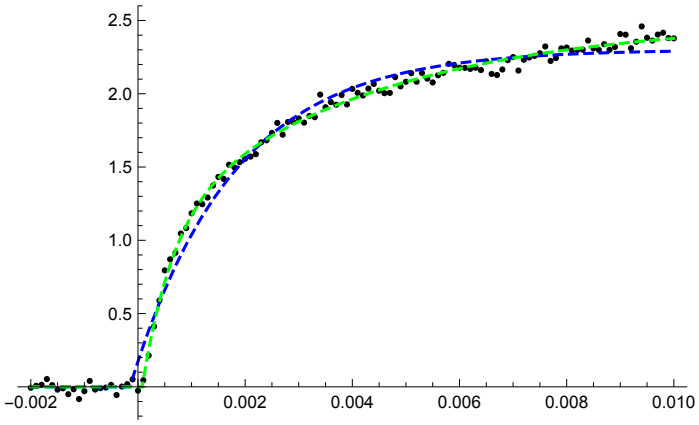
Bi: ratio chi2Mono/chi2Bi = 1.09199 d = 0.000162192 a = 3.47881 a1 = 1.51173 t1 =0.00079419 t2 = 0.0150339

---------- D

take bi

take bi

take bi

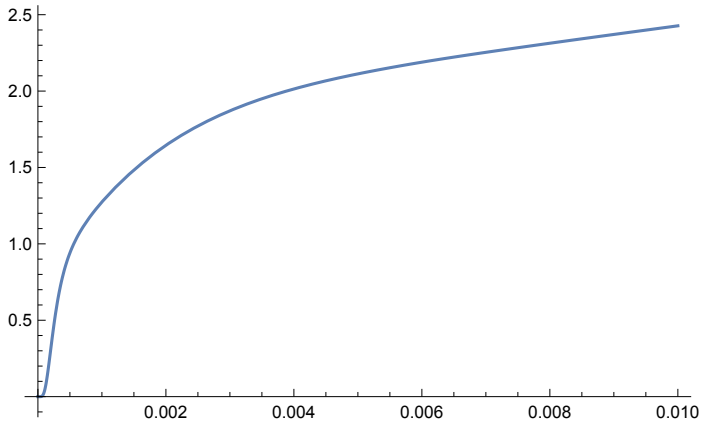


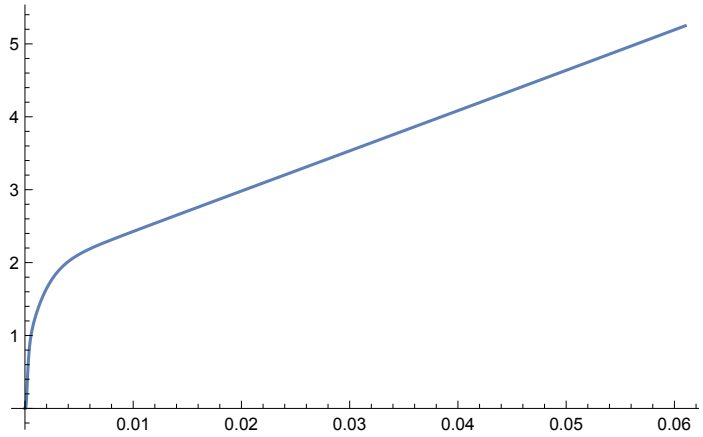
Mono: chi2 = 0.637257 d = 0.637257 a = 2.30186 t = 0.00191212

Bi: ratio chi2Mono/chi2Bi = 3.72297 d = 0.0000797056 a = 2.50343 a1 = 1.10874 t1 =0.000567663 t2 = 0.0041203

---------- Ca = 30.4385

uM ----------



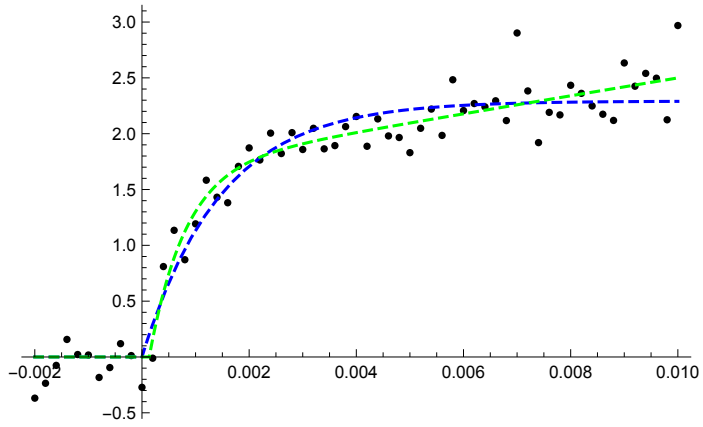


---------- C5

take bi

take bi

take bi



Mono: chi2 = 2.81571 d = 2.81571 a = 2.29197 t = 0.00146332

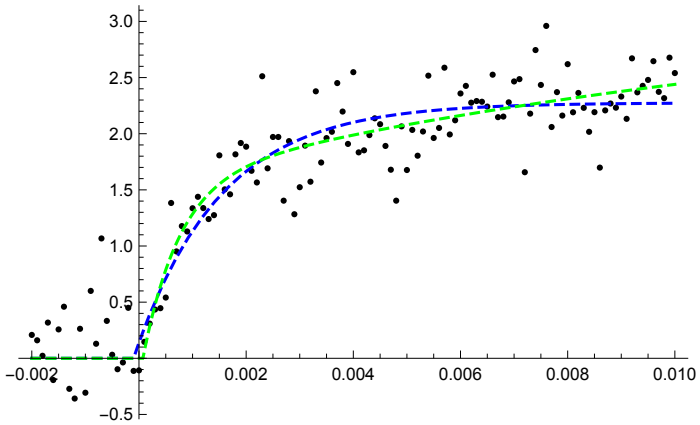
Bi: ratio chi2Mono/chi2Bi = 1.2612 d = 0.000135555 a = 19.4201 a1 = 1.69521 t1 =0.000668213 t2 = 0.212436

---------- C10

take mono

take bi

take bi



Mono: chi2 = 10.4952 d = 10.4952 a = 2.27521 t = 0.0015946

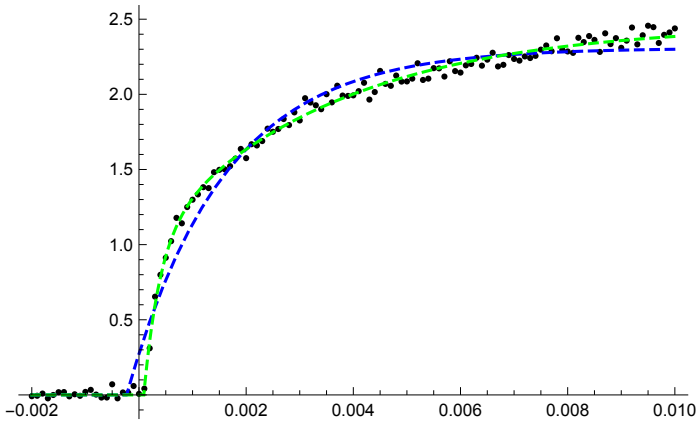
Bi: ratio chi2Mono/chi2Bi = 1.10011 d = 0.0000738803 a = 3.31022 a1 = 1.58329 t1 =0.000679344 t2 = 0.0145024

---------- D

take bi

take bi

take bi

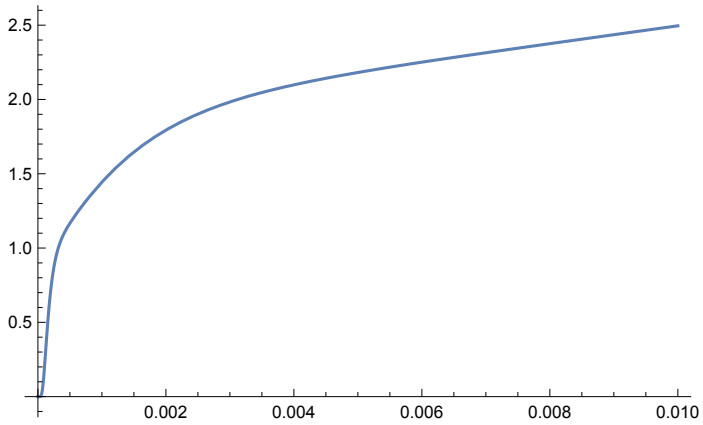


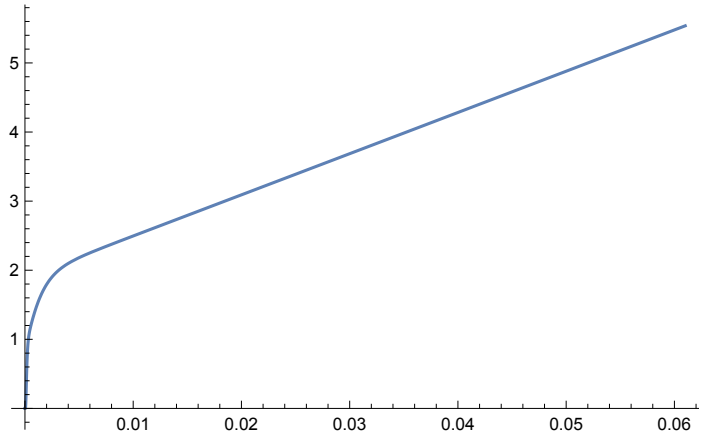
Mono: chi2 = 0.921959 d = 0.921959 a = 2.30794 t = 0.0018089

Bi: ratio chi2Mono/chi2Bi = 4.84035 d = 0.0000900679 a = 2.46702 a1 = 1.02232 t1 =0.000300933 t2 = 0.00345117

---------- Ca = 74.5861

uM ----------





---------- C5

General: Exp[−759.786] is too small to represent as a normalized machine number; precision may be lost.

General: Exp[−828.735] is too small to represent as a normalized machine number; precision may be lost.

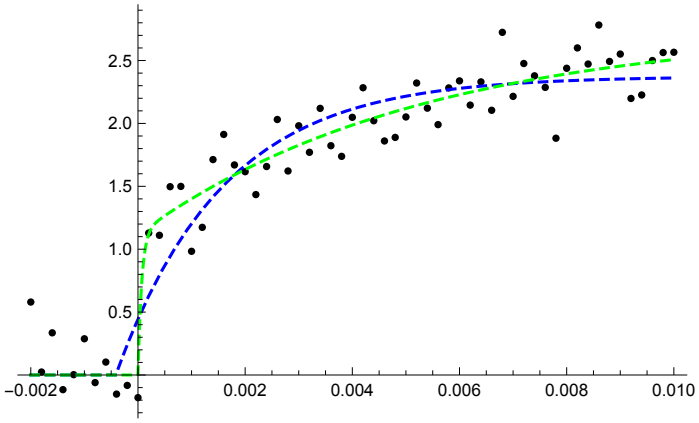
General: Exp[−897.685] is too small to represent as a normalized machine number; precision may be lost.

