```
In[1]:= (* Mathematica code for Model1 of Eshra et al. eLife 2021 *)
       (* Stefan Hallermann and Hartmut Schmidt Aug 2021 *)
```

Import

general

```
In[2]:= (* all Ca in uM, all times in ms, all ampltiudes and Nv in vesicels *)
```

```
In[3]:= CmToVesConversionFactor = (1/90.12) * (1/70*^-18);(* explained in methods *)
```

```
In[4]:= rrp = 10;(* pool of release-ready vesicles per connection *)
```

```
In[5]:= dir = NotebookDirectory[];
       SetDirectory[dir];
       dataFolder = "../data to fit/";
```

tau1 Cm 5kHz

```
In[8]:= data = Import[dataFolder <> "all_t1_v02_C5.txt", "Table"];
```

```
In[9]:=  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[22]:=  data = Import[dataFolder <> "all_t1_v02_C10.txt", "Table"];
```

```
In[23]:= 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[36]:= data = Import[dataFolder <> "all_t1_v02_D.txt", "Table"];
```

```
In[37]:= dataT1DCa = 0.001 * data[[All, 1]];
        dataT1DRelRate = 1000. * data[[All, 2]];
        dataT1DDelay = data[[All, 3]];
        dataT1DChiRatio = data[[All, 4]];
        dataT1DAmplitude = data[[All, 5]];

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

tau2 Cm 5kHz

```
In[50]:= data = Import[dataFolder <> "all_t2_v02_C5.txt", "Table"];
```

```
In[51]:= dataT2C5Ca = 0.001 * data[[All, 1]];
        dataT2C5RelRate = 1000. * data[[All, 2]];
        dataT2C5Amplitude2 = CmToVesConversionFactor data[[All, 3]];
        dataT2C5Amplitude1 = CmToVesConversionFactor data[[All, 4]];
```

tau2 Cm 10kHz

```
In[55]:= data = Import[dataFolder <> "all_t2_v02_C10.txt", "Table"];
```

```
In[56]:= dataT2C10Ca = 0.001 * data[[All, 1]];
        dataT2C10RelRate = 1000. * data[[All, 2]];
        dataT2C10Amplitude2 = CmToVesConversionFactor data[[All, 3]];
        dataT2C10Amplitude1 = CmToVesConversionFactor data[[All, 4]];
```
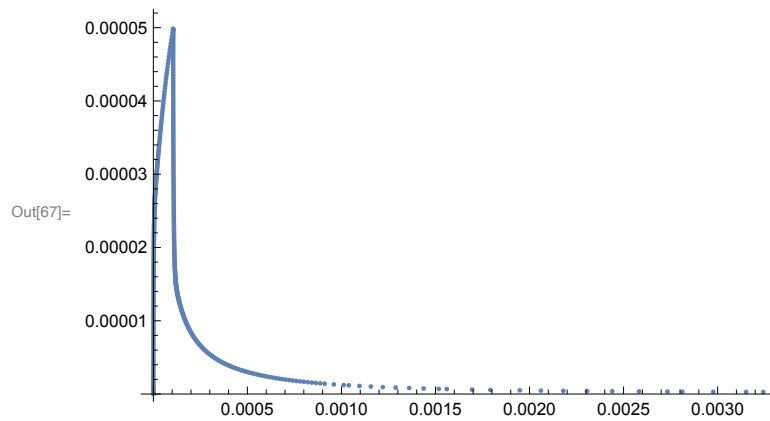
tau2 Deconv

```
In[59]:= data = Import[dataFolder <> "all_t2_v02_D.txt", "Table"];
```

```
In[60]:= dataT2DCa = 0.001 * data[[All, 1]];
        dataT2DRelRate = 1000. * data[[All, 2]];
        dataT2DAmplitude2 = data[[All, 3]];
        dataT2DAmplitude1 = data[[All, 4]];
```

local Ca

In[64]:= ```
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[67]=



## local Ca

# General parameters and definitions

## general stuff

```
In[68]:= (* 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[75]:= timeStart = AbsoluteTime[]
```

$$\text{Out[75]= } 3.841510347261920 \times 10^{9}$$

noiseRepeats

In[76]:= `noiseRepeats = 3;`
`(* should be increased to 50 for a full dataset *)`
`myQuantile1 = 0.25;`
`myQuantile2 = 0.75;`

export parameters

In[79]:= `dtOfPlotsForExport = 20*^-5;`
`exportYes = 1;`

sampling and myNoise

In[81]:= `samplingOfDataInKHzC5 = 5;`
`myNoiseC5 = 0.1;(*cannot be 0*)`
`signalToNoiseRatioC5 = 1.;(*minimum s-to-n-ratio to attempt fitting*)`
$dtOfDataC5 = \left(1 \big/ \left(1000 * samplingOfDataInKHzC5\right)\right);$

`samplingOfDataInKHzC10 = 10;`
`myNoiseC10 = 0.1;(*cannot be 0*)`
`signalToNoiseRatioC10 = 1.;(*minimum s-to-n-ratio to attempt fitting*)`
$dtOfDataC10 = \left(1 \big/ \left(1000 * samplingOfDataInKHzC10\right)\right);$

`samplingOfDataInKHzD = 10;`
`myNoiseD = 0.02;(*cannot be 0*)`
`signalToNoiseRatioD = 1.;(*minimum s-to-n-ratio to attempt fitting*)`
$dtOfDataD = \left(1 \big/ \left(1000 * samplingOfDataInKHzD\right)\right);$

`samplingOfDataInKHzLong = 1;`
`myNoiseLong = 0.1;(*cannot be 0*)`
$dtOfDataLong = \left(1 \big/ \left(1000 * samplingOfDataInKHzLong\right)\right);$

number of simulations per DMN

```
In[96]:= aNumberDMN05 = 2;
         aNumberDMN2 = 2;
         aNumberDMN10 = 2;

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

Exp fit function

```
In[99]:= 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[105]:= pre_final01

Out[105]= 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[106]:= CaListReal = CaListDye = {};
```

0.5 mM DMN

general parameters

In[107]:=
```
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[124]:=
```
(*Dye*)
If[OGB1 == 1,
  kOnOG  = 4.3 * 10⁸;
kOffOG = 103.;]

If[OGB5N == 1,
  kOnOG  = 2.5 * 10⁸;
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[154]:= 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[171]:= (*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[201]:= TimeWindow = 0.006; (*End of simulation*)
        tflash = 0.0;  (*Time of flash*)
        PlStart = 0.; (*Plot start*)

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

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

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

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

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

definitions and loop

```
In[218]:= (*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[248]:=
```
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[248]=

Real free Ca (µM)

Out[249]=

Free Ca reported by dye (µM)

Out[250]=

Real free Ca (µM}

Out[251]=

Free Ca reported by dye (µM)

Out[252]= $\left\{7.03073 \times 10^{-7}, \, 4.79194 \times 10^{-6}, \, 5.70182 \times 10^{-6}, \right.$
$\left. 0.0000262105, \, 0.0000304385, \, 0.0000745861\right\}$

In[253]:= `runs = Length[CaListDyePeak];`

Release scheme and parameters

Define release scheme

In[254]:= `nStates = 4;`
`mat = Table[0, {nStates}, {nStates}];`

forward rates

```
In[256]:= from = 1;(*0 ca bound to 1 ca bound*)
    kk = 2 kon;
    mat[[from + 1, from]] += kk; mat[[from, from]] += -kk;
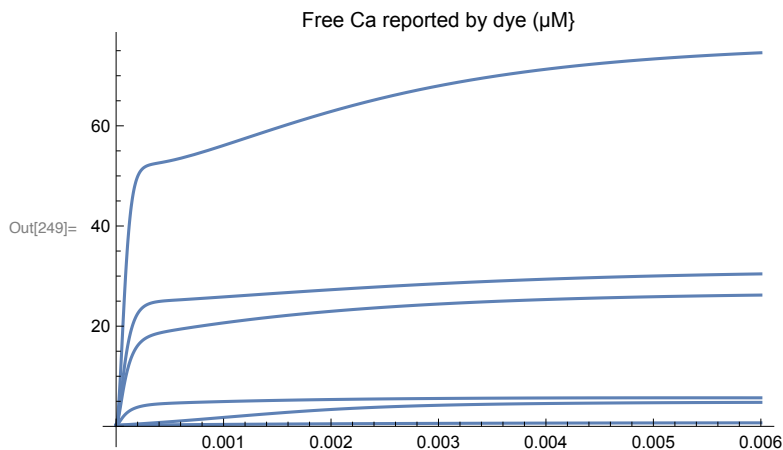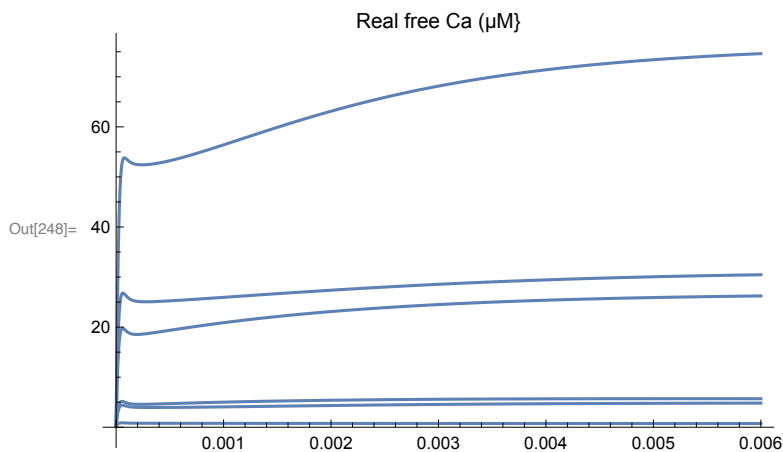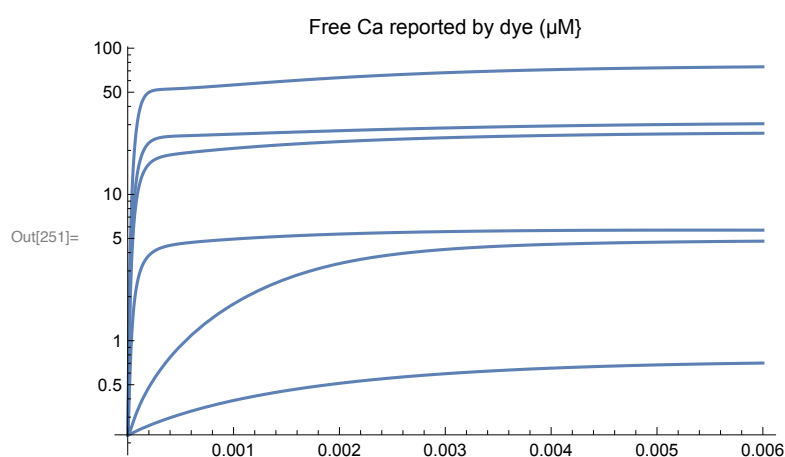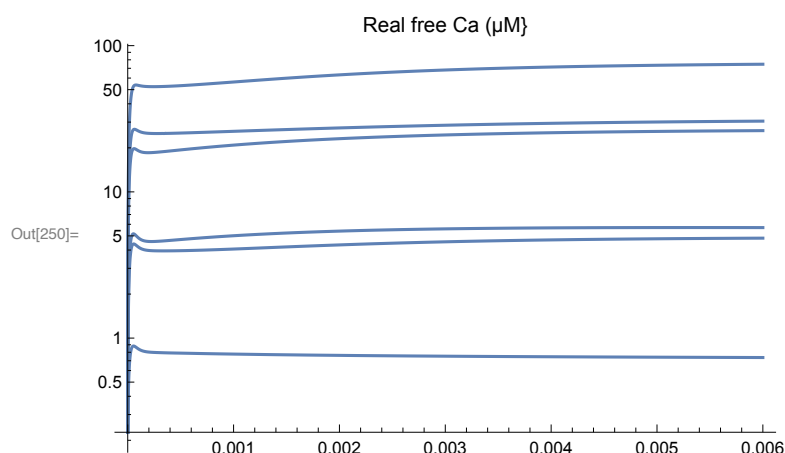

    from = 2;(*1 ca bound to 2 ca bound*)
    kk = 1 kon;
    mat[[from + 1, from]] += kk; mat[[from, from]] += -kk;


    from = 3;(*from 2 ca bound to fused*)
    kk = gamma;
    mat[[from + 1, from]] += kk; mat[[from, from]] += -kk;
```

backwards rates

```
In[265]:= from = 2;(*1 ca bound to 0 ca bound*)
    kk = koff b^0;
    mat[[from - 1, from]] += kk; mat[[from, from]] += -kk;


    from = 3;(*2 ca bound to 1 ca bound*)
    kk = 2 koff b^1;
    mat[[from - 1, from]] += kk; mat[[from, from]] += -kk;
```

outflux from matrix

```
In[271]:= mat[[1, 1]] += -kunprim;
```

influx in matrix

```
In[272]:= mat[[1, 1]] += kprim / ss1[t];
```

Matrix

```
In[273]:= mat // TableForm
```

Out[273]//TableForm=

$-2\,kon - kunprim + \frac{kprim}{ss1[t]}$	koff	0	0
2 kon	$-koff - kon$	2 b koff	0
0	kon	$-gamma - 2\,b\,koff$	0
0	0	gamma	0

Parameters of release scheme

```
In[274]:= q10 = 2.3;
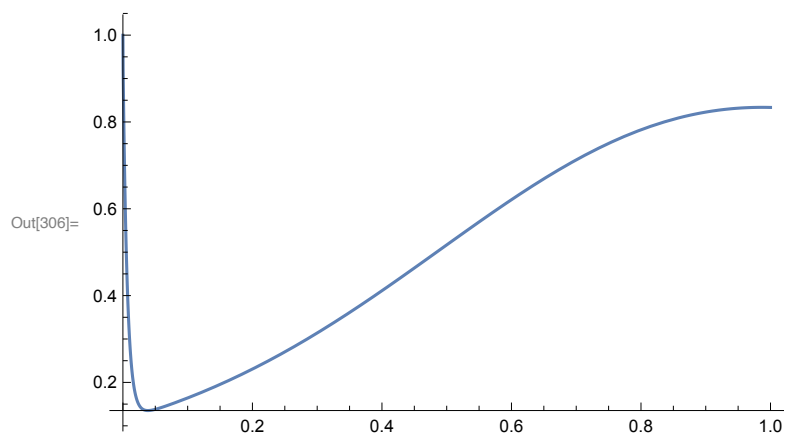       tempFact = q10 ^ ((37 - 24) / 10);
```

```
In[276]:= 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 = 1.0;
       konScheme = 100. * caFunc[t] Sqrt[affinityFactor] tempFact 1.*^7;(*M^-1 s^-1*)
       koffScheme = 100. * (1 / Sqrt[affinityFactor]) tempFact 1500;(*s^-1*)
       gammaScheme = 10. * tempFact 6000;(*s^-1*)
       bScheme = 0.25;


       KdPrim = 2.*^-6;
       kprimScheme = 0.6 + 30. (caFunc[t] / (KdPrim + caFunc[t]));
       kunprimScheme = 0.6 + 30. (caRest / (KdPrim + caRest));


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

           kprim → kprimScheme ,
           kunprim → kunprimScheme

         };
```

```
In[287]:= tauOfDecayOfUncagedCa = 0.4;
```

Initial occupancy

```
In[288]:= (*test initial equilibrium occupancy*)
    caFunc[t_] := caTmp;
    fillStateSSInitial = kprimScheme / kunprimScheme;
    ss0Initial = fillStateSSInitial;
    (ss0Initial /. caTmp → 180*^-9)
    (ss0Initial /. caTmp → 30*^-9)
    (ss0Initial /. caTmp → 180*^-9) / (ss0Initial /. caTmp → 30*^-9)
    (*test initial *)
    Plot[kprimScheme / kunprimScheme, {caTmp, 0, 300*^-9}]
```

Out[291]= 0.841205

Out[292]= 0.28523

Out[293]= 2.94922


```
In[295]:= (*calualte initial equilibrium occupancy*)
    caFunc[t_] := caRest;
    kprimScheme
    kunprimScheme
    ss0Initial = kprimScheme / kunprimScheme
```

Out[296]= 3.65793

Out[297]= 3.65793

Out[298]= 1.

Diff eq.

```
In[299]:= 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]};
    eq = {ss'[t] == (mat /. repl).ss[t],
        ss[0] == {ss0Initial, 0, 0, 0}};
```

Solve all states

In[302]:= `caFunc[t_] := 3*^-6 * Exp[-t / tauOfDecayOfUncagedCa];;`
`Plot[caFunc[t], {t, 0, 0.4}, PlotRange → All]`

Out[303]=

In[304]:= `myNDSolveResults = NDSolve[eq, {ss1, ss2, ss3, ss4}, {t, 0, 10.}];`
`tEndForPlot = 1.;`
`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]`

Out[306]=

Out[307]=

Out[308]=

Out[309]=

Loop

make lists for later

```
In[310]:= simCaList = Table[0, {runs}];
       simParamNv = Table[0, {7}, {runs}];
```

C5

```
In[312]:= 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[317]:= 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[322]:= 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[327]:= baselineLong = Table[{ttt, 0}, {ttt, -0.05, -1*^-6, dtOfDataLong}];
```

Loop

```
In[328]:= 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, ss4, {t, 0, 0.4}];
(* plot results: *)
fused[t_] := ss4[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
  and do the fitting several times and then 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 improve(=decrease) by >4%*)
  (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 improve(=decrease) by >4%*)
 (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
   several times and do the fitting and than 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%*)
```

```mathematica
       (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   --
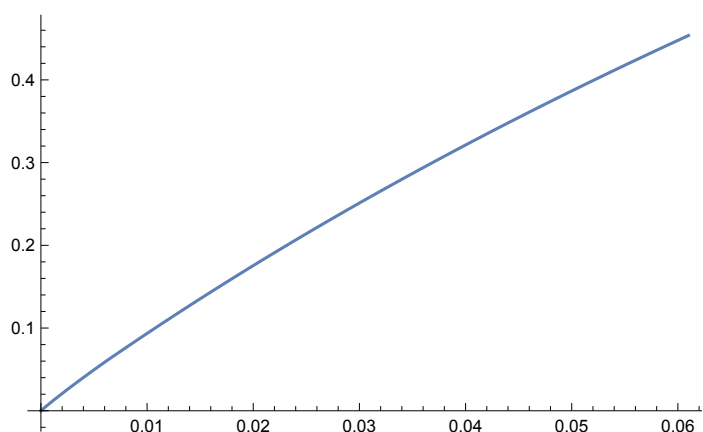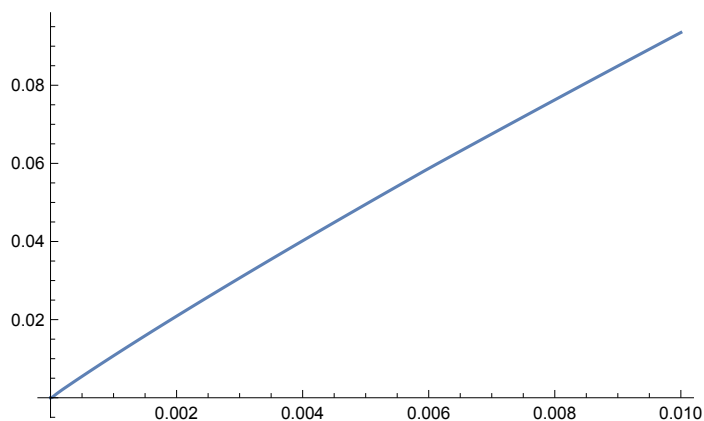
-- C5

-- C10

-- D

take mono

take mono

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

take mono
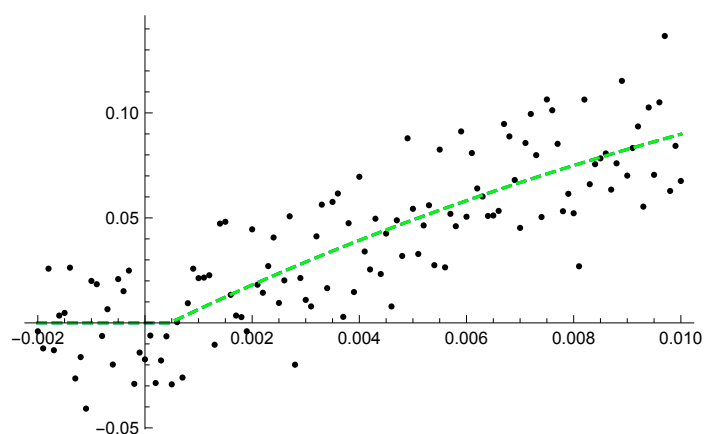
Mono: chi2 = 0.052073   d = 0.052073   a = 0.207514   t = 0.0167936

Bi: ratio chi2Mono/chi2Bi = 1.   d = 0.000470026   a =
   0.21825   a1 = 0.207217   t1 =0.0167806   t2 = 1.443

⚬⚬⚬ 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.

Long trace was used for fitting.

take mono

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

take mono

take mono
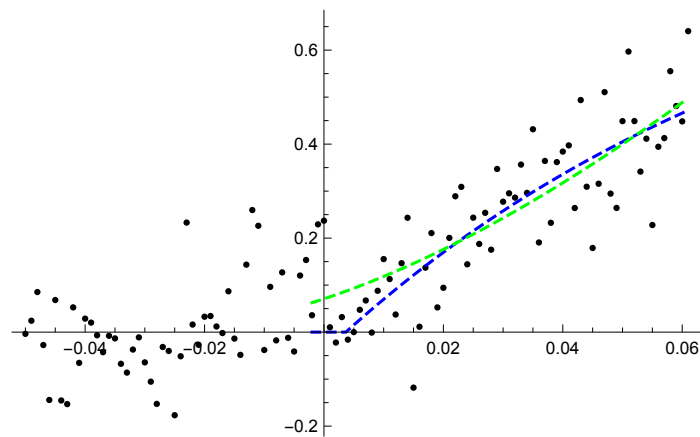
Mono: chi2 = 1.03411   d = 0.00366323   a = 0.933623   t = 0.0814763

Bi: ratio chi2Mono/chi2Bi = 1.05217   d = -0.028   a =
 9.46705   a1 = -4.83606   t1 =0.147526   t2 = 0.427475

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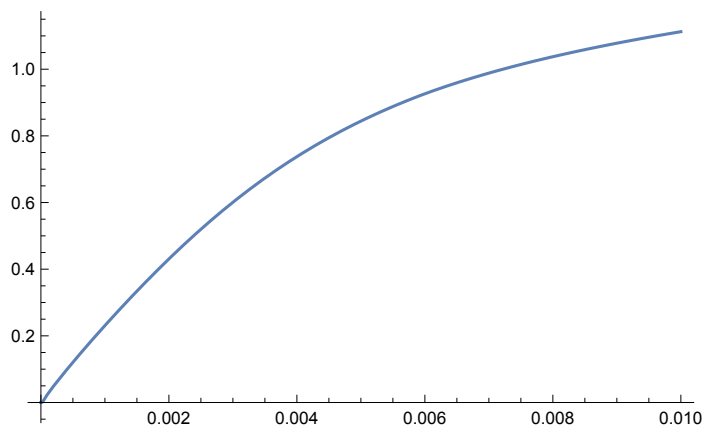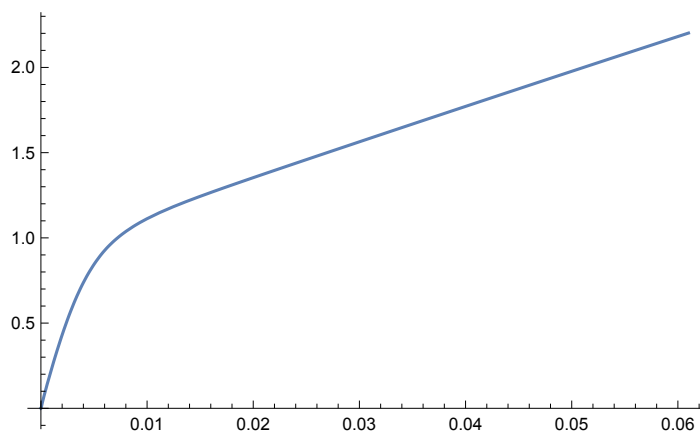

```
---------------------------------------------------------------------- C5
```

take mono

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
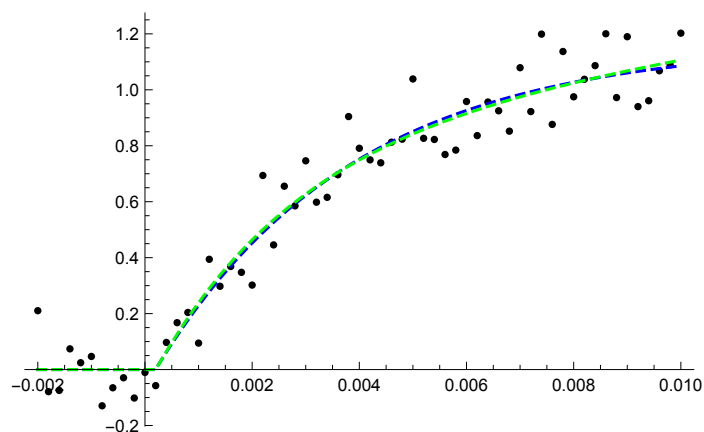

```
Mono: chi2 = 0.558169      d = 0.558169      a = 1.16632      t = 0.00367285

Bi: ratio chi2Mono/chi2Bi = 1.00805      d = 0.00020429      a =
  7.96209      a1 = 0.840996      t1 =0.00268227      t2 = 0.238056
```

```
---------------------------------------------------------------------- C10
```

take mono

take mono
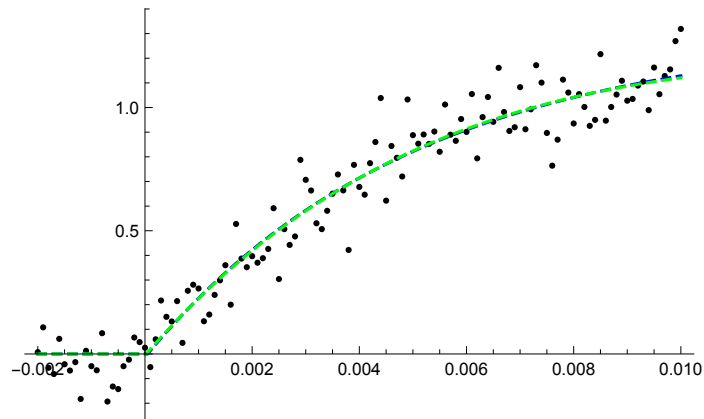
take mono

Mono: chi2 = 1.20884   d = 1.20884   a = 1.30731   t = 0.0049998

Bi: ratio chi2Mono/chi2Bi = 1.00234   d = 0.0000239968   a =
 0.777713   a1 = 2.85899   t1 =0.00736634   t2 = 0.0152262
-- D

take mono

take mono
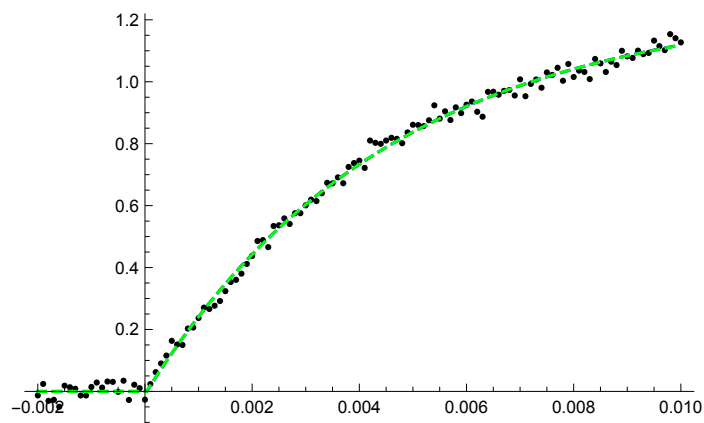
take mono

Mono: chi2 = 0.0558425   d = 0.0558425   a = 1.2585   t = 0.00453768

Bi: ratio chi2Mono/chi2Bi = 1.00026   d = 0.0000301442   a =
 1.06289   a1 = 1.3005   t1 =0.00465986   t2 = 0.0742347

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

--- C5