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

take mono

take bi

take bi



Mono: chi2 = 4.02317 d = 4.02317 a = 2.37414 t = 0.00199601

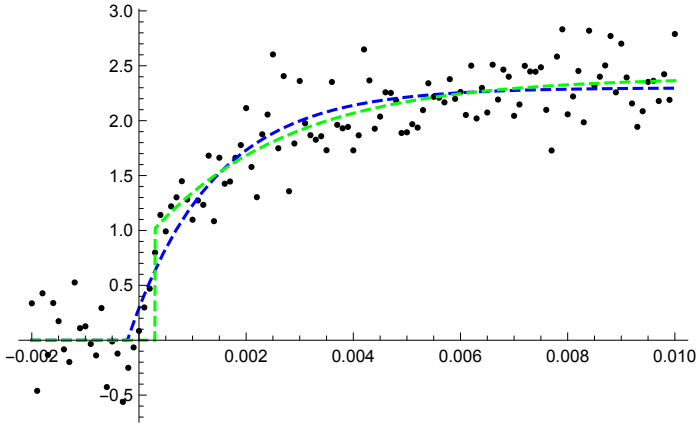
Bi: ratio chi2Mono/chi2Bi = 1.58786 d = 3.10281×10^{-17} a = 2.75379 a1 = 1.11905 t1 =0.0000693949 t2 = 0.00528651

---------- C10

take bi

take mono

take bi



Mono: chi2 = 8.53216 d = 8.53216 a = 2.2995 t = 0.00158135

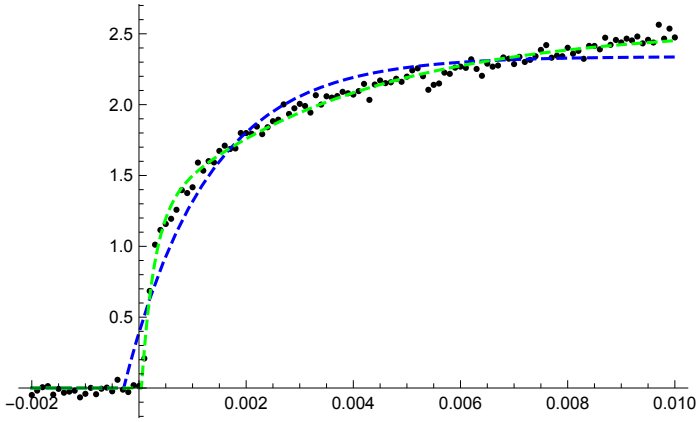
Bi: ratio chi2Mono/chi2Bi = 1.04938 d = 0.000298449 a = 2.39656 a1 = 1.01686 t1 =1.0115×10^{-6} t2 = 0.00258027

---------- D

take bi

take bi

take bi



Mono: chi2 = 1.41882 d = 1.41882 a = 2.33928 t = 0.00154846

Bi: ratio chi2Mono/chi2Bi = 5.7546 d = 0.0000457998 a = 2.54439 a1 = 1.22293 t1 =0.000235629 t2 = 0.00375127

Plots

In[•]:= **caFact = 1*^6;**

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

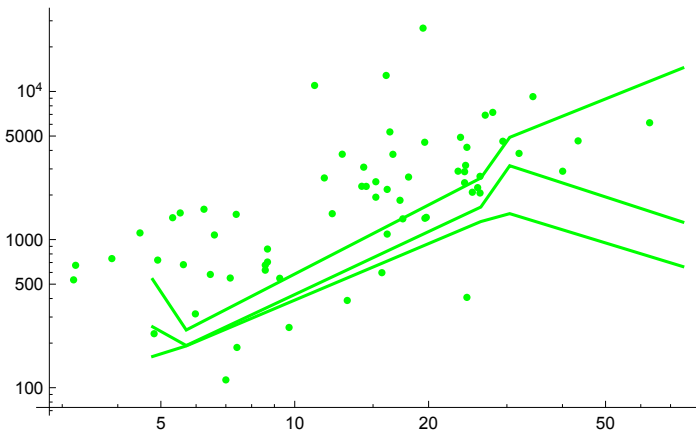
release rate 1/tau1 (merge of mono 1/tau and bi 1/tau1)

```
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"];
         ];
```

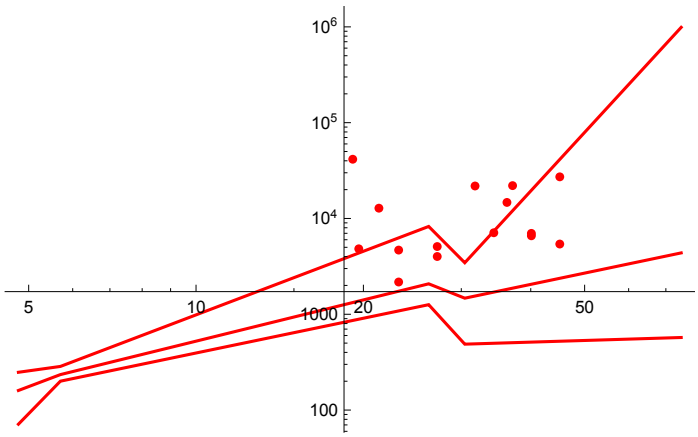
Out[•]=



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"];
 ];
```

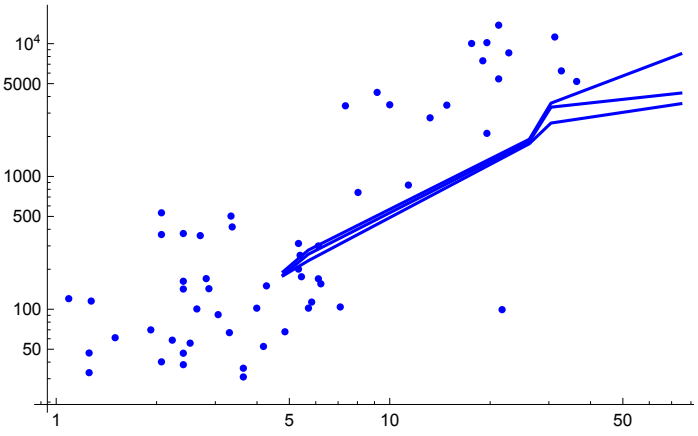
Out[•]=



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"];
 ];
```

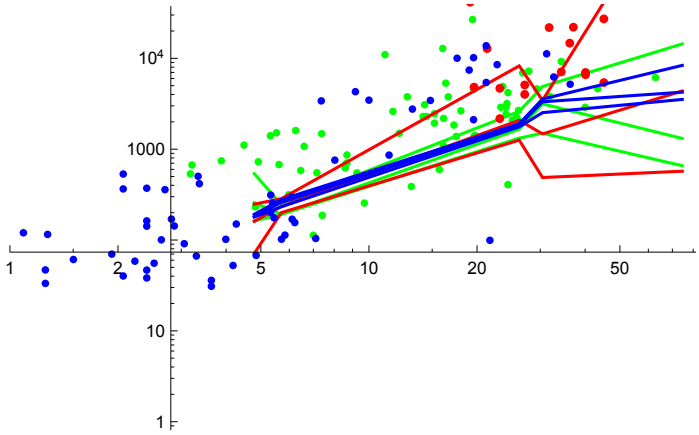
Out[•]=



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
```

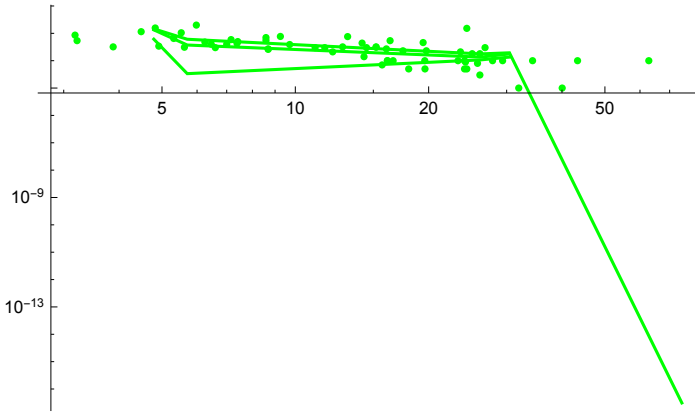
Out[•]//TableForm=

0.703073	
4.79194	0.00134142
5.70182	0.000373913
26.2105	0.000135174
30.4385	0.000176799
74.5861	−0.000147985

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"];
 ];
```

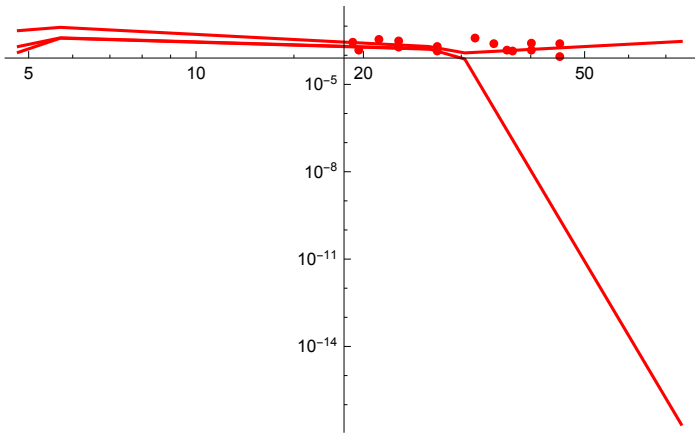
Out[•]=



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"];
 ];
```

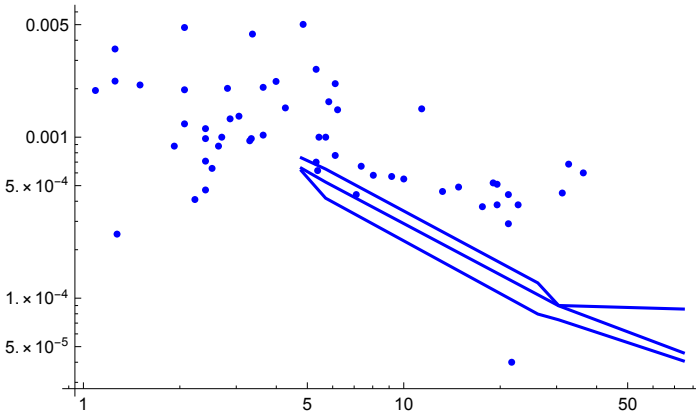
Out[•]=



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"];
 ];
```

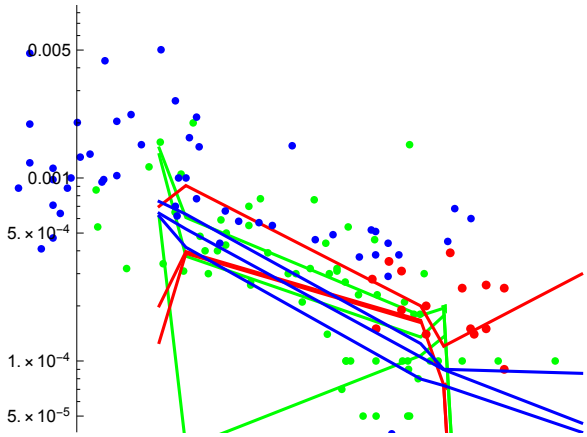
Out[•]=



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];
```

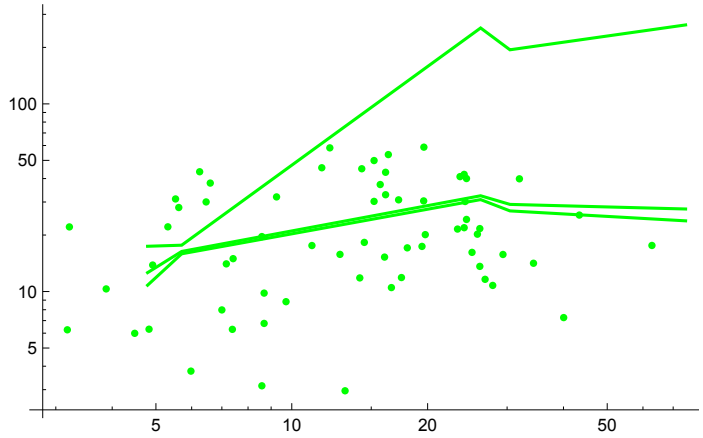
amp (merge of mono amp and bi amp)

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 ]
```

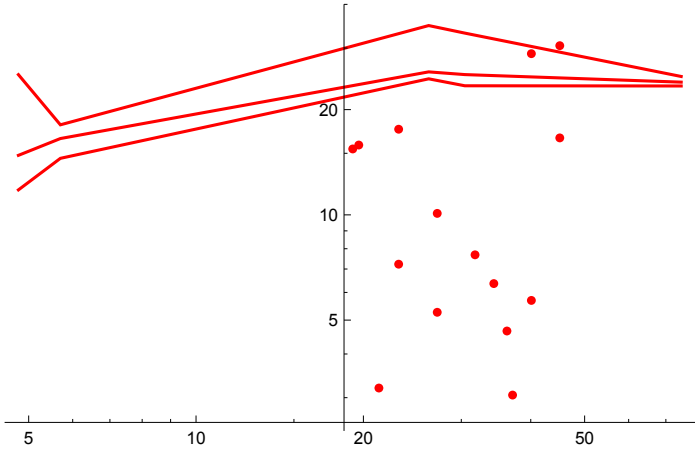
Out[•]=



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 ]
```

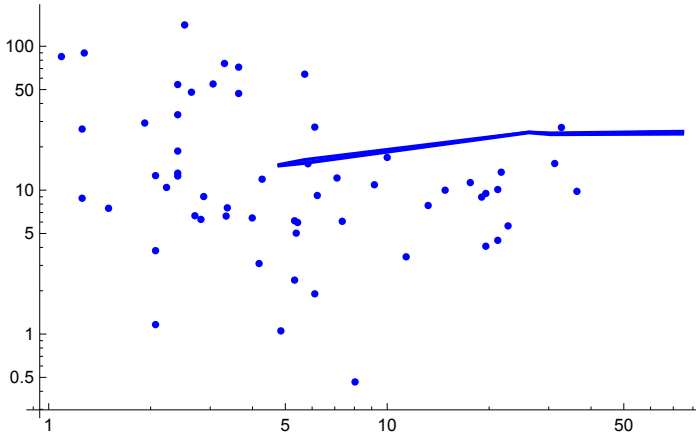
Out[•]=



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 ]
```