```
take mono

take mono

take mono
```
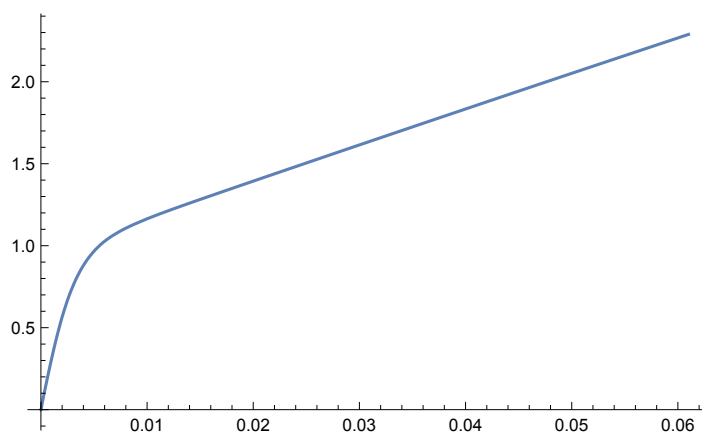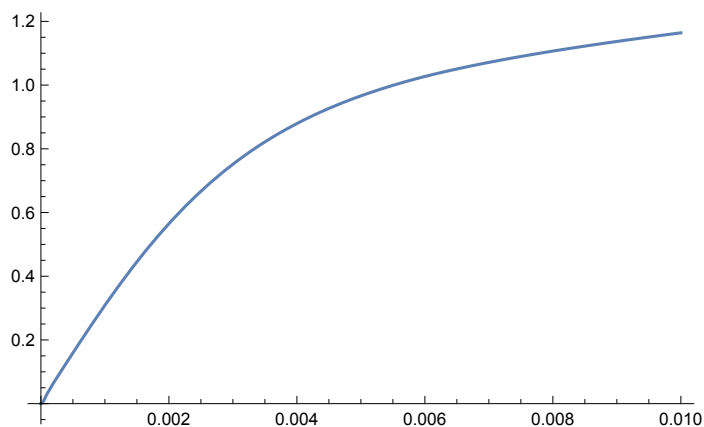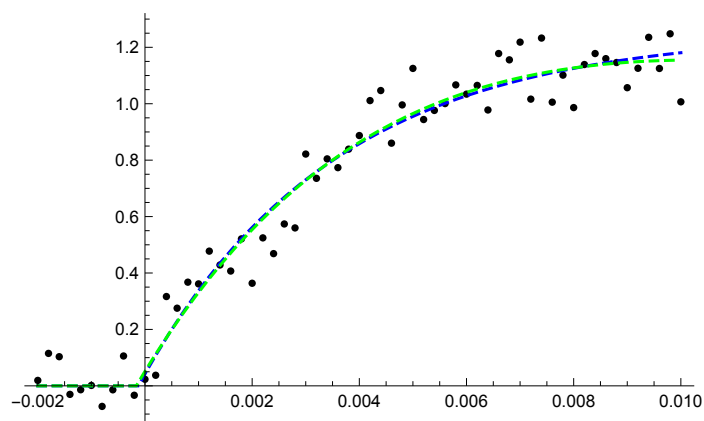


```
Mono: chi2 = 0.436364      d = 0.436364      a = 1.2548      t = 0.00357603

Bi: ratio chi2Mono/chi2Bi = 1.05093      d = -0.000161091      a =
  0.44176      a1 = 3.59182      t1 =0.00632169      t2 = 0.0128969
```

--- C10

```
take mono

take mono

take mono
```
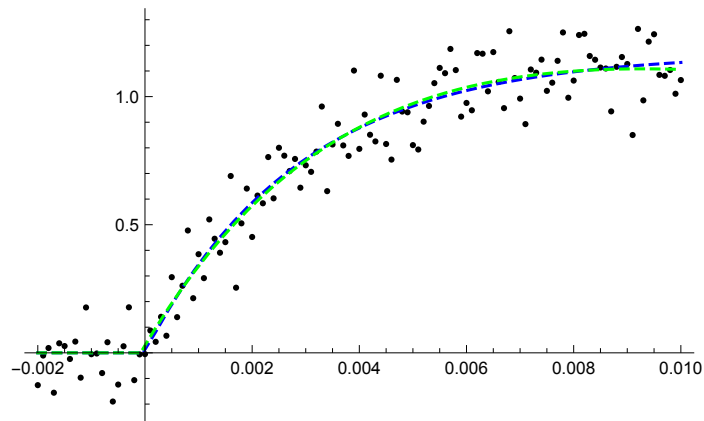
Mono: chi2 = 1.2785   d = 1.2785   a = 1.16986   t = 0.00289041

Bi: ratio chi2Mono/chi2Bi = 1.00915   d = -0.0000802886   a =
 0.344116   a1 = 2.22145   t1 =0.00469203   t2 = 0.0165497

--- D

take mono

take mono

take mono
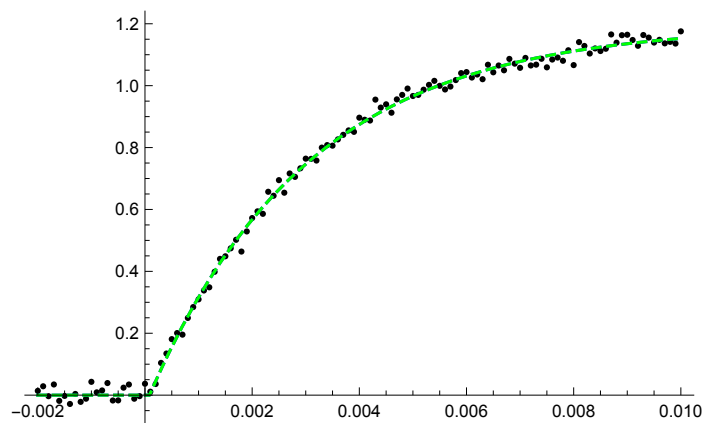
Mono: chi2 = 0.0444231   d = 0.0444231   a = 1.19511   t = 0.00297389

Bi: ratio chi2Mono/chi2Bi = 1.   d = 0.0000854186   a =
 1.28692   a1 = 1.1943   t1 =0.00297171   t2 = 1.24585

----------------   Ca = 26.2105
  uM   --
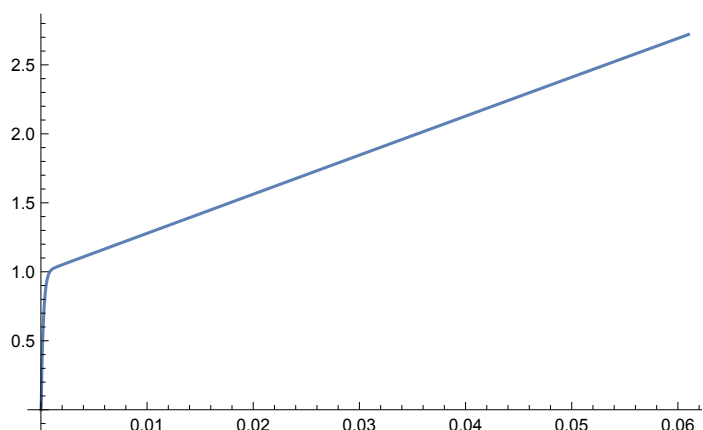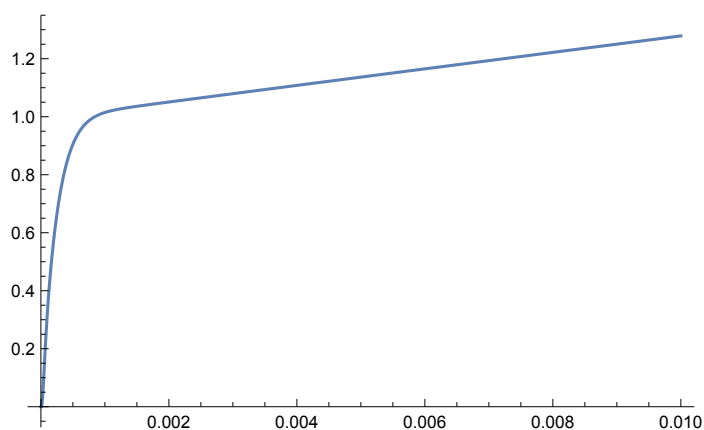
-- C5

take bi

⊙ General: Exp[−741.526] is too small to represent as a normalized machine number; precision may be lost.

⊙ General: Exp[−775.209] is too small to represent as a normalized machine number; precision may be lost.

⊙ General: Exp[−808.891] is too small to represent as a normalized machine number; precision may be lost.

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

take bi

take bi
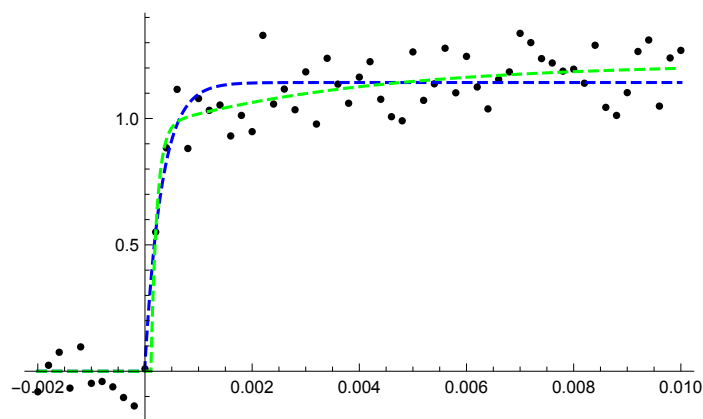

```
Mono: chi2 = 0.650755      d = 0.650755      a = 1.14245      t = 0.000314221

Bi: ratio chi2Mono/chi2Bi = 1.22686      d = 0.000109866      a =
  1.21845      a1 = 0.967146      t1 =0.000109483      t2 = 0.0038985
```

-- C10

take bi

take bi
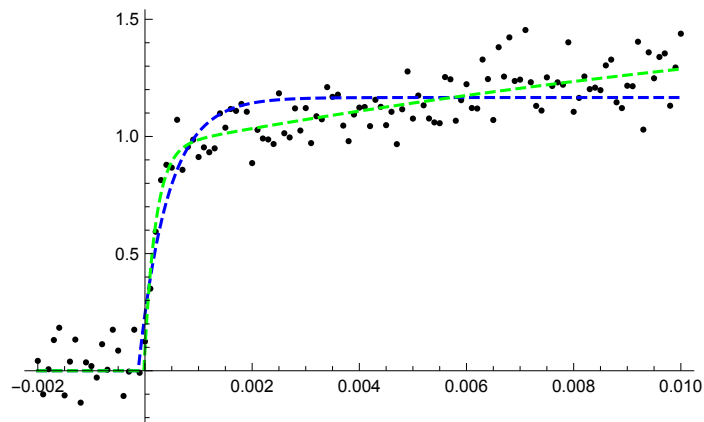
take bi

Mono: chi2 = 1.59125   d = 1.59125   a = 1.16621   t = 0.000546398

Bi: ratio chi2Mono/chi2Bi = 1.57996   d = -0.0000206044   a =
 1.73621   a1 = 0.949411   t1 =0.000210073   t2 = 0.0178237

-- D

take bi

take bi

take bi
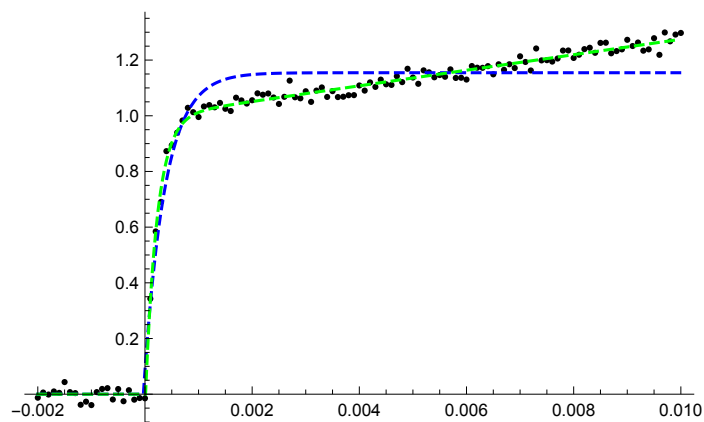
Mono: chi2 = 0.511676   d = 0.511676   a = 1.15458   t = 0.000392733

Bi: ratio chi2Mono/chi2Bi = 10.6068   d = 0.0000110068   a =
 9.97422   a1 = 0.994692   t1 =0.000218414   t2 = 0.315371

----------------   Ca = 30.4385
  uM   ---
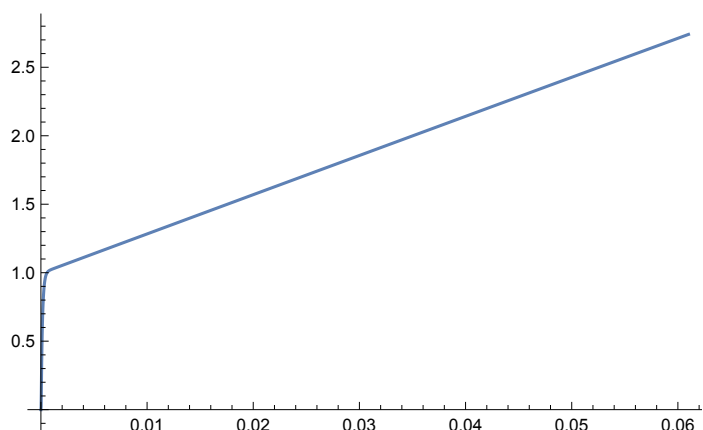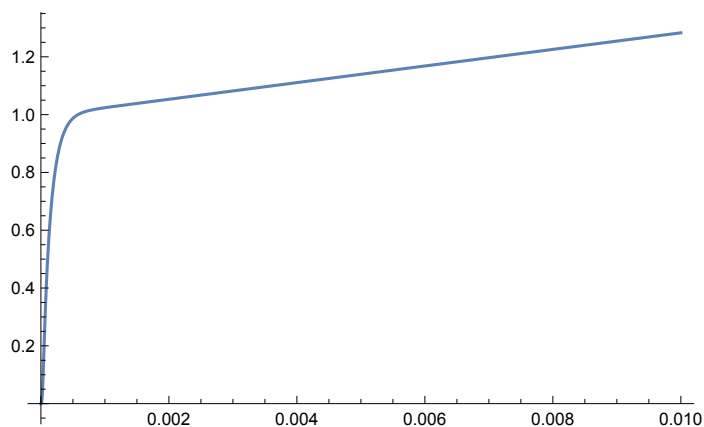
-- C5

take bi

take bi

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

take bi
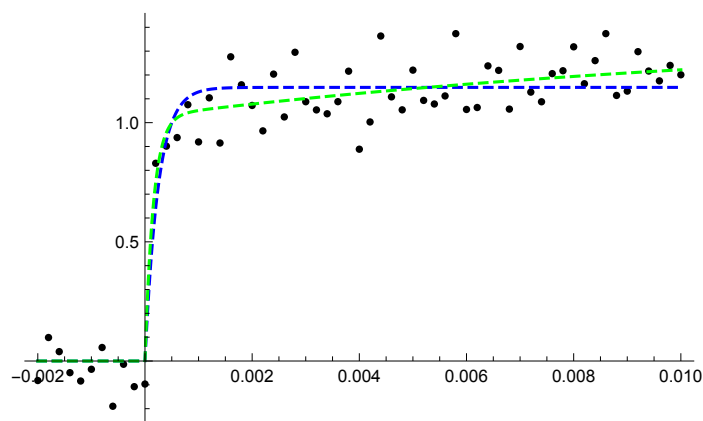
Mono: chi2 = 0.78773   d = 0.78773   a = 1.14792   t = 0.000244255

Bi: ratio chi2Mono/chi2Bi = 1.24482   d = 1.40004×10^{-19} a =
  1.38242   a1 = 1.02496   t1 = 0.000155164   t2 = 0.012503

-- C10

take bi

take bi
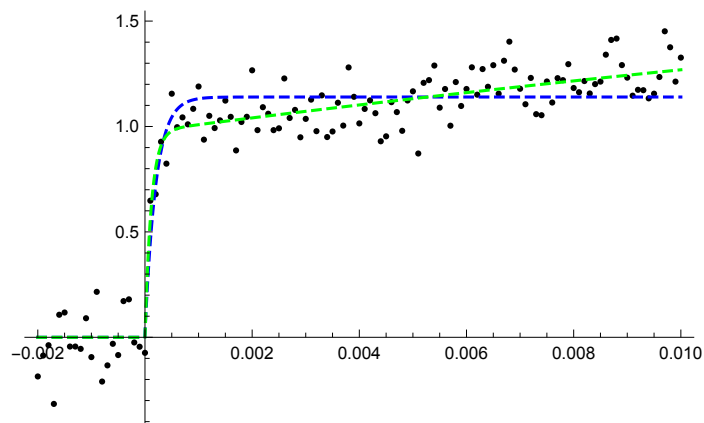
take bi

Mono: chi2 = 1.92084   d = 1.92084   a = 1.13982   t = 0.000203106

Bi: ratio chi2Mono/chi2Bi = 1.4438   d = -5.36729×10^{-18} a =
 2.27386   a1 = 0.975989   t1 =0.000118164   t2 = 0.0390768

-- D

take bi

take bi

take bi
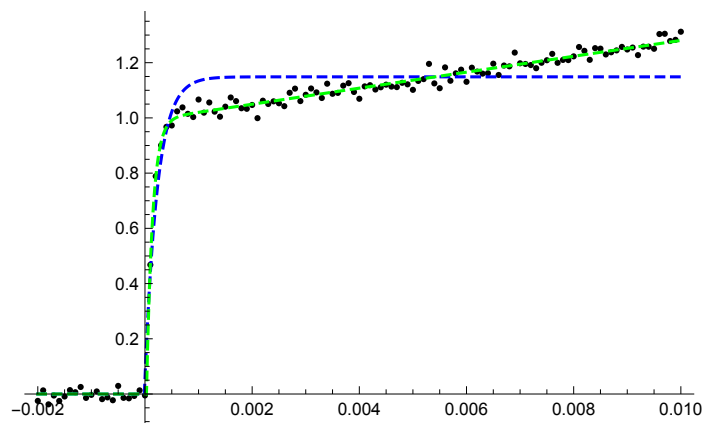
Mono: chi2 = 0.616647   d = 0.616647   a = 1.1487   t = 0.000232058

Bi: ratio chi2Mono/chi2Bi = 13.2869   d = 0.0000283387   a =
 11.7005   a1 = 0.99178   t1 =0.00011251   t2 = 0.364794

----------------   Ca = 74.5861
  uM   ---
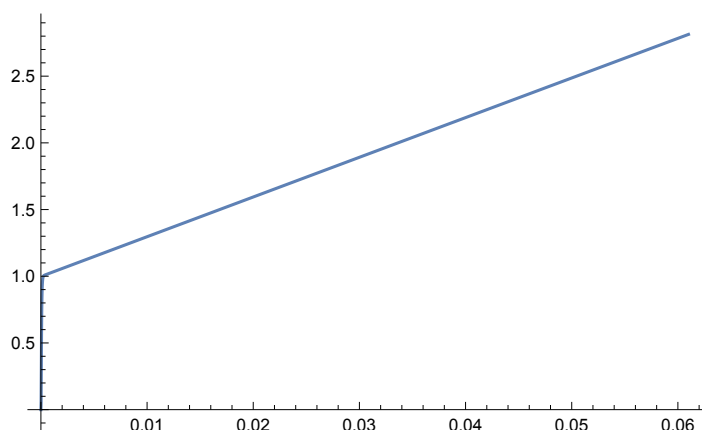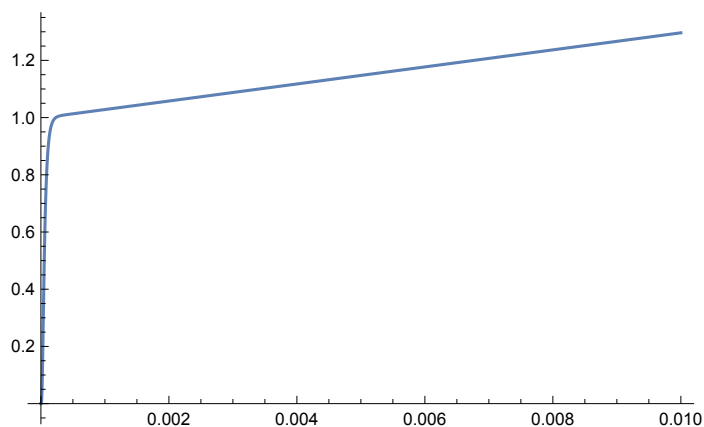
--- C5

take bi

(...) LinearAlgebra`BLAS`TRSV : Machine overflow encountered during computations.

take bi

take mono
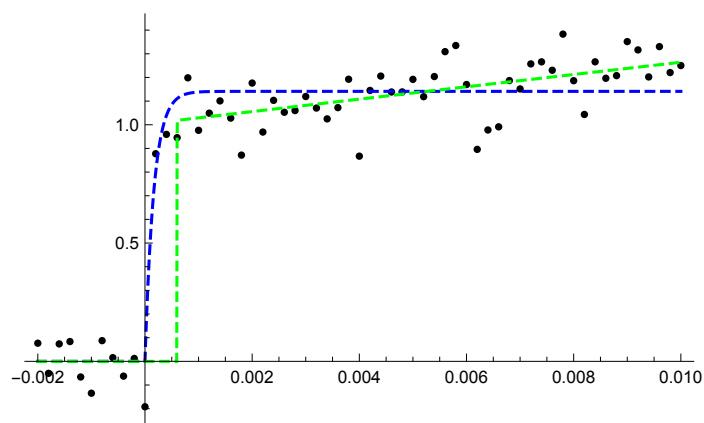
Mono: chi2 = 0.857642   d = 0.857642   a = 1.14114   t = 0.000165993

Bi: ratio chi2Mono/chi2Bi = 0.377459   d = 0.000595189   a =
 9.47387   a1 = 1.01846   t1 =1.82709×10^{-6} t2 = 0.319066

--- C10

take bi

take bi
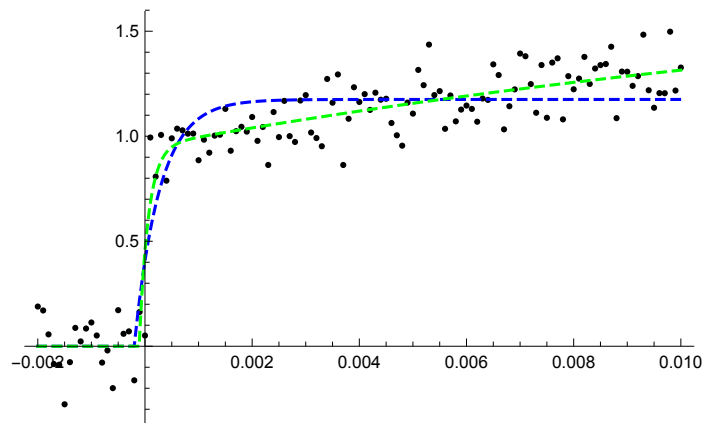
take bi

```
Mono: chi2 = 2.45568      d = 2.45568      a = 1.17507     t = 0.000470653

Bi: ratio chi2Mono/chi2Bi = 1.49993      d = -0.000108103      a =
   1.8332      a1 = 0.945565      t1 =0.000168578      t2 = 0.0188106
```
-- D
```
take bi

take bi

take bi
```
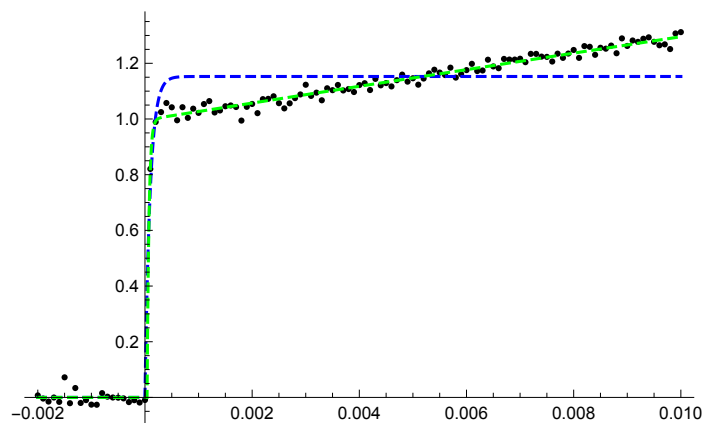


```
Mono: chi2 = 0.73026      d = 0.73026      a = 1.15265     t = 0.0000962743

Bi: ratio chi2Mono/chi2Bi = 17.9606      d = 0.0000393691      a =
   9.18693      a1 = 0.997681      t1 =0.0000352924      t2 = 0.268951
```

Plots

In[329]:= **rrp = 10;**

In[330]:= **caFact = 1*^6;**

In[331]:= **colorA = Green;**
colorB = Red;
colorC = Blue;

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

In[334]:= `simParam = 15;`

C5

In[335]:=
```
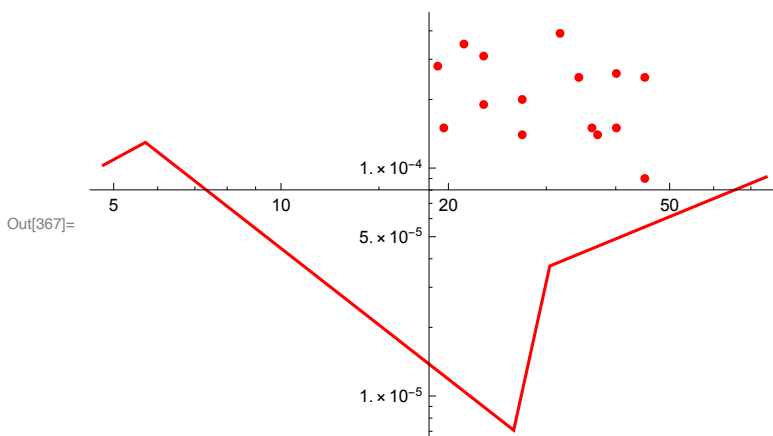gr1a = ListLogLogPlot[
    Transpose[{dataT1C5Ca, dataT1C5RelRate}], PlotStyle → {colorA}];
gr2a = ListLogLogPlot[Transpose[{caFact simCaList,
      simParamMedianC5[[simParam, All]]}],
    PlotStyle → {colorA}, Joined → True, PlotRange → All];
gr3a = ListLogLogPlot[Transpose[{caFact simCaList,
      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[339]=
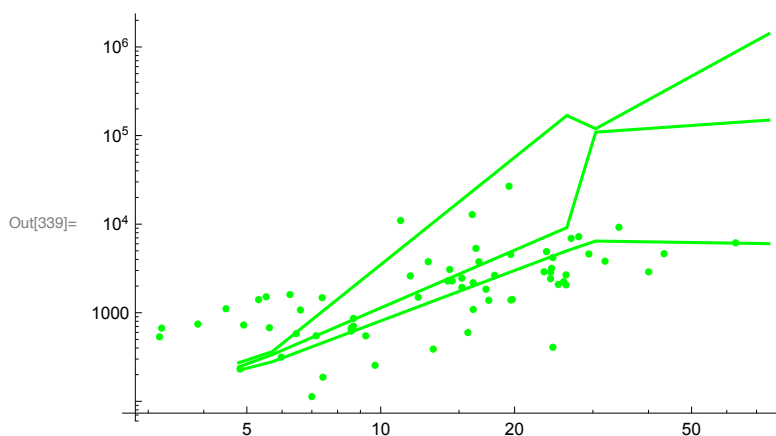
C10

```
In[341]:= 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"];
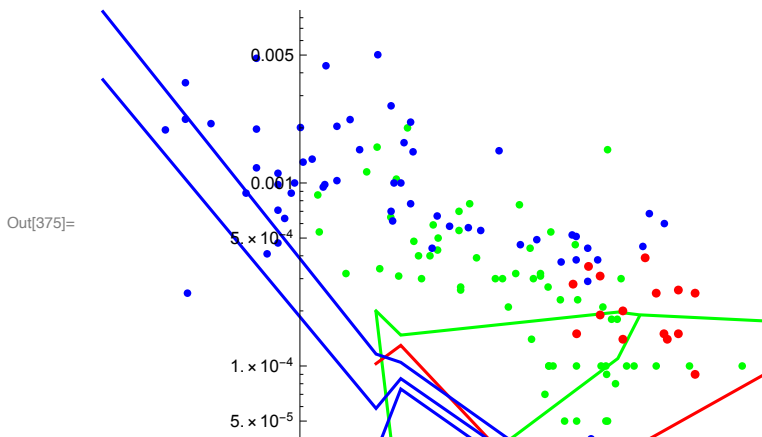     ];
```