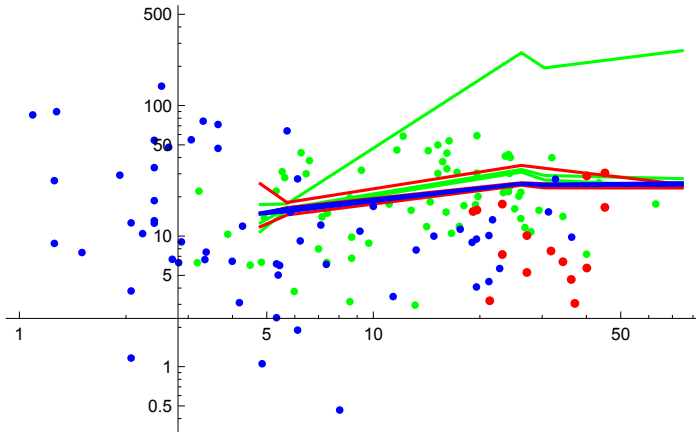
Out[•]=



C5 and C10 and D

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

Out[•]=



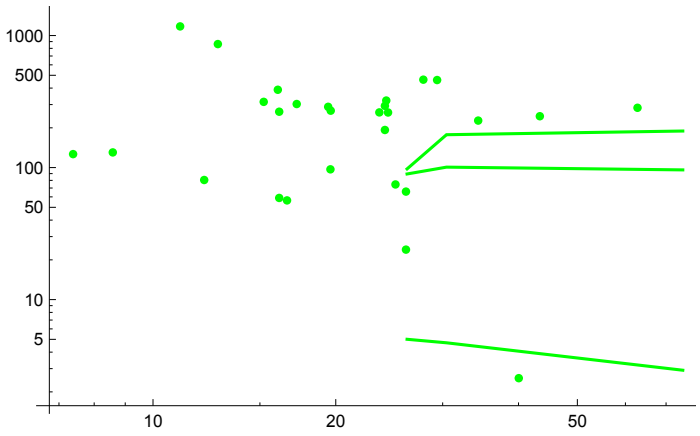
release rate 1/tau2 of bi fits (if bi is justified)

```
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"];
 ];
```

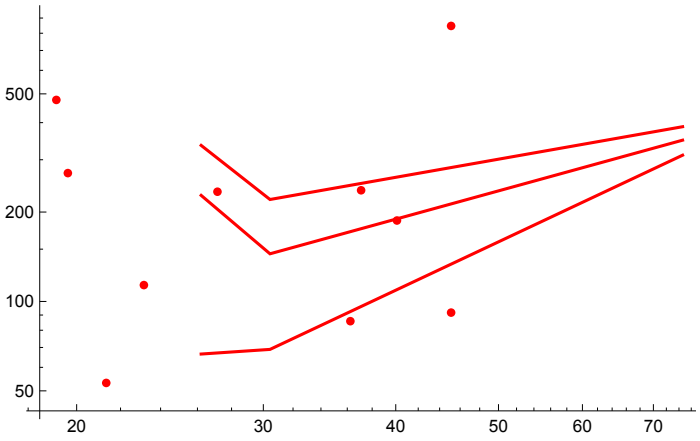
Out[•]=



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"];
 ];
```

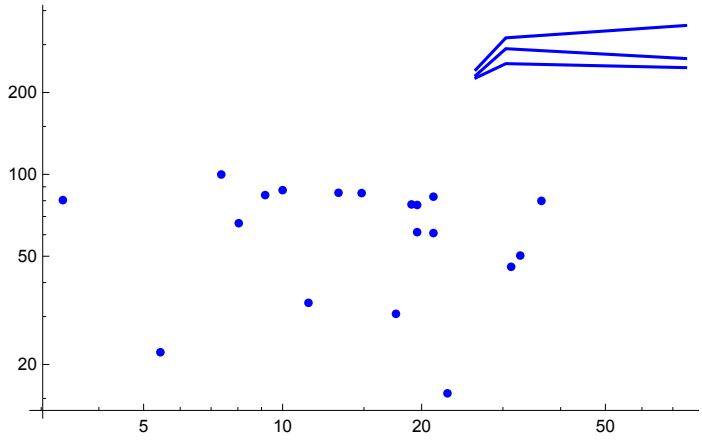
Out[•]=



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"];
 ];
```

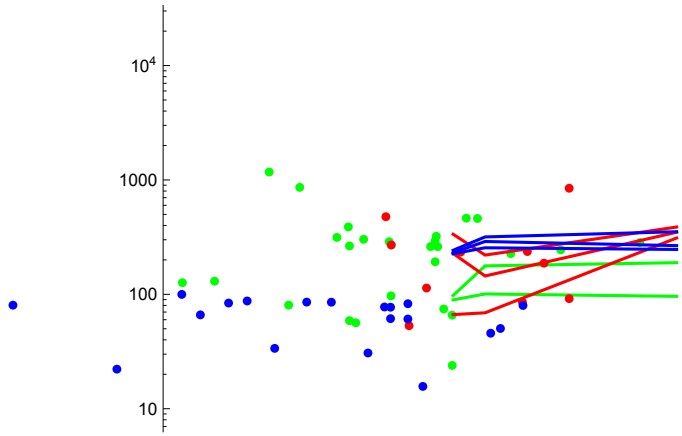
Out[•]=



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];
```

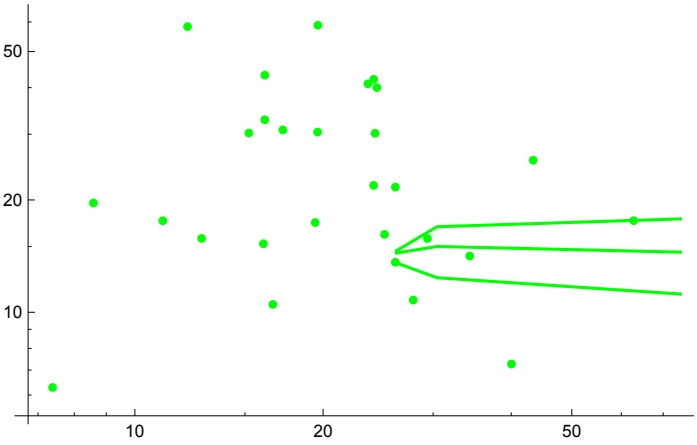
amp1 of bi fits (if bi is justified)

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 ]
```

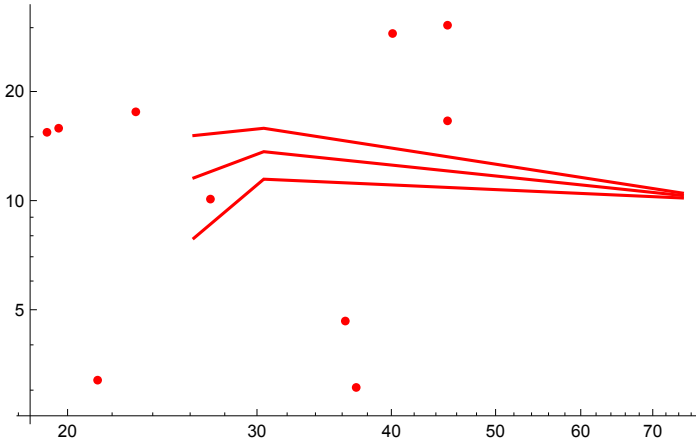
Out[•]=



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 ]
```

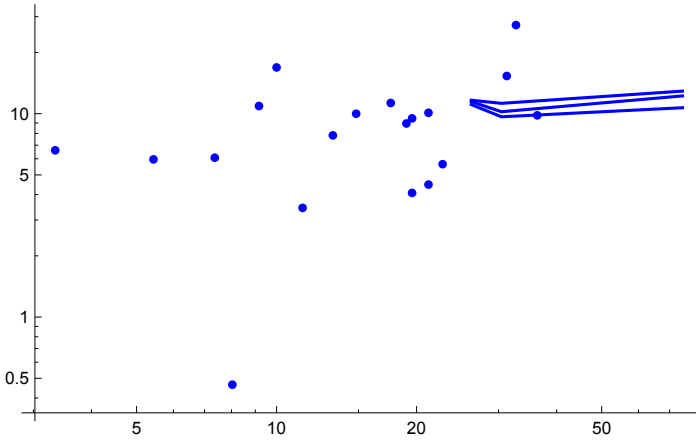
Out[•]=



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 ]
```

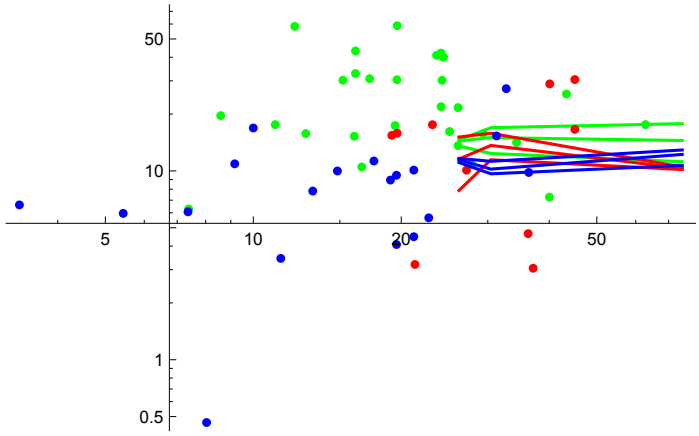
Out[•]=



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[•]=



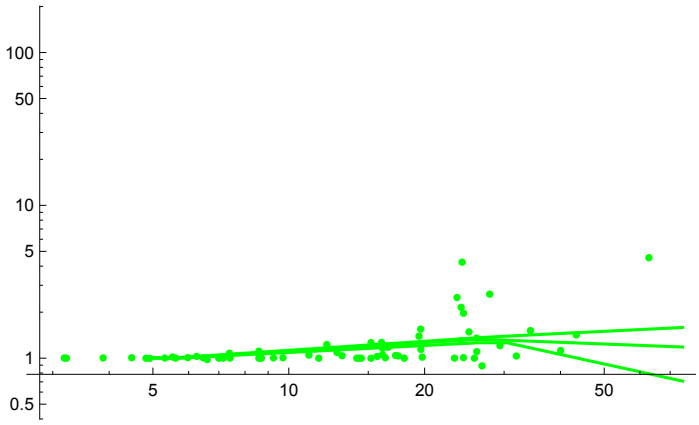
chi2 mono/bi ratio

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"];
 ];
```

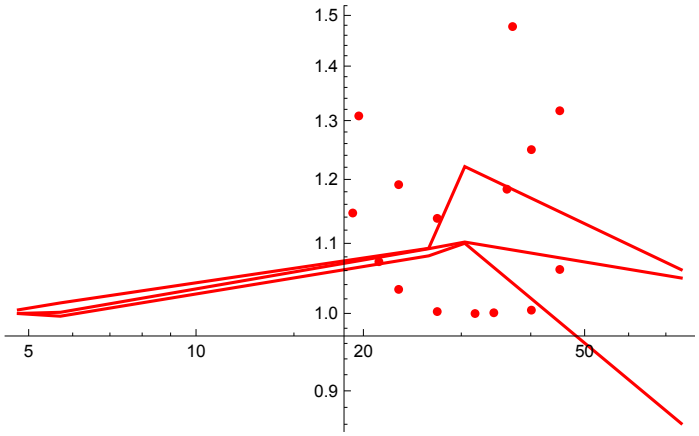
Out[•]=



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"];
 ];
```

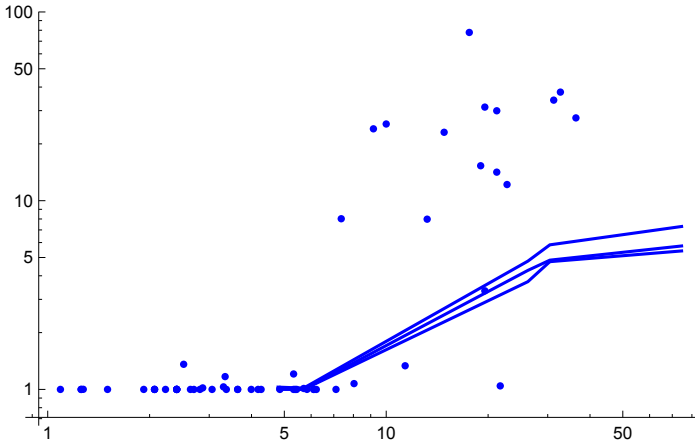
Out[•]=



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"];
 ];
```

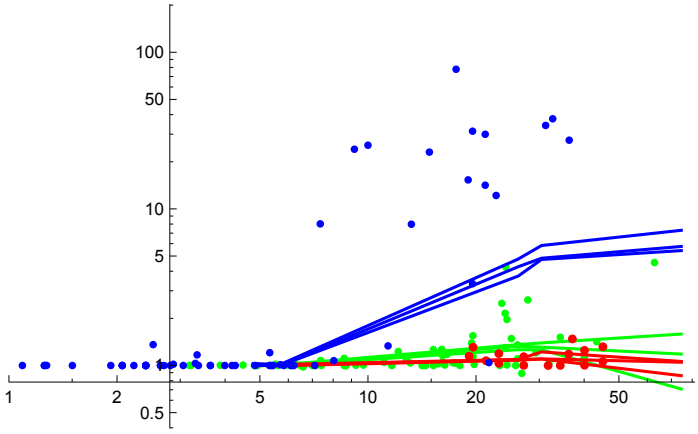
Out[•]=



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];
```

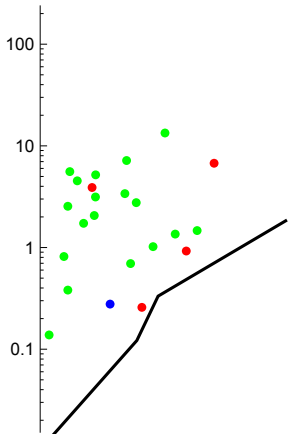
Out[•]=



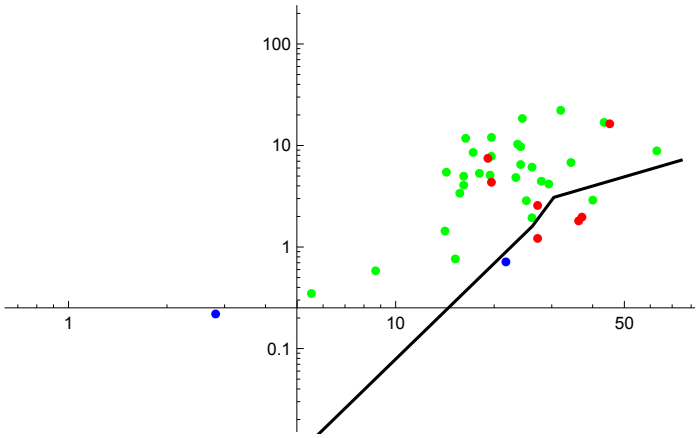
Nv

```
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;
        ];
```

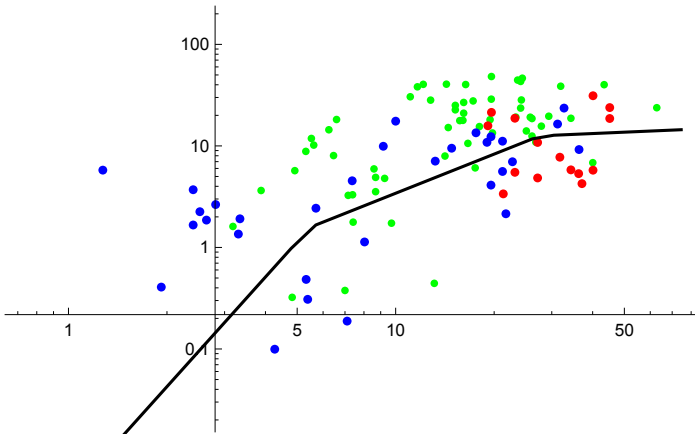
```
time for Nv (ms) = 0.1
```



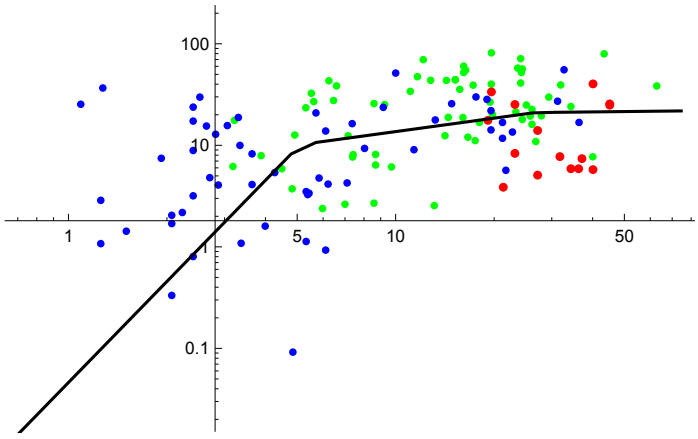

```
time for Nv (ms) = 0.2
```



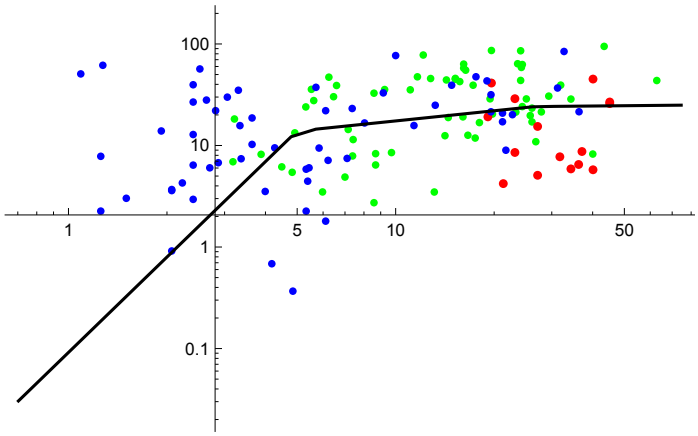
time for Nv (ms) = 1.



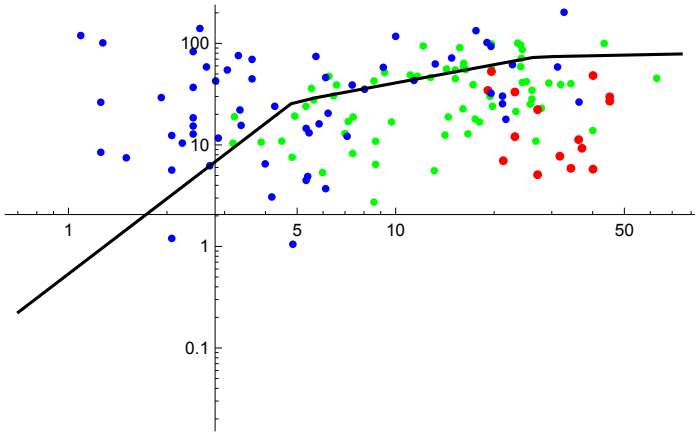
time for Nv (ms) = 5.



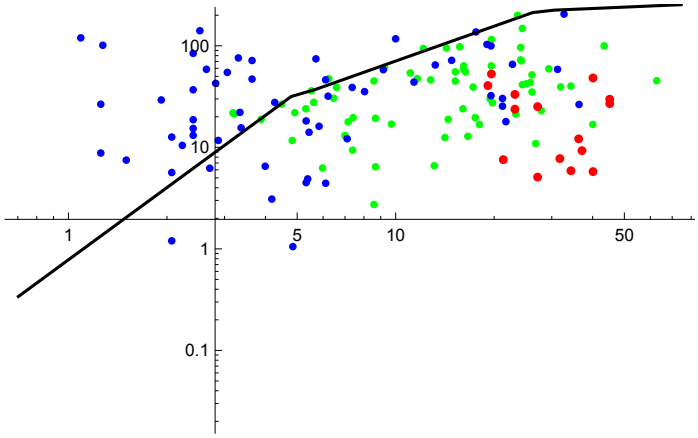
time for Nv (ms) = 10.




```
time for Nv (ms) = 100.
```

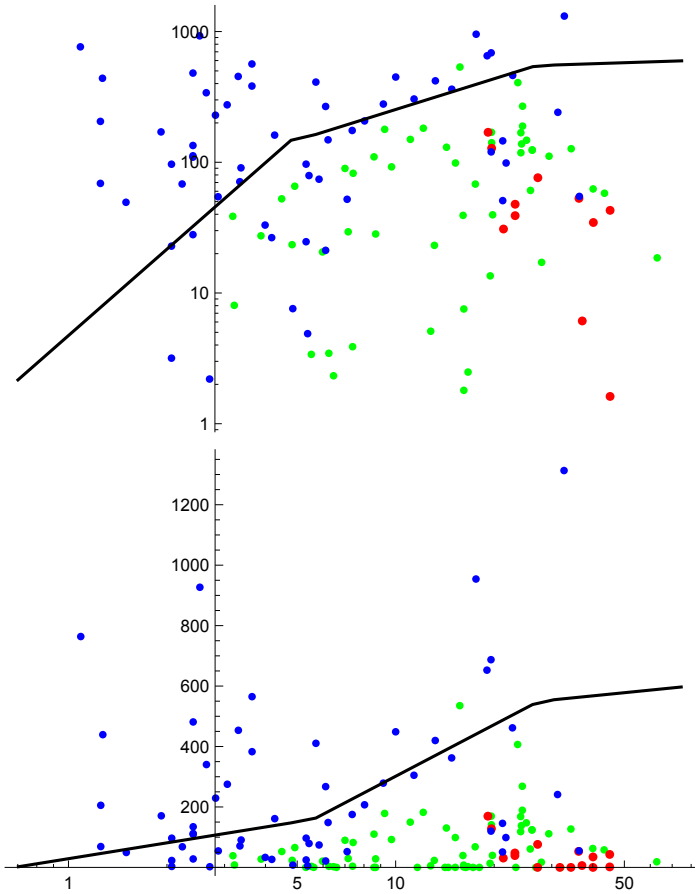



```
time for Nv (ms) = 400.
```



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;
```



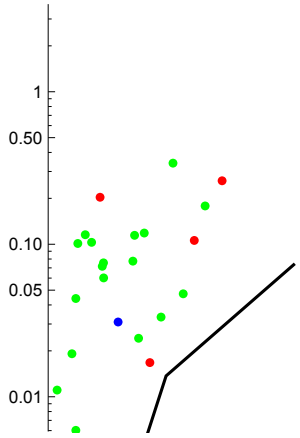
```
In[•]:= If[exportYes == 1,
          Export["plot sustained release Cm5 data.txt", ttt1, "Table"];
          Export["plot sustained release Cm10 data.txt", ttt2, "Table"];
          Export["plot sustained release D data.txt", ttt3, "Table"];
          Export["plot sustained release sim.txt", ttt4,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

Nv normalize to the value at 5 ms

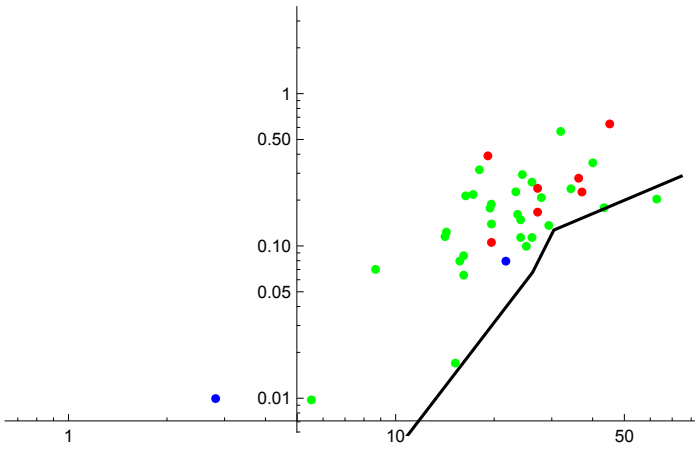
```
In[•]:= timeOfNv
        nromPos = 5;
        1000 * timeOfNv[[nromPos]]
Out[•]= {0.0001, 0.0002, 0.001, 0.005, 0.01, 0.1, 0.4}

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=
1.15831 × 10^-8    0.0000225289    0.000047581    0.0121758    0.0332647    0.183084
4.04537 × 10^-7    0.000699228     0.00135246     0.161133     0.30843      0.716014
0.000100151        0.0977688       0.166414       1.17424      1.27397      1.44388
0.00143825         0.822265        1.06648        2.08574      2.11265      2.18153
0.00304654         1.21973         1.4482         2.40481      2.42716      2.49566
0.022522           2.54577         2.91692        7.25498      7.41929      7.86737
0.0339701          3.16342         3.72736        21.2295      22.4337      25.4124

Out[•]//TableForm=
3.80206 × 10^-6    0.0000184704    0.0000328553    0.00506312    0.0137052    0.07336
0.000132786        0.000573263     0.000933887     0.0670046     0.127074     0.28690
0.0328736          0.0801559       0.114911        0.488288      0.524878     0.57855
0.472094           0.674135        0.736418        0.86732       0.870421     0.87412
1.                 1.              1.              1.            1.           1.
7.39265            2.08715         2.01417         3.01686       3.05677      3.15242
11.1504            2.59354         2.57379         8.82791       9.24274      10.1826
```

```
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;
 ];
```