Out[345]=
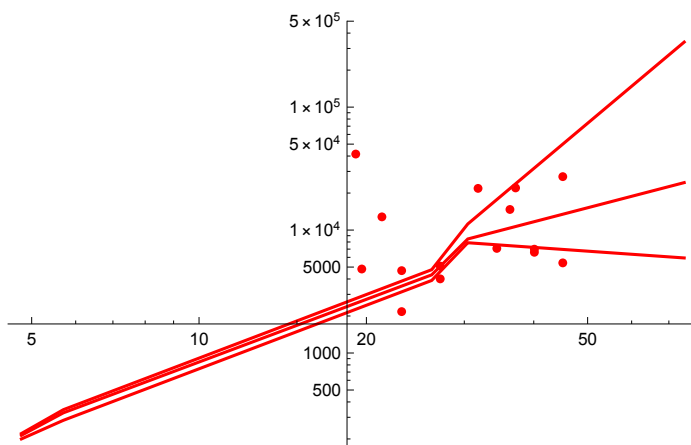
D

```
In[347]:= 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[351]=
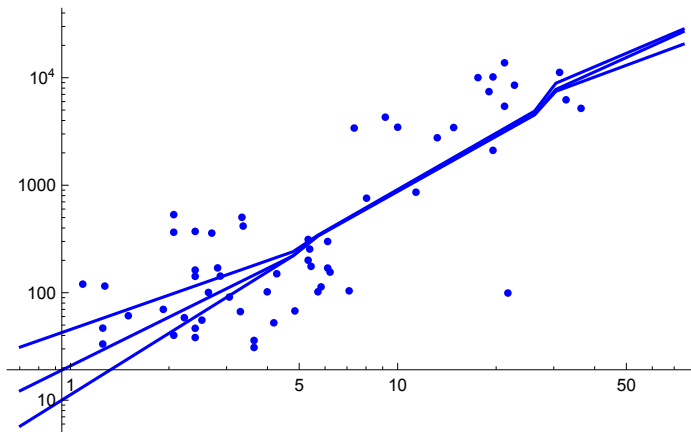
C5 and C10 and D

In[353]:= `Show[gr1a, gr2a, gr3a, gr4a, gr1b, gr2b, gr3b,`
`  gr4b, gr1c, gr2c, gr3c, gr4c, PlotRange → {All, {0, 10}}]`
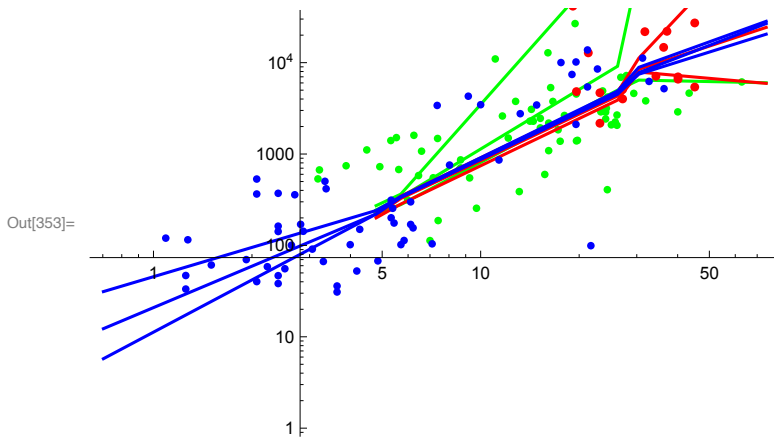
Out[353]=

In[354]:= `Show[gr1a, gr1b, gr1c, PlotRange → {All, {2, 10}}];`

delay (mono and bi merged)

In[355]:= `simParam = 12;`

In[356]:= `Transpose[{caFact simCaList, simParamMedianC5[[simParam, All]]}] // TableForm`

Out[356]//TableForm=

0.703073	
4.79194	0.000196264
5.70182	0.0000141801
26.2105	0.000109866
30.4385	0.000185505
74.5861	-3.22945×10^{-18}

C5

```
In[357]:= 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"];
     ];
```
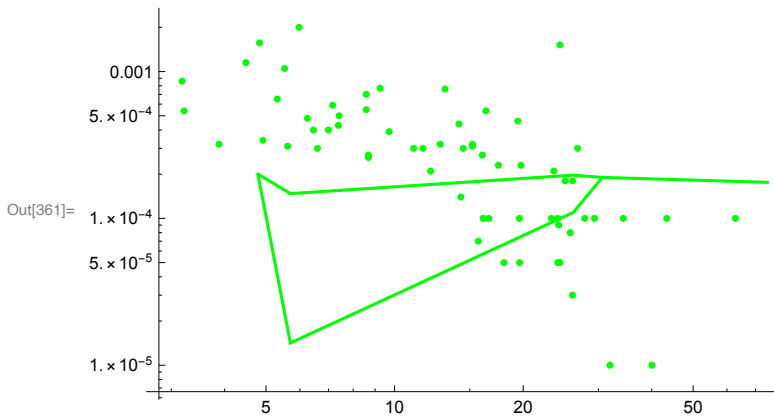
C10

```
In[363]:= 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"];
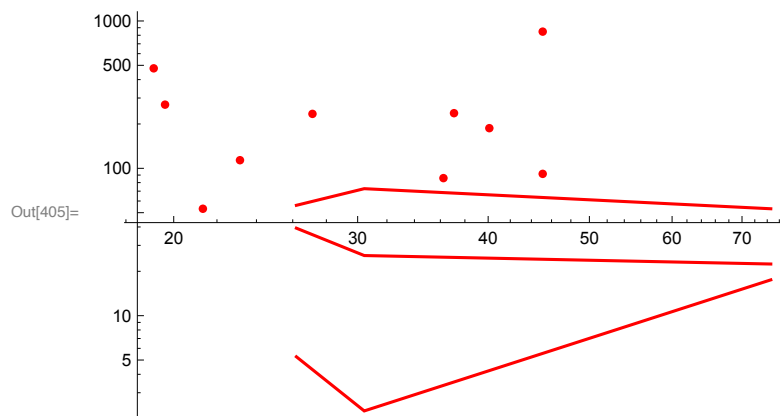      ];
```

D

In[369]:=
```
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[373]=
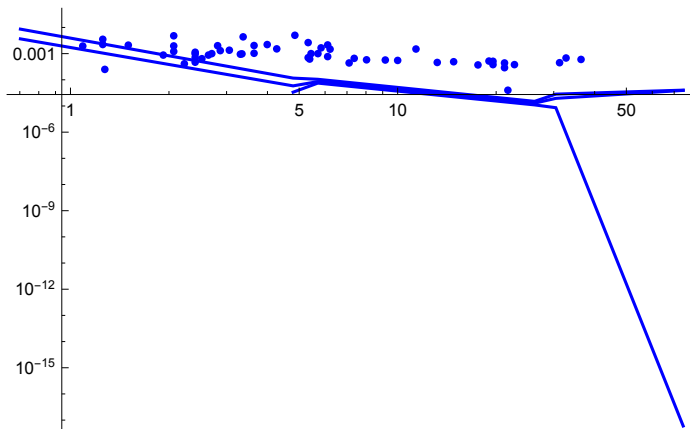
C5 and C10 and D

In[375]:= `Show[gr1a, gr2a, gr3a, gr4a, gr1b, gr2b, gr3b,`
`gr4b, gr1c, gr2c, gr3c, gr4c, PlotRange → {-10, -5}]`

Out[375]=

In[376]:= `Show[gr1a, gr1b, gr1c, PlotRange → All];`

amp (merge of mono amp and bi amp)

In[377]:= `simParam = 13;`

C5

In[378]:=
```
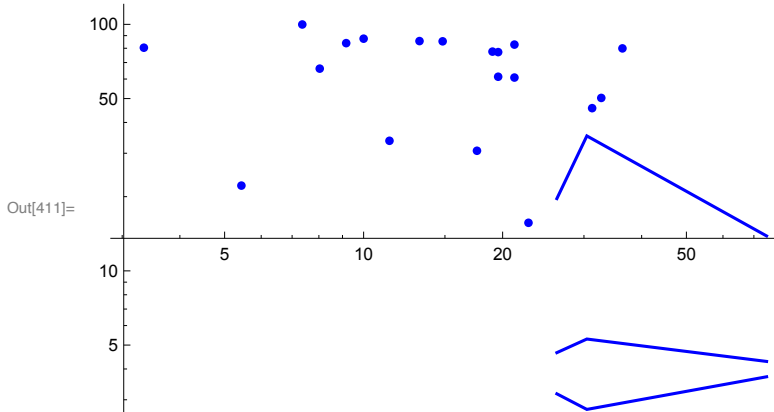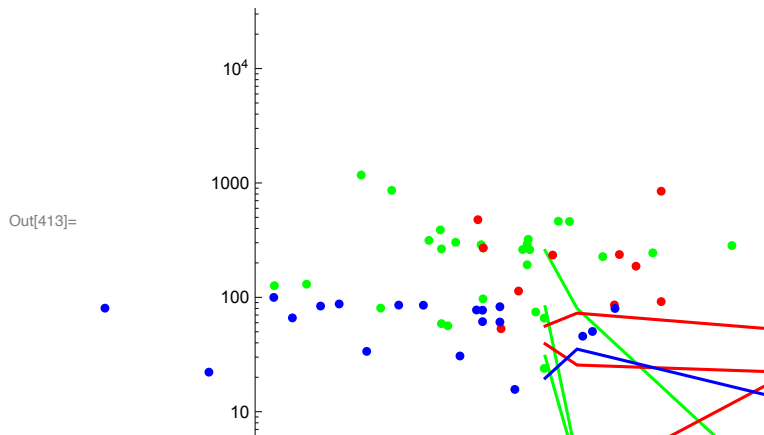gr1a = ListLogLogPlot[
    Transpose[{dataT1C5Ca, dataT1C5Amplitude}], PlotStyle → {colorA}];
gr2a = ListLogLogPlot[Transpose[{caFact simCaList,
      rrp simParamMedianC5[[simParam, All]]}],
    PlotStyle → {colorA}, Joined → True, PlotRange → All];
gr3a = ListLogLogPlot[Transpose[{caFact simCaList,
      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[382]=
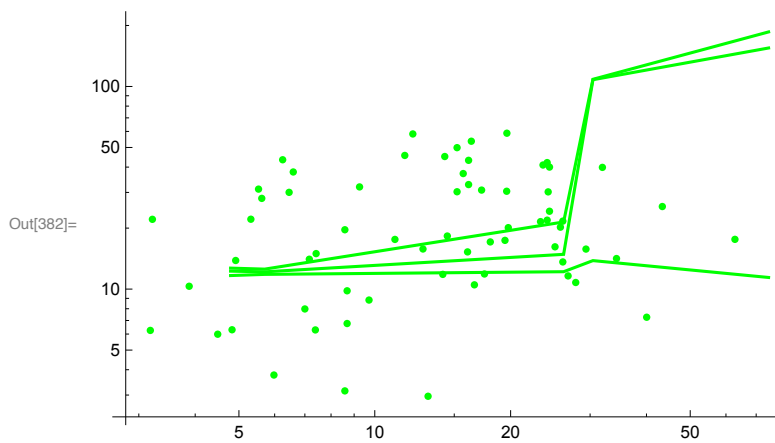
C10

In[383]:=
```
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[387]=
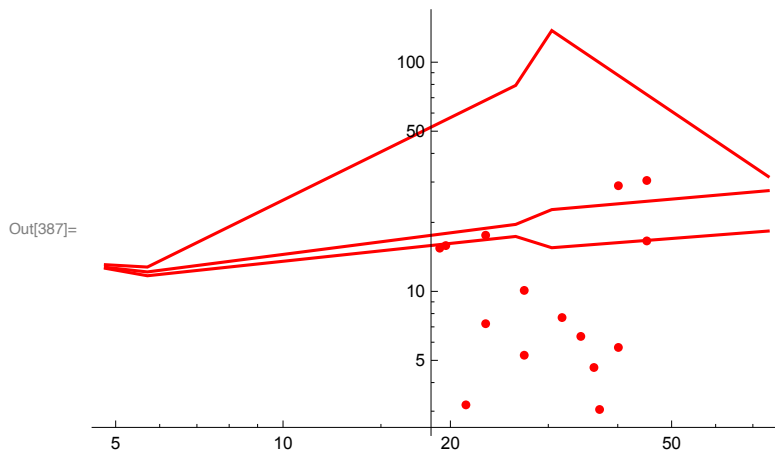
D

```
In[388]:= 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[392]=
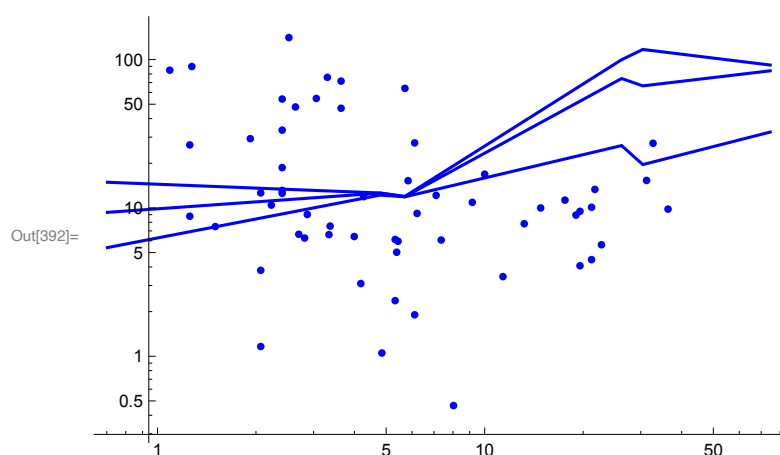
C5 and C10 and D