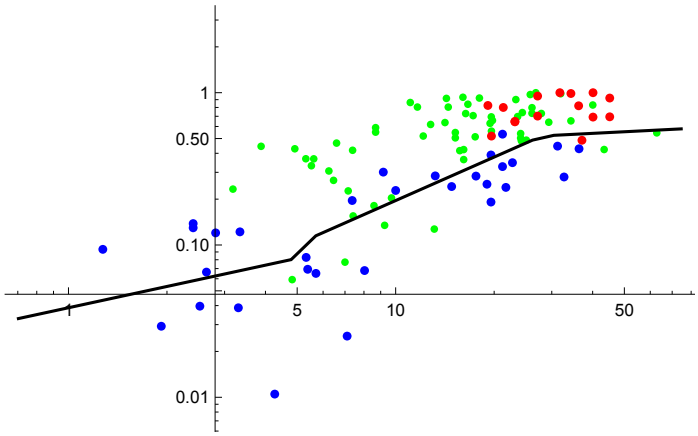
```
time for Nv (ms) = 0.1
```



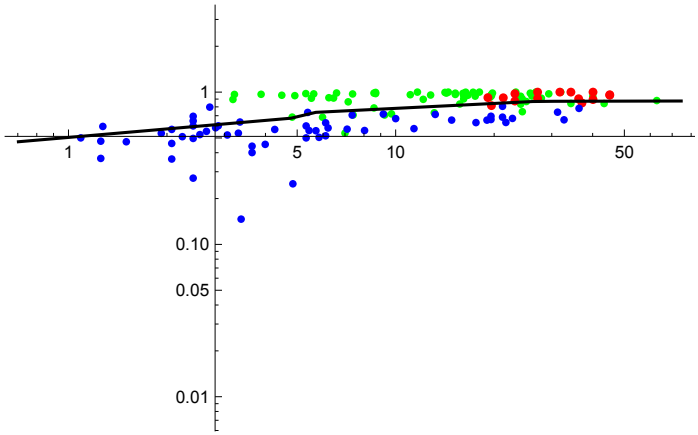

```
time for Nv (ms) = 0.2
```



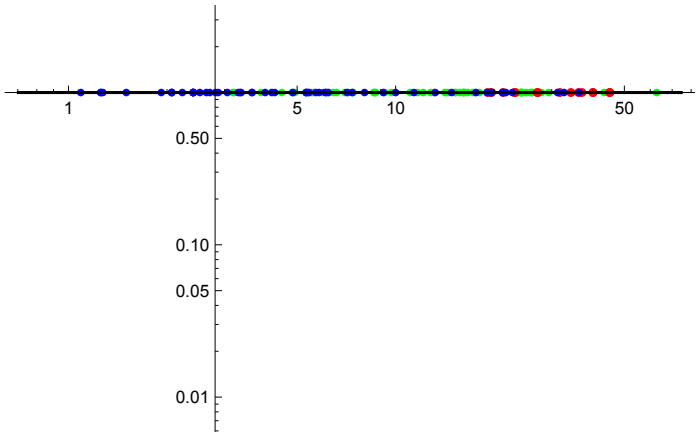

```
time for Nv (ms) = 1.
```



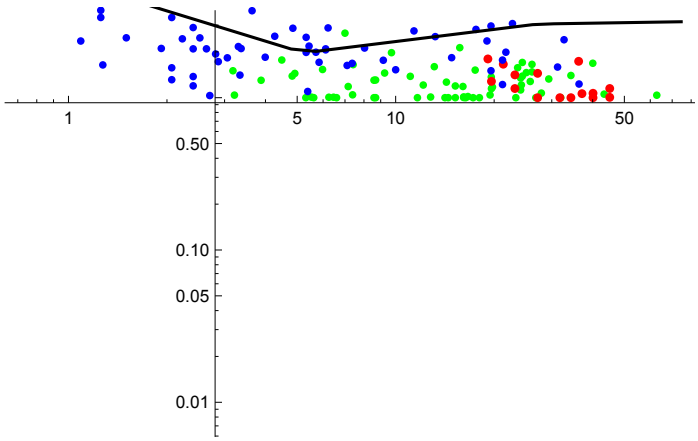
time for Nv (ms) = 5.



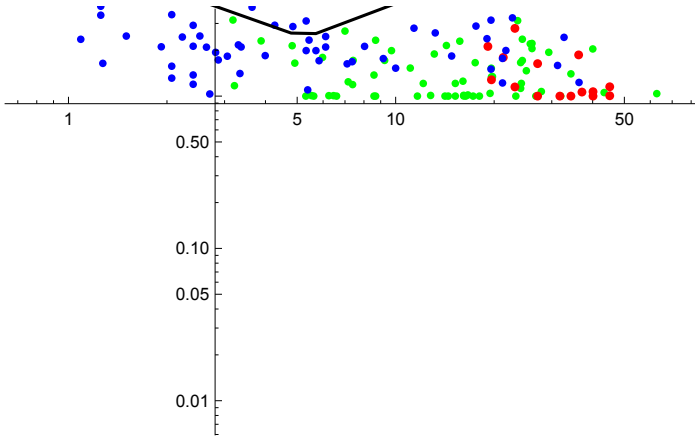
time for Nv (ms) = 10.



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	2.56849	0.00134142	1.26086	257.573	1.0015	0.00133815
0	2.50544	0.000373913	1.63549	192.607	1.00137	0.000524317
0	2.69357	−0.0002	2.33714	538.914	1.32846	0.000135174
0	3.61998	−0.0000932546	2.26729	683.377	1.30954	0.000176799
0	3.41927	−0.000336824	2.37414	650.483	1.183	3.10281×10^{-17}

Out[•]//TableForm=

0	2.51745	0.000612205	1.08074	162.488	1.0013	0.000716352	0
0	2.45802	0.0000335951	1.58885	191.444	1.00011	0.000334901	2
0	2.34076	−0.000259939	2.25599	390.257	1.26115	0.000107103	3
0	2.81571	−0.0002	2.21336	641.229	1.2612	0.000135555	2
0	3.22158	−0.000412631	2.31681	500.999	0.707839	−0.000147985	2

Out[•]//TableForm=

0	2.96481	0.00145823	1.74265	538.158	1.00554	0.00150193	21
0	2.74406	0.000607772	1.76573	245.215	1.00874	0.000620321	30
0	2.88019	−0.000104138	2.47322	547.98	1.35958	0.000177041	25
0	3.62558	2.24981×10^{-17}	2.29197	686.502	1.39369	0.000194232	19
0	4.02317	−0.000301854	2.38109	657.601	1.58786	0.000596114	26

C10

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

Out[•]//TableForm=

0	7.71188	0.0002	1.47987	160.138	0.999958	0.0002	2
0	7.85112	0.000395857	1.65303	234.528	1.00152	0.000540053	1
0	8.8445	−0.000100955	2.32637	497.549	1.09199	0.000166779	2
0	7.99295	−0.000113042	2.27521	615.792	1.10197	0.0000738803	3
0	8.53216	−0.000353874	2.33672	582.456	1.04938	0.000298449	2

Out[•]//TableForm=

0	7.08688	0.000126294	1.17872	71.2006	0.999794	0.0002
0	7.64174	0.000387083	1.45098	200.209	0.996193	0.00035278
0	6.85199	−0.000210674	2.31357	493.191	1.08148	0.000162192
0	7.74767	−0.000541889	2.23401	487.484	1.10011	−0.0000273294
0	7.11932	−0.000405998	2.2995	572.054	0.860995	2.13752×10^{-17}

Out[•]//TableForm=

0	9.16516	0.0007	2.51777	247.957	1.00488	0.0007	2.
0	8.51897	0.000909127	1.80949	285.43	1.01441	0.00103408	33
0	9.77387	−0.0000253857	2.34432	553.317	1.0926	0.0002	3.
0	10.4952	−0.0001	2.33996	627.118	1.22113	0.000120457	35
0	10.3482	−0.00021438	2.34295	632.372	1.06134	0.000530659	2.

D

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

Out[•]//TableForm=

0	0.158654	0.000641456	1.49144	181.605	1.01159	0.000623127	0
0	0.150768	0.000526167	1.55454	259.515	1.00027	0.000529527	2
0	0.671419	−0.000149229	2.30186	528.196	4.27138	0.000105078	2
0	0.898694	−0.000208572	2.29933	564.662	4.84035	0.0000896786	2
0	1.36737	−0.000260088	2.33487	666.792	5.7546	0.0000457998	2

Out[•]//TableForm=

0	0.157877	0.000618486	1.47258	178.885	1.	0.000589261	0
0	0.131724	0.000418059	1.53529	232.122	1.00016	0.000422226	1
0	0.637257	−0.000151934	2.30156	522.979	3.72297	0.0000797056	2
0	0.850823	−0.000228866	2.29639	552.822	4.75232	0.0000733992	2
0	1.36318	−0.000284118	2.32785	645.802	5.41601	0.0000407914	2

Out[•]//TableForm=

0	0.171913	0.000743647	1.50705	191.866	1.02839	0.000743647	1
0	0.164601	0.000633954	1.63125	278.805	1.0163	0.000671117	2
0	0.730297	−0.000113274	2.30305	537.903	4.78312	0.00012441	2
0	0.921959	−0.0002	2.30794	568.041	5.83534	0.0000900679	2
0	1.41882	−0.000257655	2.33928	667.852	7.29914	0.0000855821	2

Nv

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

Out[]//TableForm=

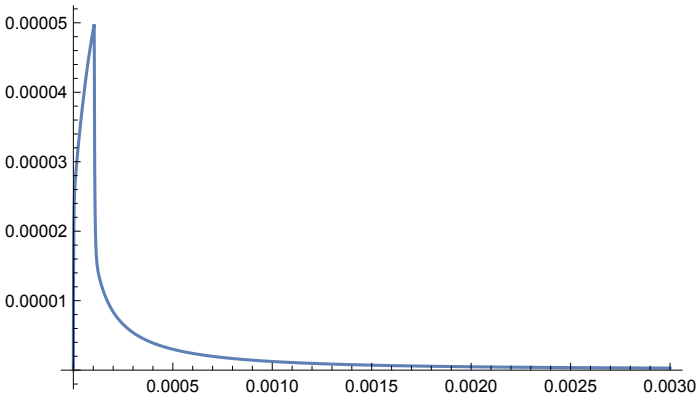
1.15831×10^{-8}	4.04537×10^{-7}	0.000100151	0.00143825	0.00304654	0.02252
0.0000225289	0.000699228	0.0977688	0.822265	1.21973	2.54577
0.000047581	0.00135246	0.166414	1.06648	1.4482	2.91692
0.0121758	0.161133	1.17424	2.08574	2.40481	7.25498
0.0332647	0.30843	1.27397	2.11265	2.42716	7.41929
0.183084	0.716014	1.44388	2.18153	2.49566	7.86737

EPSC with different caRest

Interpolate

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]
```

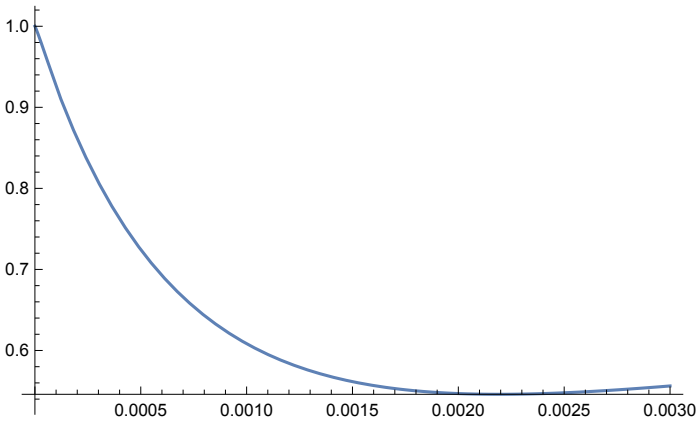
Out[]=



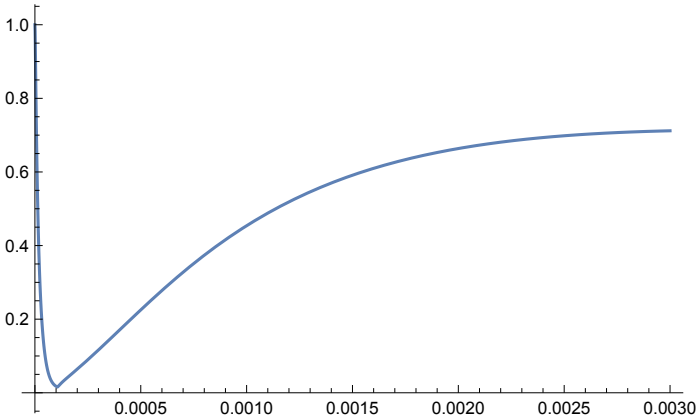
NDSolve

In[]:=
```
timeStartForPLot = 0.0;
timeEndForPLot = 0.003;
myNDSolveResults = NDSolve[eq,
    {fillStateSS, ss1, ss2, ss3, ss4, ss5, ss6, ss7, sitePlugging}, {t, 0, 0.003}];
Plot[(fillStateSS[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss1[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss2[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss3[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss4[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss5[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss6[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss7[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(sitePlugging[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
```

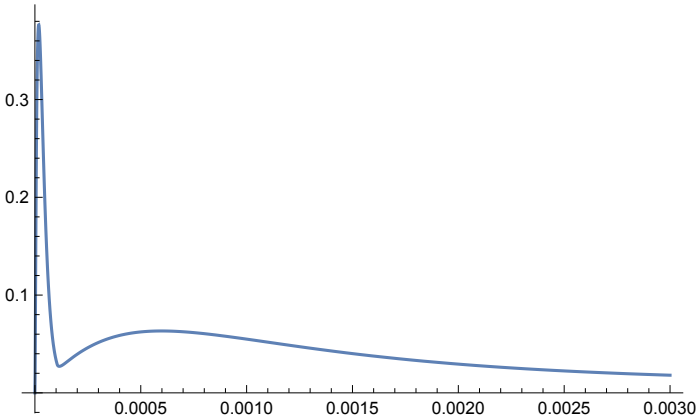
Out[]=



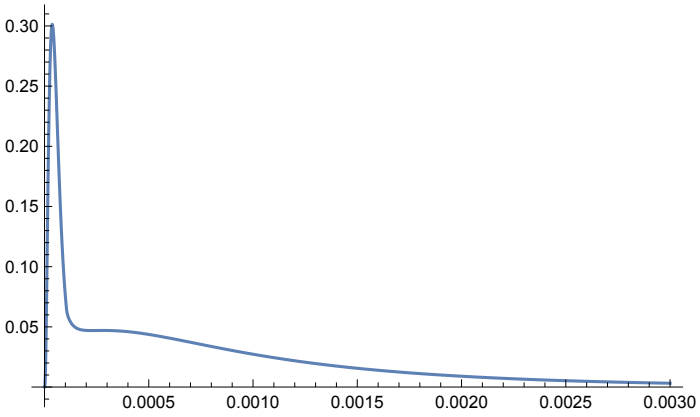
Out[•]=



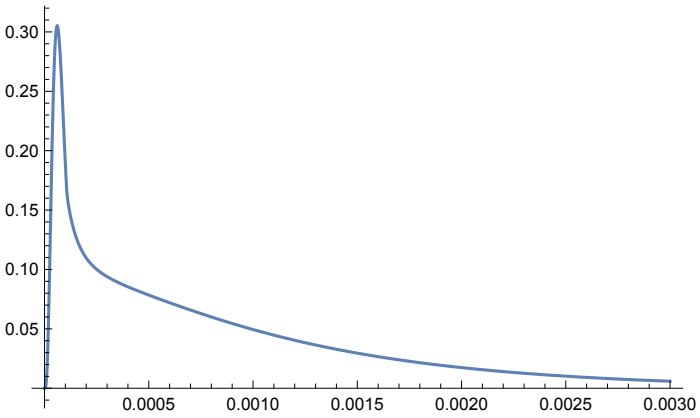
Out[•]=



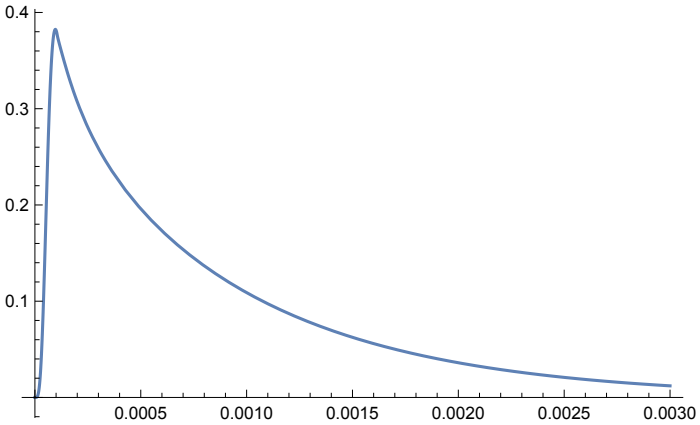
Out[•]=



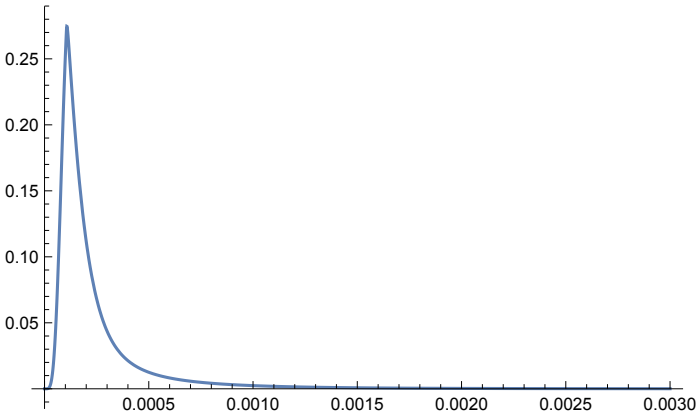
Out[•]=



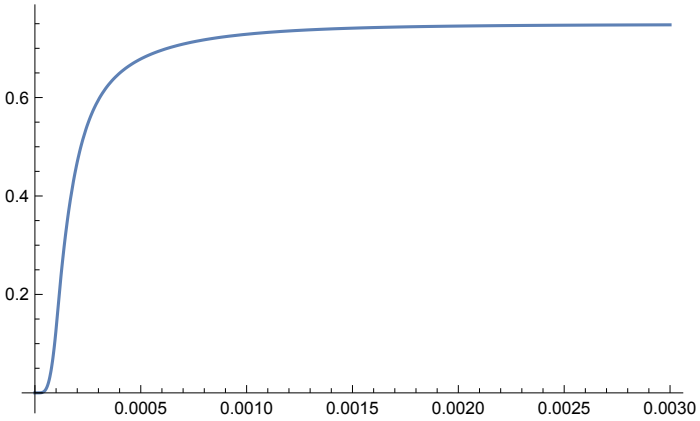
Out[•]=



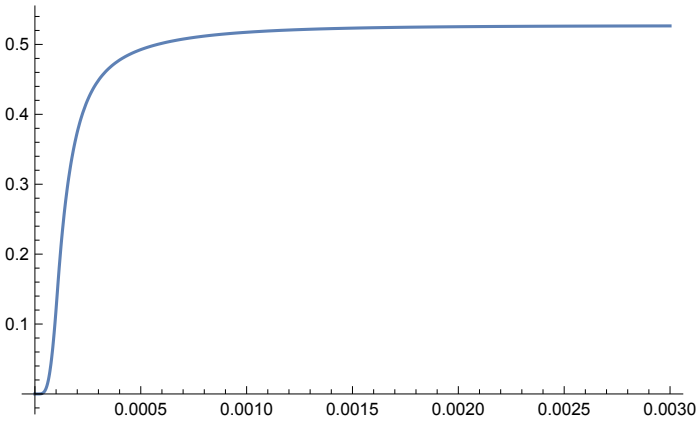
Out[•]=



Out[•]=



Out[•]=



different caRest

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

Low Ca

Initial occupancy

```
In[•]:= (*calualte initial equilibrium occupancy*)
        caFunc[t_] := caRestLow;
        kprimScheme
        kunprimScheme
        fillStateSSInitial = kprimScheme / kunprimScheme
        kfillScheme
        kunfillScheme
        ss0Initial = fillStateSSInitial * kfillScheme / kunfillScheme
```

```
Out[•]= 3.3867
```

```
Out[•]= 8.61585
```

```
Out[•]= 0.393078
```

```
Out[•]= 111.823
```

```
Out[•]= 181.545
```

```
Out[•]= 0.242117
```

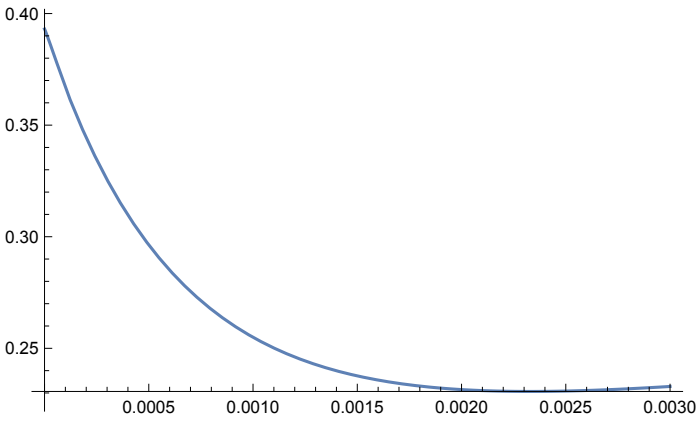
Diff eq.