```
In[393]:= Show[gr1a, gr2a, gr3a, gr4a, gr1b, gr2b, gr3b,
         gr4b, gr1c, gr2c, gr3c, gr4c, PlotRange → {-1, 6} ]
```

Out[393]=

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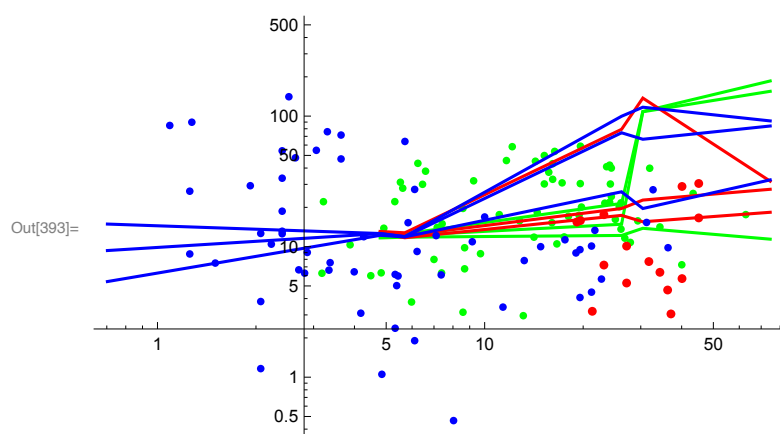
release rate 1/tau2 of bi fits (if bi is justified)

In[394]:= `simParam = 16;`

C5

In[395]:=
```
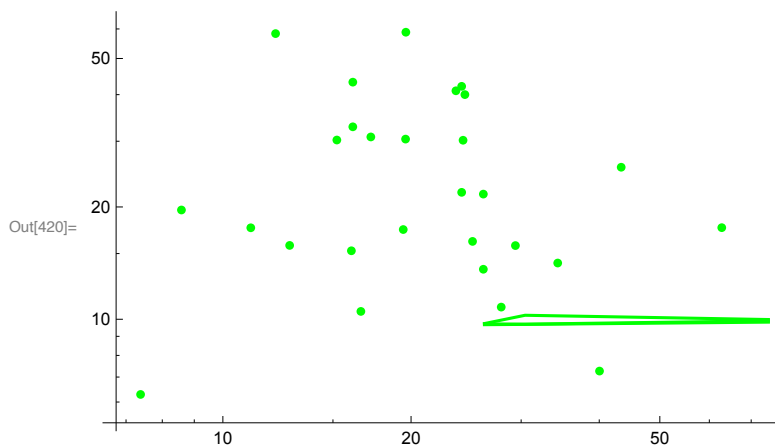gr1a = ListLogLogPlot[
    Transpose[{dataT2C5Ca, dataT2C5RelRate}], PlotStyle → {colorA}];
  gr2a = ListLogLogPlot[Transpose[{caFact simCaList,
      simParamMedianC5[[simParam, All]]}],
    PlotStyle → {colorA}, Joined → True, PlotRange → All];
  gr3a = ListLogLogPlot[Transpose[{caFact simCaList,
      simParamQuantile1C5[[simParam, All]]}],
    PlotStyle → {colorA}, Joined → True, PlotRange → All];
  gr4a = ListLogLogPlot[Transpose[{caFact simCaList,
      simParamQuantile2C5[[simParam, All]]}],
    PlotStyle → {colorA}, Joined → True, PlotRange → All];
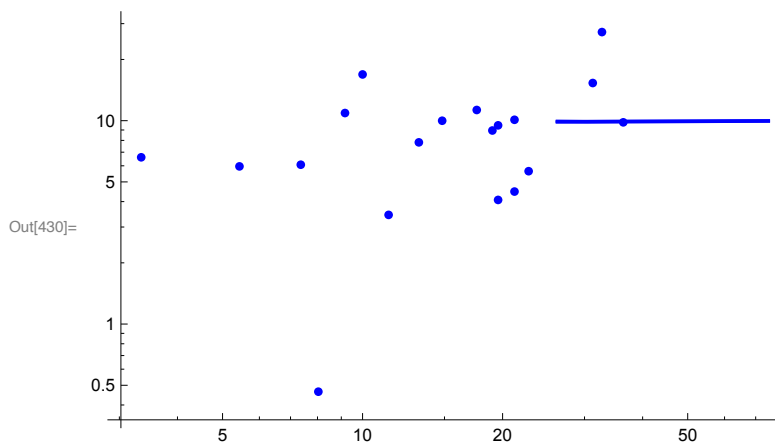  Show[gr1a, gr2a, gr3a, gr4a, PlotRange → All ]
  If[exportYes == 1,
    Export["plot InvTau2 C5 data.txt",
     Transpose[{dataT2C5Ca, dataT2C5RelRate}], "Table"];
    toExport = Transpose[{caFact simCaList, simParamQuantile1C5[[simParam, All]],
       simParamMedianC5[[simParam, All]], simParamQuantile2C5[[simParam, All]]}];
    Export["plot InvTau2 C5 fit - quantiles and median.txt", toExport, "Table"];
   ];
```

Out[399]=
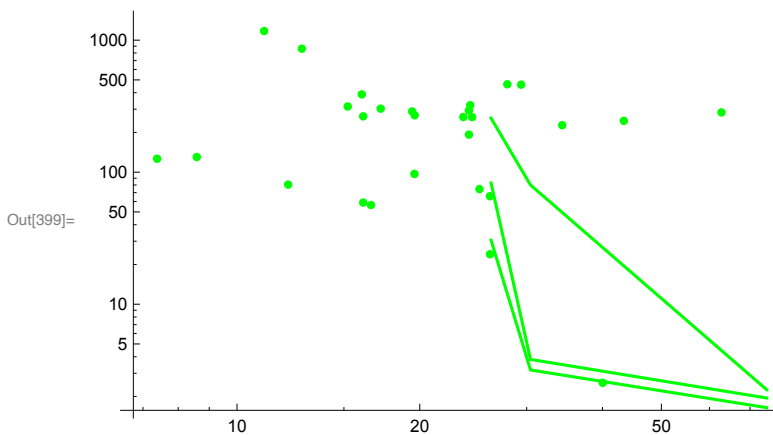
C10

```
In[401]:= 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"];
     ];
```

```
In[407]:= 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[411]=

C5 and C10 and D

In[413]:= `Show[gr1a, gr2a, gr3a, gr4a, gr1b, gr2b, gr3b,`
`    gr4b, gr1c, gr2c, gr3c, gr4c, PlotRange → {All, {2, 10}}]`

Out[413]=

In[414]:= `Show[gr1a, gr1b, gr1c, PlotRange → All];`

amp1 of bi fits (if bi is justified)

In[415]:= `simParam = 14;`

C5

```
In[416]:= 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[420]=

C10

```
In[421]:= 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 ]
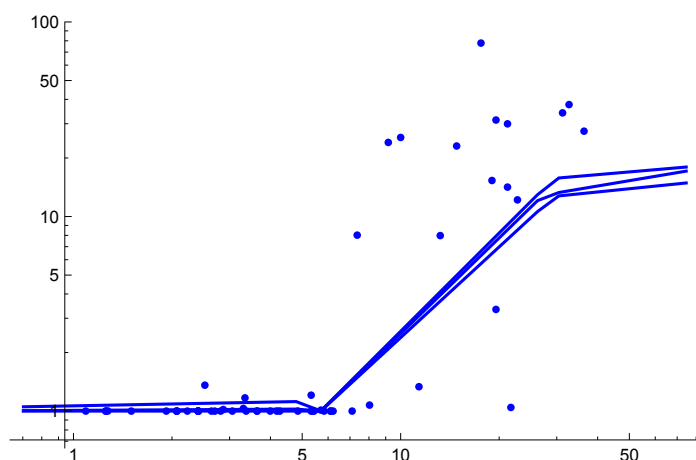```

Out[425]=
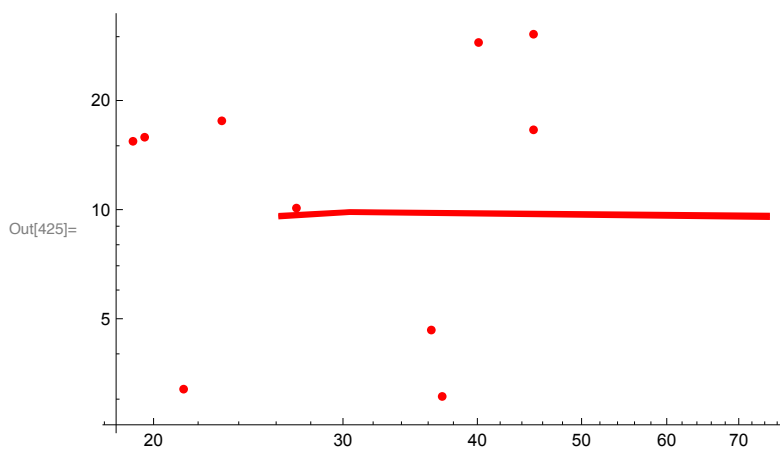
## D

```
In[426]:= 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[430]=

C5 and C10 and D

```
In[431]:= Show[gr1a, gr2a, gr3a, gr4a, gr1b, gr2b,
       gr3b, gr4b, gr1c, gr2c, gr3c, gr4c, PlotRange → All]
    Show[gr1a, gr1b, gr1c, PlotRange → All];
```


Out[431]=
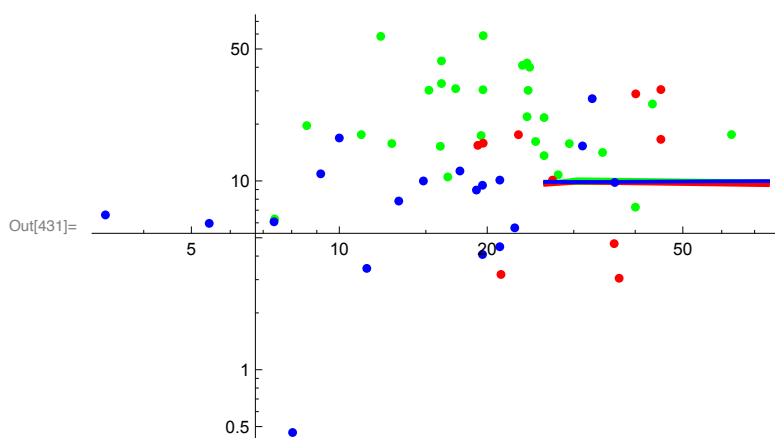
chi2 mono/bi ratio

In[433]:= `simParam = 6;`

C5

In[434]:=
```
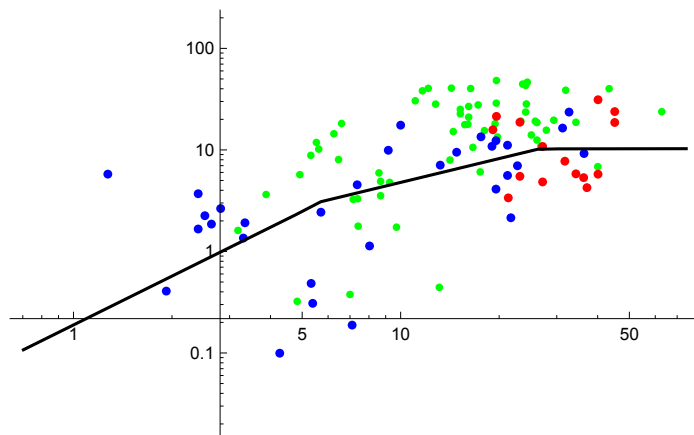gr1a = ListLogLogPlot[
    Transpose[{dataT1C5Ca, dataT1C5ChiRatio}], PlotStyle → {colorA}];
gr2a = ListLogLogPlot[Transpose[{caFact simCaList,
    simParamMedianC5[[simParam, All]]}],
  PlotStyle → {colorA}, Joined → True, PlotRange → All];
gr3a = ListLogLogPlot[Transpose[{caFact simCaList,
    simParamQuantile1C5[[simParam, 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[438]=
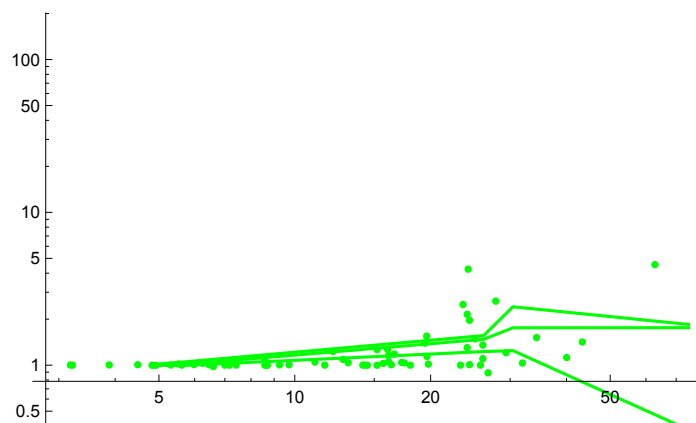
C10

```
In[440]:= 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[444]=
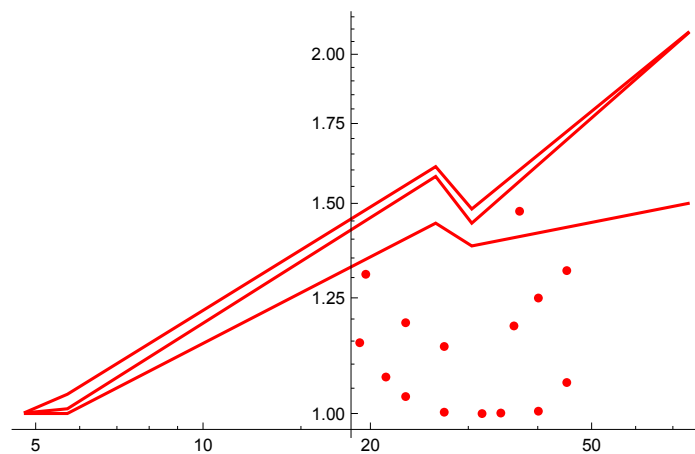
D