```
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];
        ss[t_] = {ss1[t], ss2[t], ss3[t], ss4[t], ss5[t], ss6[t], ss7[t]};
        eq = {ss'[t] == (mat /. repl).ss[t],
            ss[0] == {ss0Initial, 0., 0., 0., 0., 0., 0.},
            (fillStateSS'[t] == kprim - kunprim fillStateSS[t] -
                 kfill fillStateSS[t] + kunfill ss1[t]) /. repl,
            fillStateSS[0] == fillStateSSInitial,
            sitePlugging'[t] ==
              (1 - sitePlugging[t]) ss7'[t] - siteClearanceTau sitePlugging[t],
            sitePlugging[0] == 0
           };
```

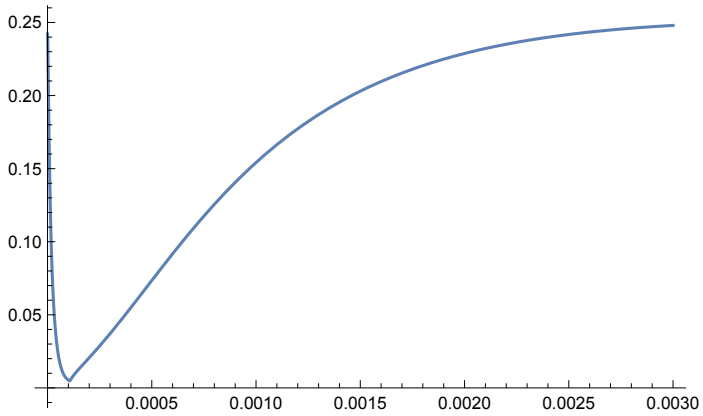
NDSolve

In[•]:=
```
myNDSolveResults = NDSolve[eq,
    {fillStateSS, ss1, ss2, ss3, ss4, ss5, ss6, ss7, sitePlugging}, {t, 0, 0.003}];
Plot[(fillStateSS[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss1[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss2[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss3[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss4[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss5[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss6[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(ss7[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
Plot[(sitePlugging[t] /. myNDSolveResults),
 {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
```

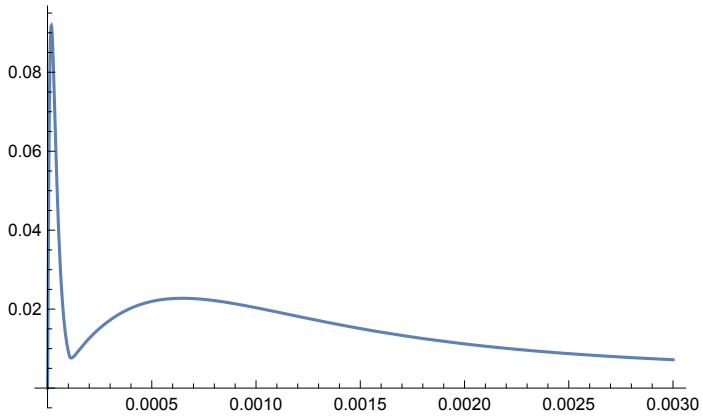
Out[•]=



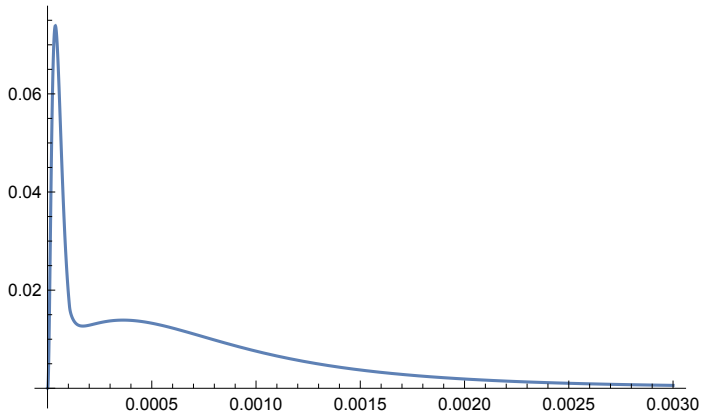
Out[ ]=



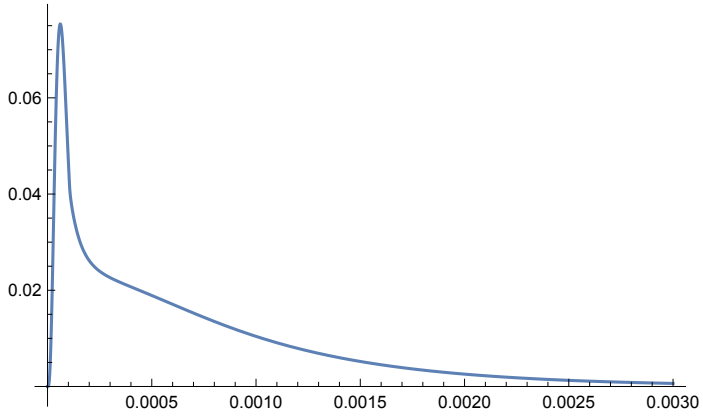
Out[ ]=



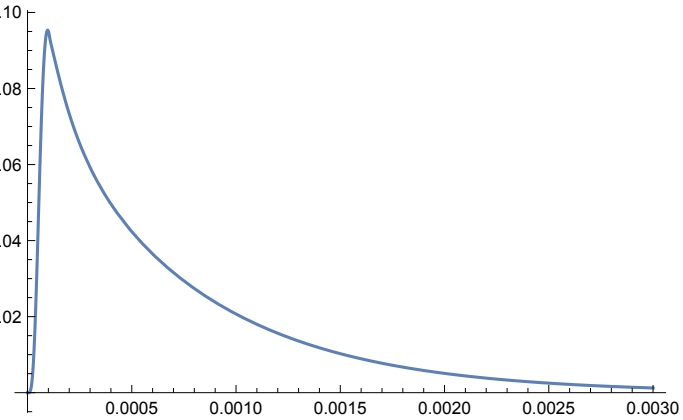
Out[ ]=



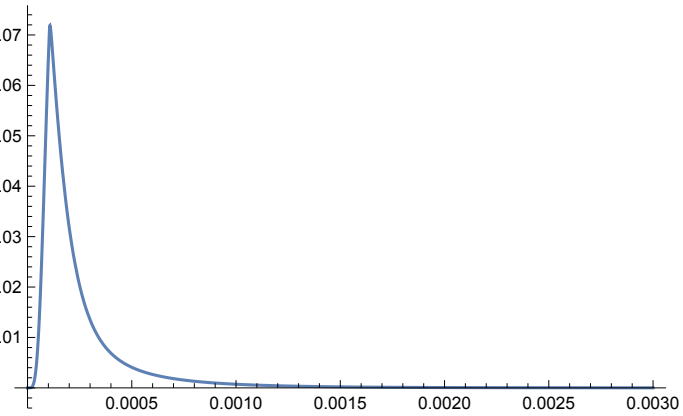
Out[ ]=



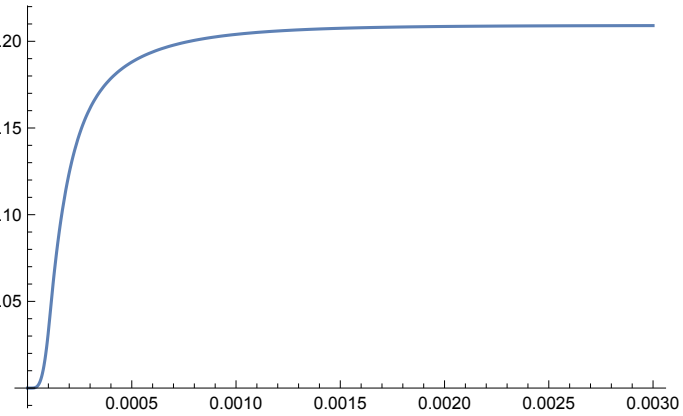
Out[•]=



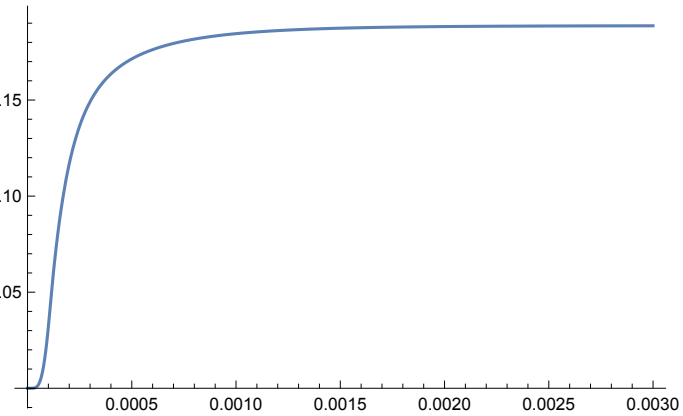
Out[•]=



Out[•]=



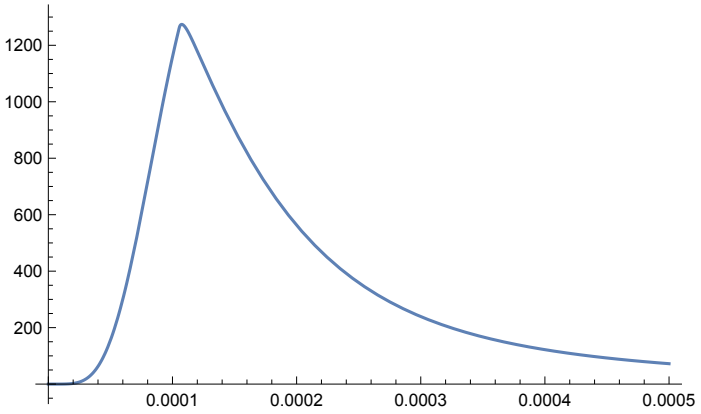
Out[•]=



Plot EPSC

```
In[•]:= epscLowCa = D[(ss7[t] /. myNDSolveResults), t];
        Plot[(ss7[t] /. myNDSolveResults), {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;
        kprimScheme
        kunprimScheme
        fillStateSSInitial = kprimScheme / kunprimScheme
        kfillScheme
        kunfillScheme
        ss0Initial = fillStateSSInitial * kfillScheme / kunfillScheme
```

Out[•]= 7.45413

Out[•]= 8.61585

Out[•]= 0.865165

Out[•]= 166.055

Out[•]= 181.545

Out[•]= 0.791348

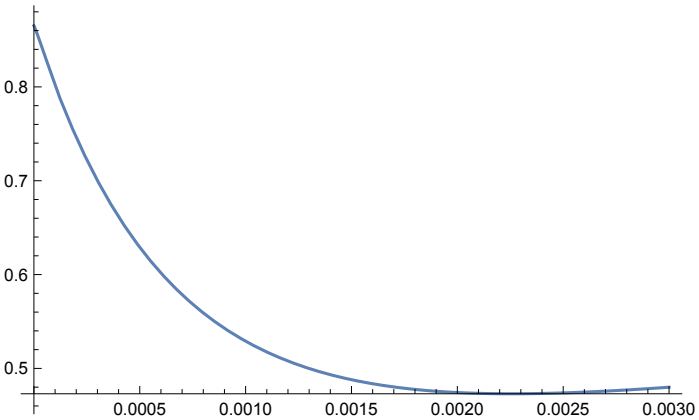
Diff eq.

```
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];
        ss[t_] = {ss1[t], ss2[t], ss3[t], ss4[t], ss5[t], ss6[t], ss7[t]};
        eq = {ss'[t] == (mat /. repl).ss[t],
           ss[0] == {ss0Initial, 0., 0., 0., 0., 0., 0.},
           (fillStateSS'[t] == kprim - kunprim fillStateSS[t] -
                kfill fillStateSS[t] + kunfill ss1[t]) /. repl,
           fillStateSS[0] == fillStateSSInitial,
           sitePlugging'[t] ==
            (1 - sitePlugging[t]) ss7'[t] - siteClearanceTau sitePlugging[t],
           sitePlugging[0] == 0
          };
```

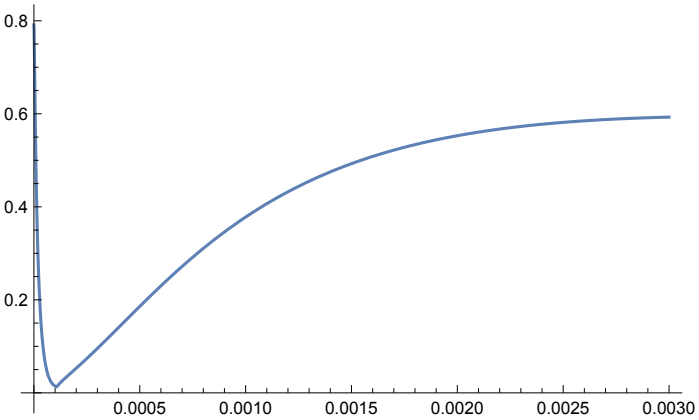
NDSolve