```
In[446]:= 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[450]=

C5 and C10 and D

```
In[452]:= 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[452]=
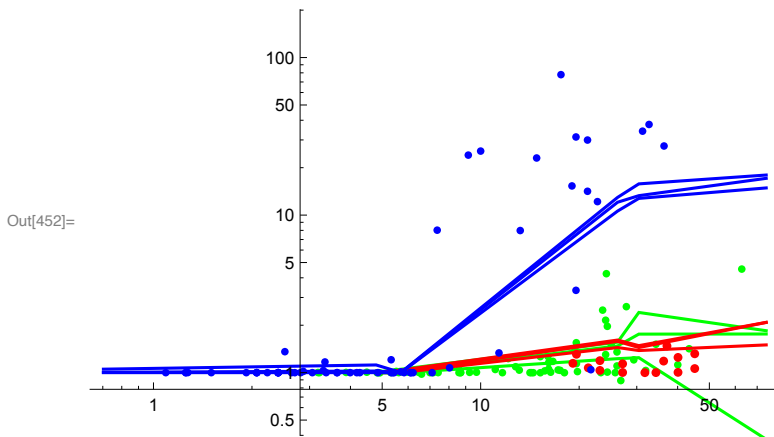
Nv

```
In[454]:= 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
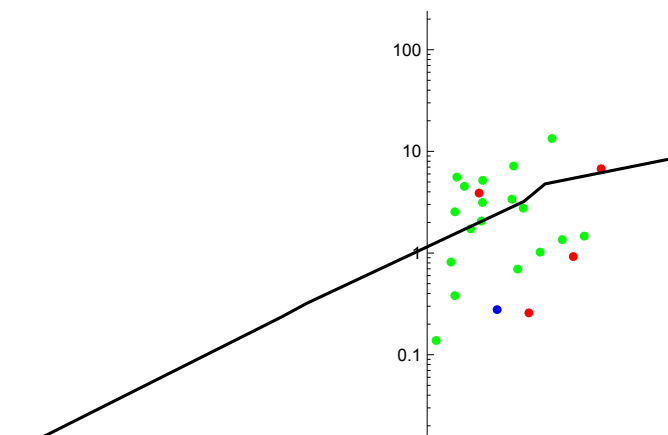
time for Nv (ms) = 0.2
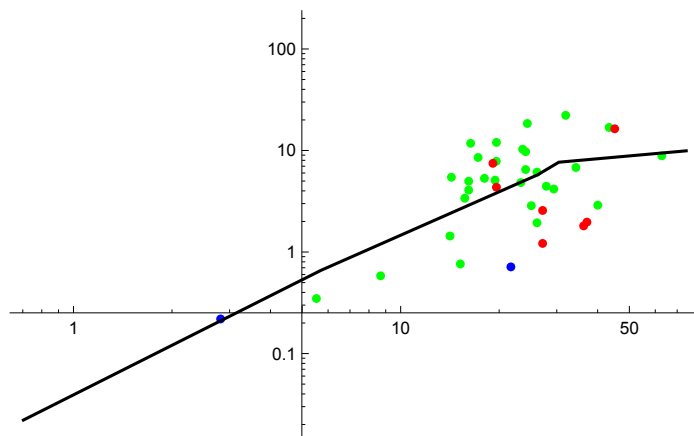
time for Nv (ms) = 1.

time for Nv (ms) = 5.
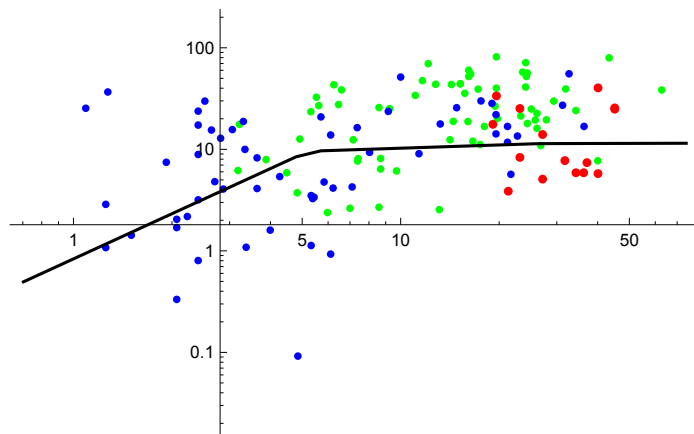
time for Nv (ms) = 10.
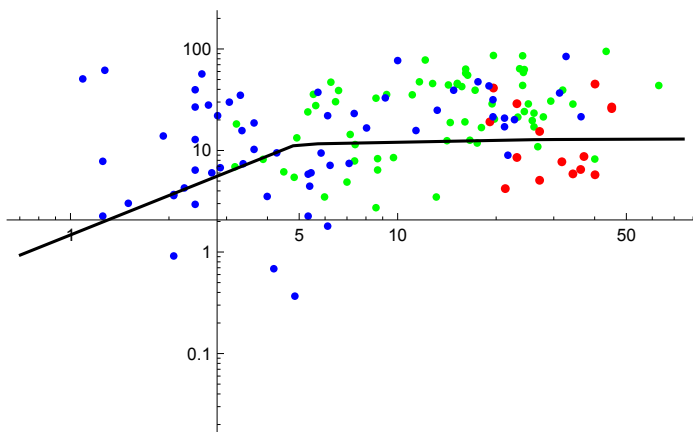
time for Nv (ms) = 100.
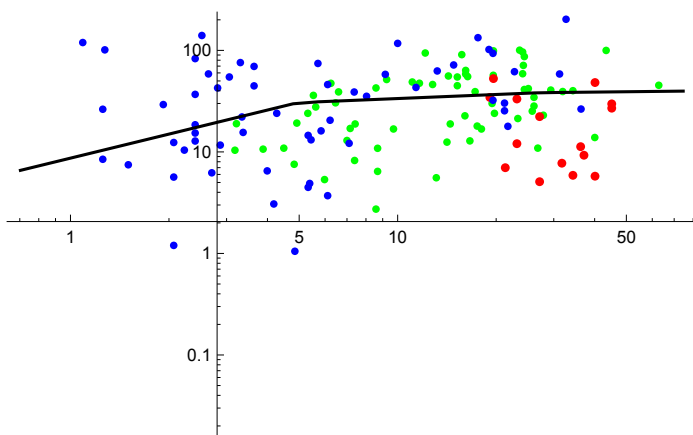
time for Nv (ms) = 400.
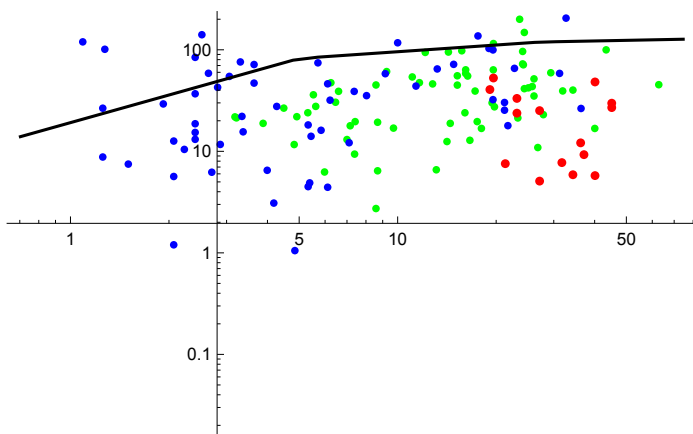
sustained release 10 to 100 ms

```
In[455]:= 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;
```
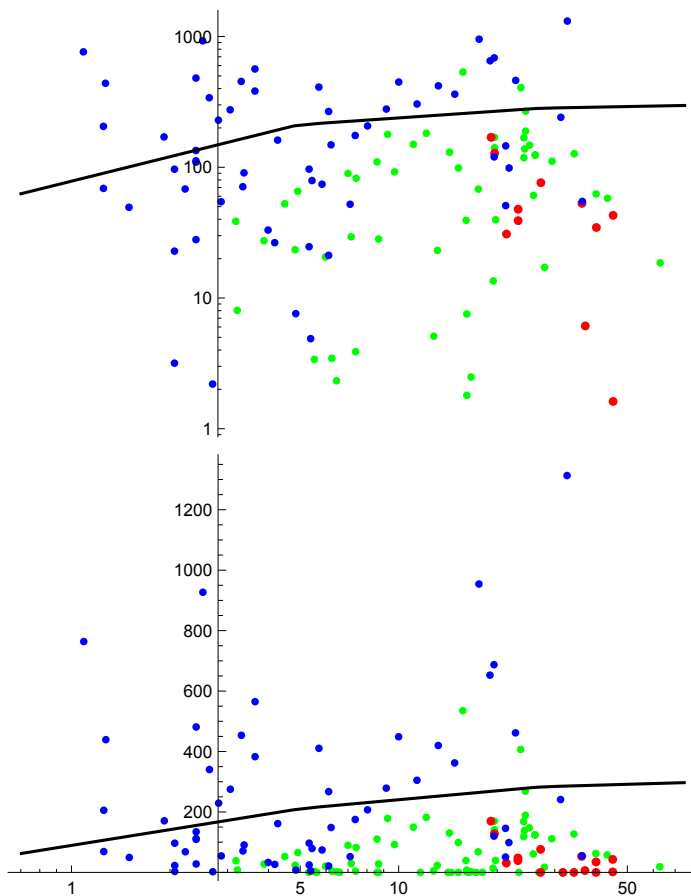
```
In[463]:= If[exportYes == 1,
    Export["plot sustained release Cm5 data.txt", ttt1, "Table"];
    Export["plot sustained release Cm10 data.txt", ttt2, "Table"];
    Export["plot sustained release D data.txt", ttt3, "Table"];
    Export["plot sustained release sim.txt", ttt4, "Table"]
    ];
```

Export Nv

```
In[464]:= 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[465]:= Transpose[simParamMedianC5] // TableForm
    Transpose[simParamQuantile1C5] // TableForm
    Transpose[simParamQuantile2C5] // TableForm
```

Out[465]//TableForm=

0	0.558169	0.000196264	1.22645	244.494	1.00456	0.0002	
0	0.436364	0.0000141801	1.21661	336.177	1.01267	-0.0001223	
0	0.694914	-7.11367×10^{-6}	1.15802	2698.21	1.47693	0.000109866	
0	0.826826	-7.05377×10^{-6}	1.14792	4094.08	1.75627	0.000185505	
0	0.857642	-1.81482×10^{-8}	1.14304	15 567.3	1.75924	0.000176046	

Out[466]//TableForm=

0	0.535	0.000108148	1.16632	223.507	0.996897	0.0002
0	0.419664	-0.000121843	1.182	279.64	0.996843	-0.000161091
0	0.650755	-0.0000539375	1.14245	2202.31	1.22686	-2.19493×10^{-18}
0	0.78773	-0.0000123543	1.12809	3744.32	1.24482	1.40004×10^{-19}
0	0.795886	-4.7488×10^{-6}	1.14114	6024.35	0.377459	-5.07355×10^{-7}

Out[467]//TableForm=

0	0.577463	0.0002	1.26918	272.268	1.00805	0.00020429
0	0.599512	0.000147517	1.2548	363.926	1.05093	0.00020205
0	0.816841	-4.66012×10^{-7}	1.15808	3182.47	1.56176	0.00019701
0	0.853025	1.04004×10^{-18}	1.16428	6138.7	2.41035	0.00019039
0	0.928911	-3.22945×10^{-18}	1.14543	3.63275×10^6	1.84598	0.00059518

C10

In[468]:= **Transpose[simParamMedianC10] // TableForm**
Transpose[simParamQuantile1C10] // TableForm
Transpose[simParamQuantile2C10] // TableForm

Out[468]//TableForm=

0	1.11476	0.0000484428	1.27367	214.143	1.00129	0.0000261636
0	1.02845	-8.80427×10^{-6}	1.21575	326.217	1.00915	-8.80427×10^{-6}
0	1.70257	-0.000126633	1.17178	1830.17	1.57996	-0.0000206044
0	1.79213	-3.03073×10^{-6}	1.14108	4923.54	1.4438	-5.36729×10^{-18}
0	2.26987	-0.00011754	1.1626	4002.43	2.08316	0.0000191632

Out[469]//TableForm=

0	1.05691	0.0000261636	1.261	200.008	1.	0.0000239968
0	0.947635	-9.69546×10^{-6}	1.16986	283.537	1.	-0.0000802886
0	1.59125	-0.000127784	1.16621	1612.49	1.44382	-0.0000484516
0	1.77504	-0.0000610271	1.13982	3087.88	1.38169	-3.66201×10^{-6}
0	1.93877	-0.0002	1.15265	2124.71	1.49993	-0.000108103

Out[470]//TableForm=

0	1.20884	0.000102995	1.30731	221.396	1.00234	0.0001
0	1.2785	0.000129601	1.27679	345.972	1.03789	0.0003
0	1.7205	-0.000103626	1.17955	1838.17	1.61024	7.07898×10^{-6}
0	1.92084	-5.07498×10^{-18}	1.17297	5574.88	1.48387	0.0000371808
0	2.45568	2.66303×10^{-18}	1.17507	10066.6	2.08562	0.0000913826

D

In[471]:= **Transpose[simParamMedianD] // TableForm**
Transpose[simParamQuantile1D] // TableForm
Transpose[simParamQuantile2D] // TableForm