```
In[•]:= myNDSolveResults = NDSolve[eq,
           {fillStateSS, ss1, ss2, ss3, ss4, ss5, ss6, ss7, sitePlugging}, {t, 0, 0.003}];
        Plot[(fillStateSS[t] /. myNDSolveResults),
         {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
        Plot[(ss1[t] /. myNDSolveResults),
         {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
        Plot[(ss2[t] /. myNDSolveResults),
         {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
        Plot[(ss3[t] /. myNDSolveResults),
         {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
        Plot[(ss4[t] /. myNDSolveResults),
         {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
        Plot[(ss5[t] /. myNDSolveResults),
         {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
        Plot[(ss6[t] /. myNDSolveResults),
         {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
        Plot[(ss7[t] /. myNDSolveResults),
         {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
        Plot[(sitePlugging[t] /. myNDSolveResults),
         {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
```

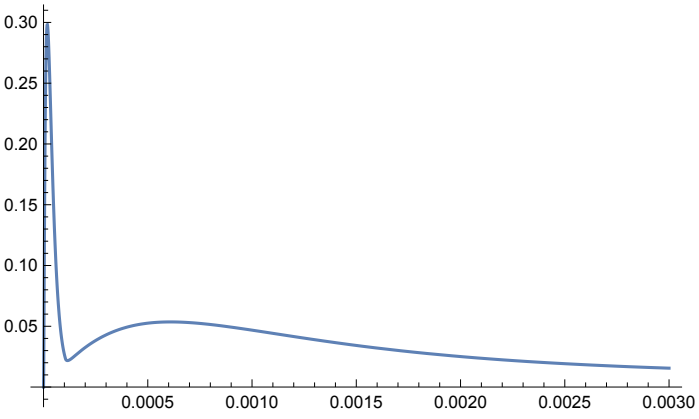
Out[•]=



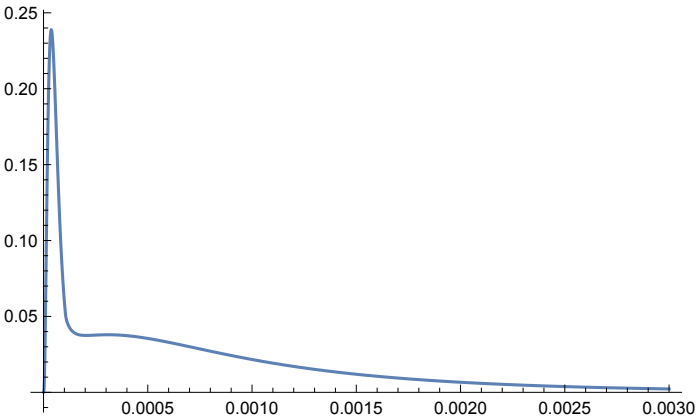
Out[•]=



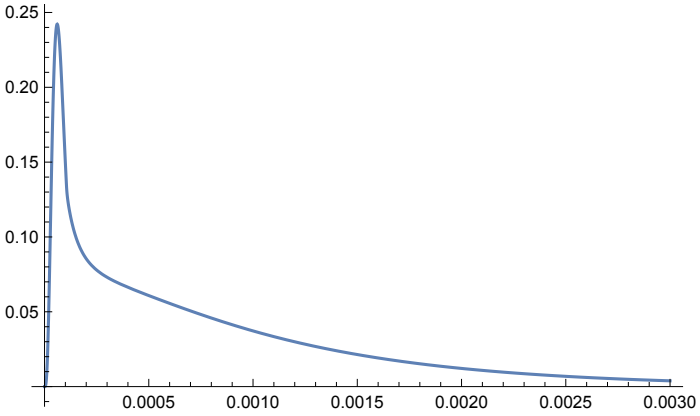
Out[•]=



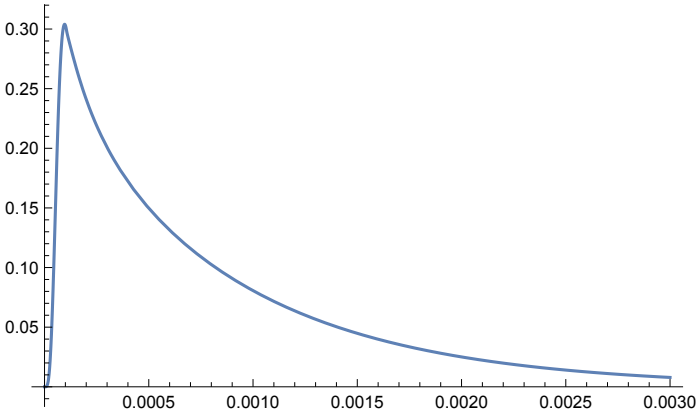
Out[•]=



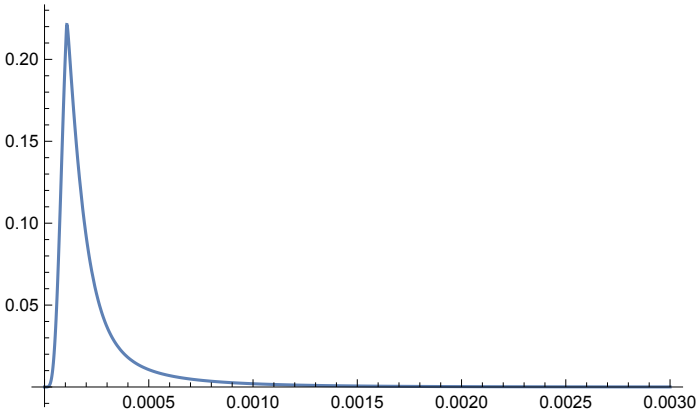
Out[•]=



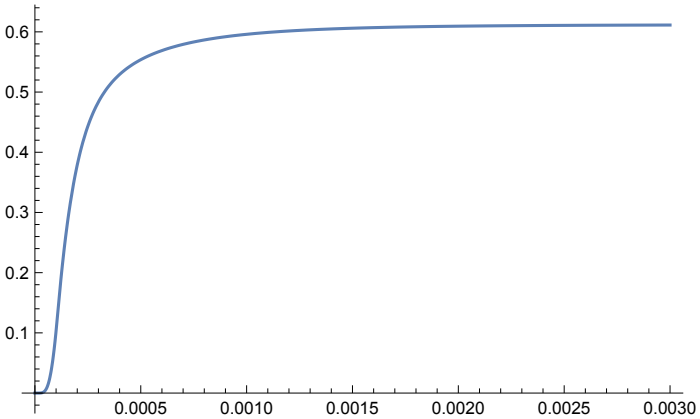
Out[•]=



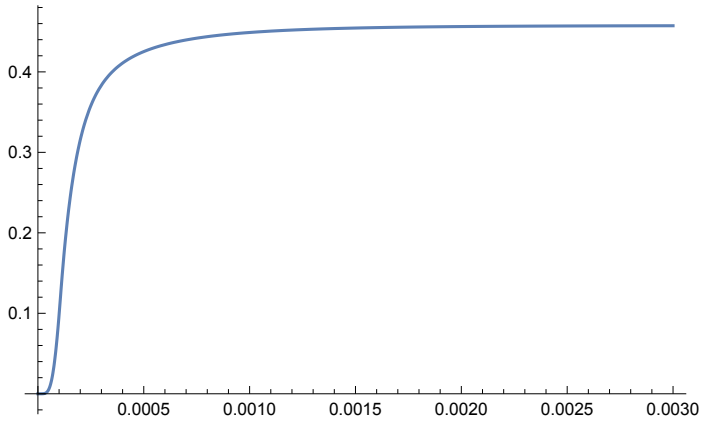
Out[•]=



Out[•]=



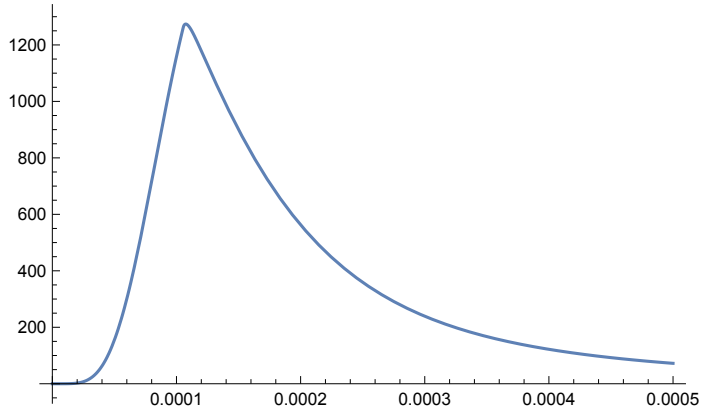
Out[•]=



Plot EPSC

In[•]:=
```
epscHighCa = D[(ss7[t] /. myNDSolveResults), t];
Plot[(ss7[t] /. myNDSolveResults), {t, 0, 2*^-3}, PlotRange → All];
Plot[epscLowCa, {t, 0, 0.5*^-3}, PlotRange → All]
```

Out[•]=

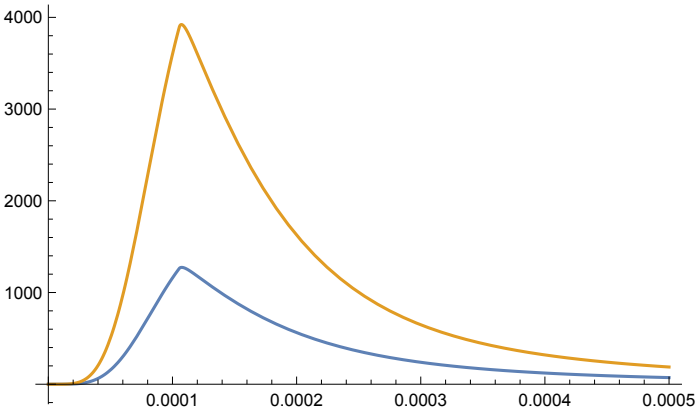


Compare

Plot both release rates

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

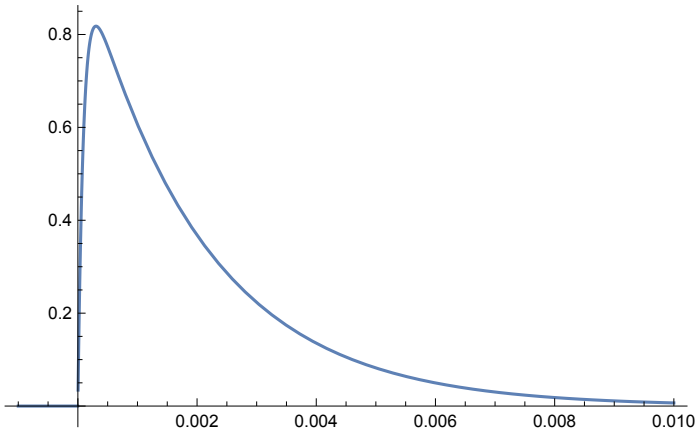
Out[ ]=



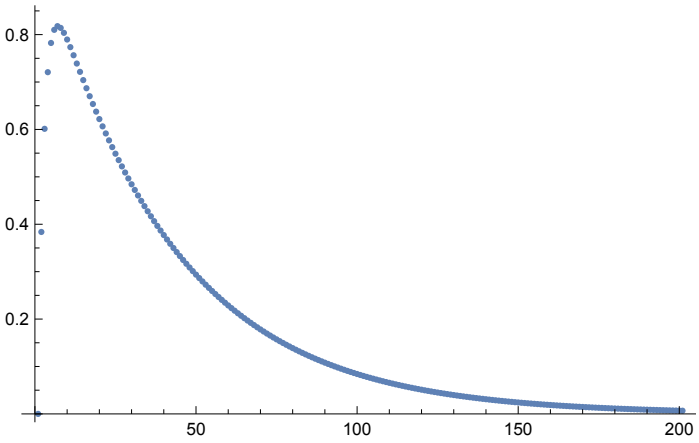
Convolution RelRate => EPSC

```
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]
```

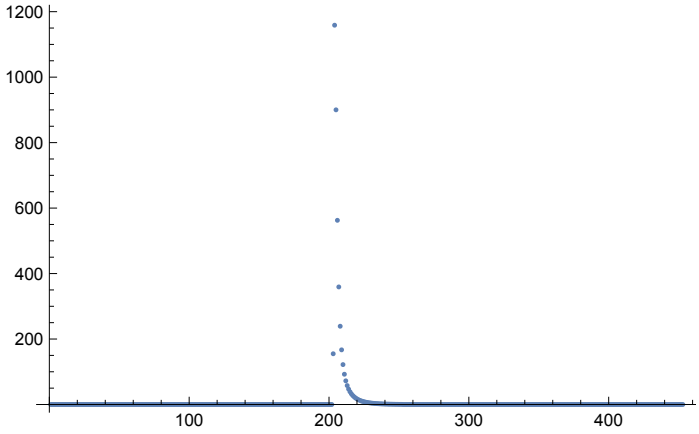
Out[ ]=



Out[•]=



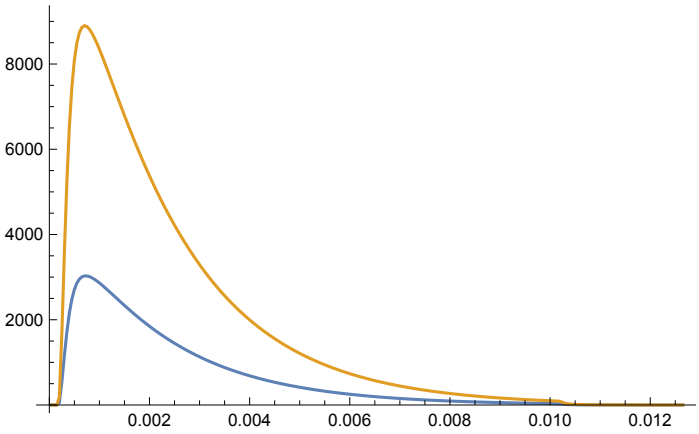
Out[•]=



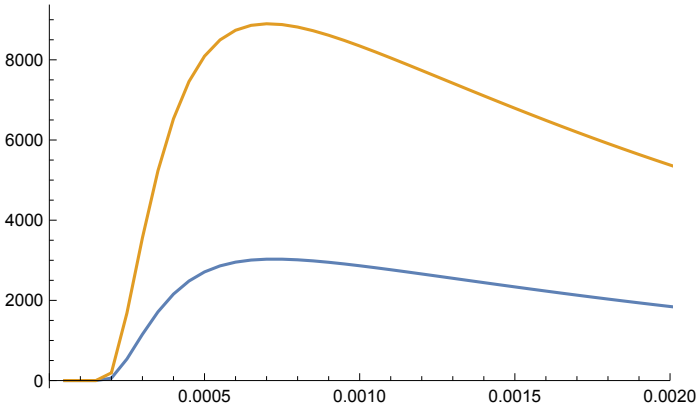
In[•]:=

```
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[•]=

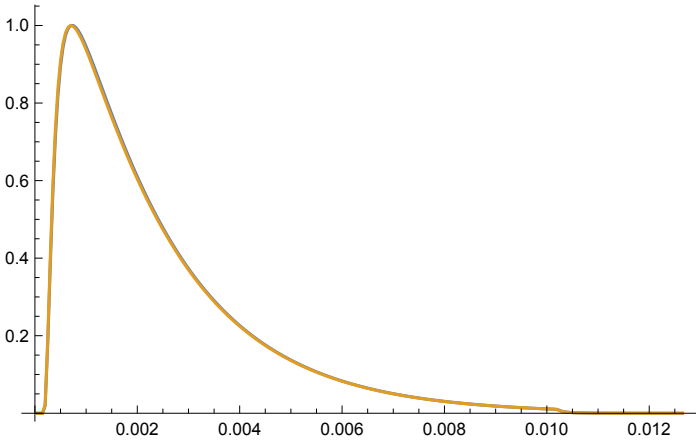


Out[•]= 3027.59

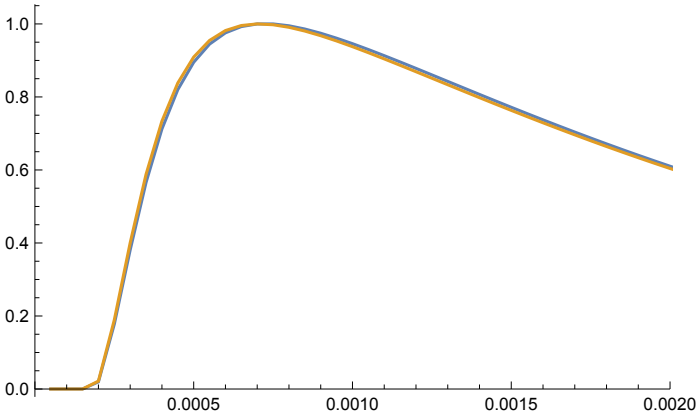
Out[•]= 8899.31

```
maxHigh/maxLow = 2.9394
```

Out[•]=



Out[•]=



Timing

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

Out[•]=
```
0.314477
```