Out[471]//TableForm=

0	1.07522	0.00366323	0.933623	12.2735	1.00524	-0.003
0	0.0470871	0.0000586294	1.2585	220.807	1.02416	0.0000301442
0	0.0444231	0.000085284	1.19006	338.456	1.00026	0.0000854186
0	0.535879	-0.0000325983	1.15672	2512.74	12.0705	0.0000128142
0	0.613988	-0.0000163065	1.15097	3871.97	13.2869	0.0000195498
0	0.73026	7.37316×10^{-18}	1.15391	10387.	17.1133	0.0000393691

Out[472]//TableForm=

0	1.03411	-0.00337928	0.54024	5.75488	1.0002	-0.028
0	0.0454326	0.0000336663	1.2204	220.377	1.00026	0.0000272162
0	0.0389857	0.0000748362	1.18797	336.26	1.	0.0000742133
0	0.511676	-0.0000426969	1.15458	2304.35	10.6068	0.0000110068
0	0.580384	-0.0000273383	1.1487	3605.69	12.765	8.60548×10^{-6}
0	0.703632	-7.03806×10^{-7}	1.15265	9412.22	14.8753	5.9678×10^{-18}

Out[473]//TableForm=

0	1.12516	0.00862974	1.48961	31.2373	1.05217	0.0110829
0	0.0558425	0.00011618	1.26081	241.017	1.11916	0.0000628942
0	0.0467604	0.000104663	1.19511	342.674	1.00223	0.0000984108
0	0.548283	-0.000029835	1.15825	2546.26	12.9511	0.0000150273
0	0.616647	-0.0000111719	1.15155	4309.27	15.7814	0.0000283387
0	0.784949	1.12156×10^{-17}	1.15585	11280.4	17.9606	0.0000402481

Nv

In[474]:= `Transpose[simParamNv] // TableForm`

Out[474]//TableForm=

0.00105711	0.00221298	0.0107502	0.0495193	0.093503	0.658054	1.394
0.0236597	0.0497429	0.230838	0.844272	1.11259	2.97902	7.902
0.0320447	0.0659015	0.308801	0.966234	1.16406	3.11308	8.458
0.32144	0.575694	1.01457	1.13656	1.27865	3.81241	11.83
0.478817	0.766558	1.02386	1.13949	1.28302	3.8455	12.06
0.852609	0.992174	1.02811	1.14733	1.29634	3.9692	12.76

EPSC with different caRest

Interpolate

In[475]:=
```
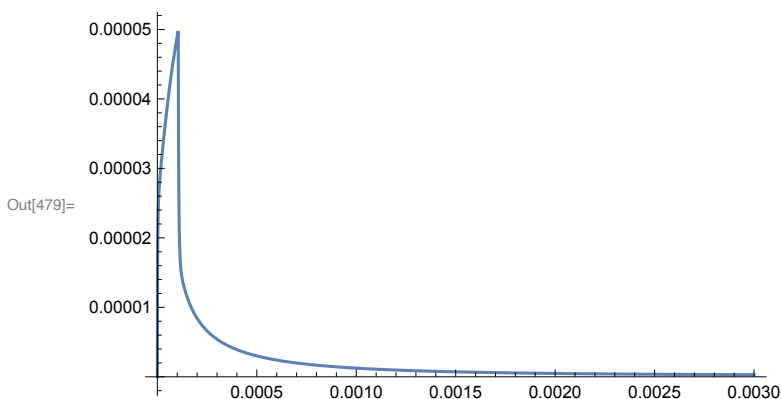locaCa = Transpose[{dataLocalCaTime, dataLocalCa}];
locaCaWithoutdublictes = Mean /@ GatherBy[locaCa, First];
interpolFunc = Interpolation[locaCaWithoutdublictes, InterpolationOrder → 1];
caFunc[t_] := interpolFunc[t];
Plot[caFunc[t], {t, 0.00, 0.003}, PlotRange → All]
```

Out[479]=

NDSolve

```
In[480]:= timeStartForPLot = 0.0;
timeEndForPLot = 0.003;
myNDSolveResults = NDSolve[eq, {ss1, ss2, ss3, ss4}, {t, 0, 0.003}];
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),
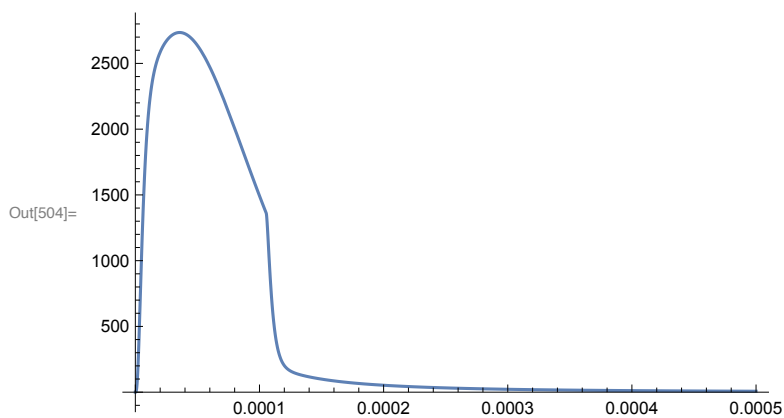  {t, timeStartForPLot, timeEndForPLot}, PlotRange → All]
```

Out[483]=

Out[484]=

Out[485]=

Out[486]=
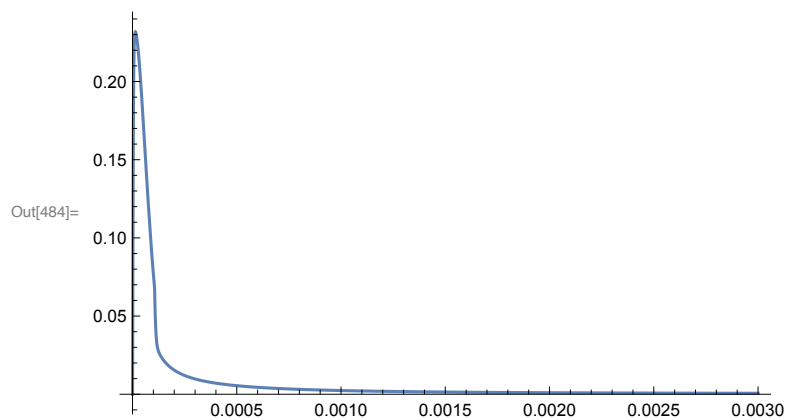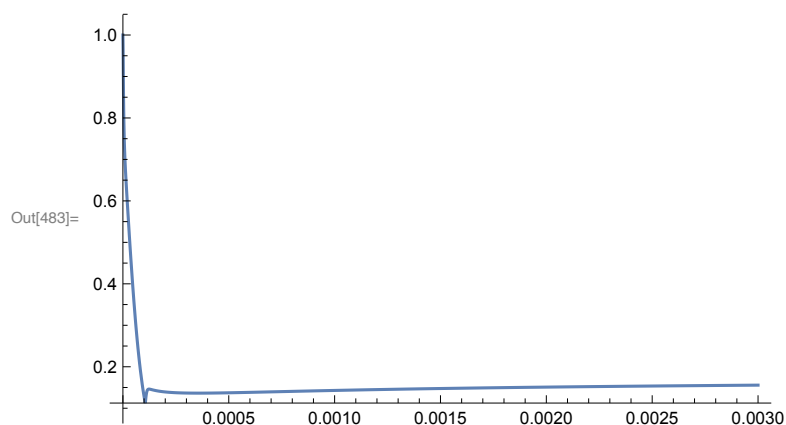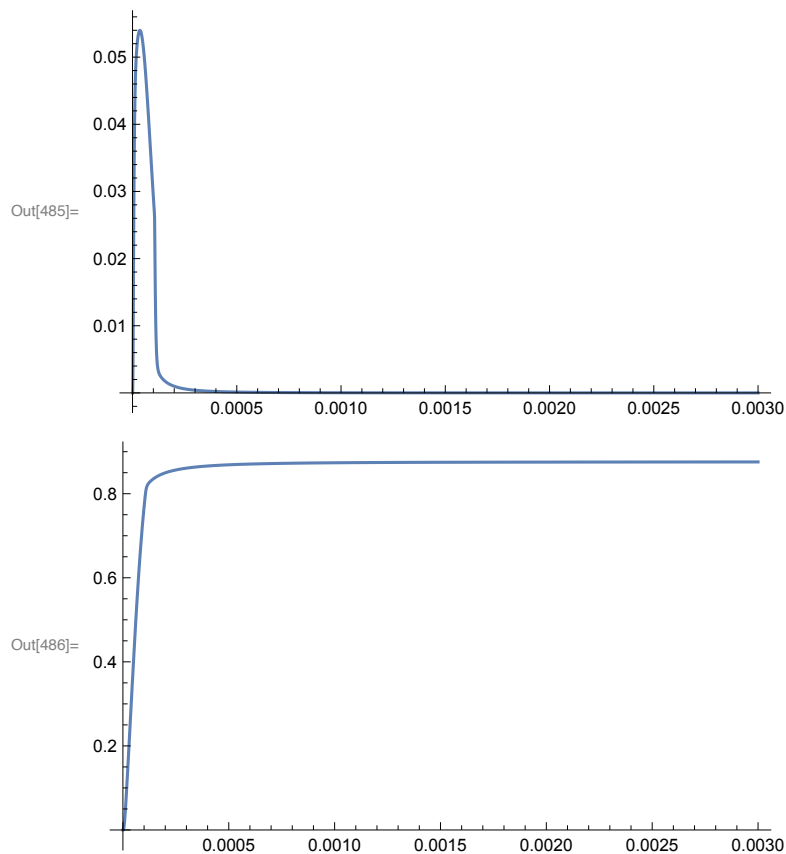
different caRest

In[487]:= **caRestLow = 30*^-9;**
caRestHigh = 180*^-9;

Low Ca

Initial occupancy

In[489]:= **(*calualte initial equilibrium occupancy*)**
caFunc[t_] := caRestLow;
kprimScheme
kunprimScheme
ss0Initial = kprimScheme / kunprimScheme

Out[490]= 1.04335

Out[491]= 3.65793

Out[492]= 0.28523

Diff eq.

```
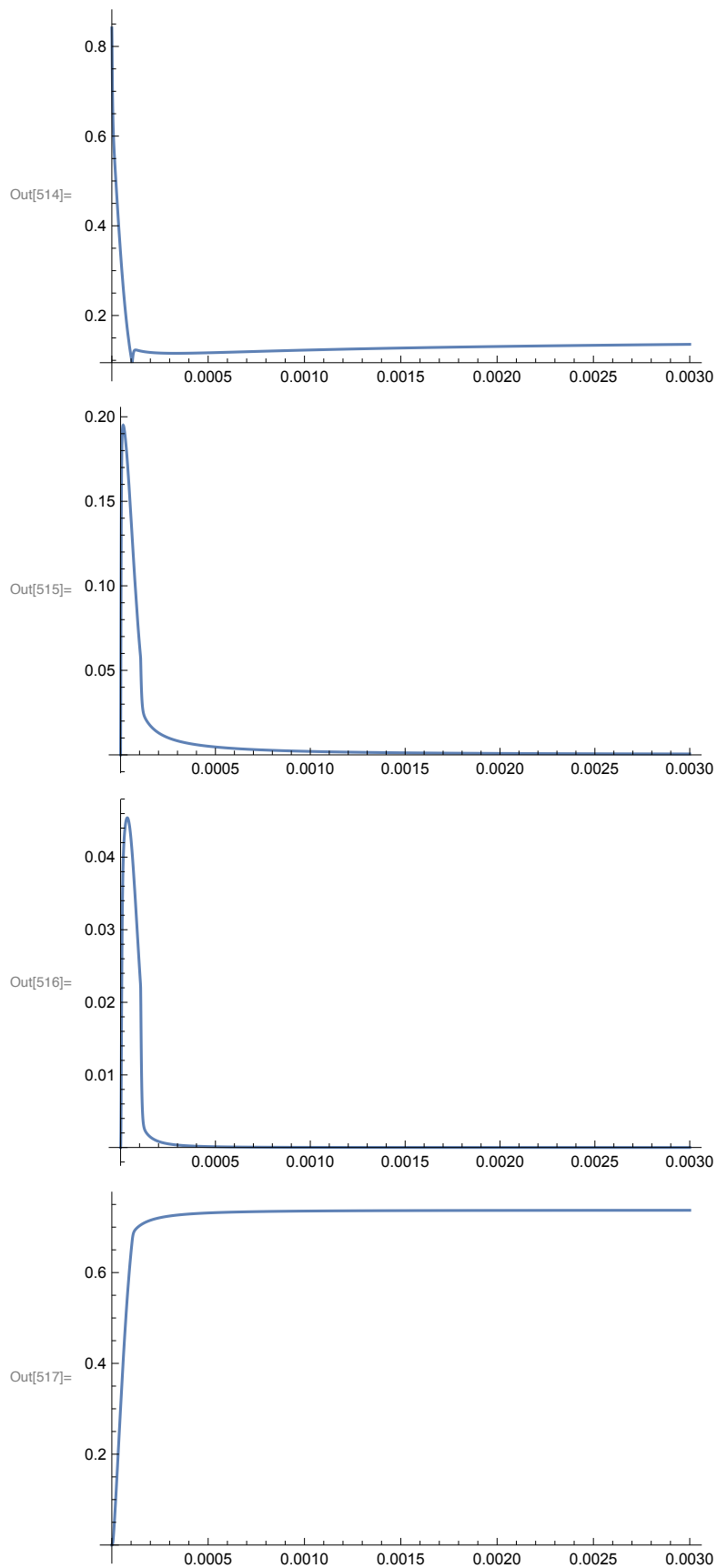In[493]:= Clear[caFunc, eq];(*Clear is needed if the cell is exectued for a 2nd time
      when caFunc is already set to a value or an Interpolationfunction*)
      caFunc[t_] := interpolFunc[t];
      ss[t_] = {ss1[t], ss2[t], ss3[t], ss4[t]};
      eq = {ss'[t] == (mat /. repl).ss[t],
         ss[0] == {ss0Initial, 0, 0, 0}};
```

NDSolve

```
In[497]:= myNDSolveResults = NDSolve[eq, {ss1, ss2, ss3, ss4}, {t, 0, 0.003}];
      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]
```

Out[498]=

Out[499]=

Out[500]=

Out[501]=
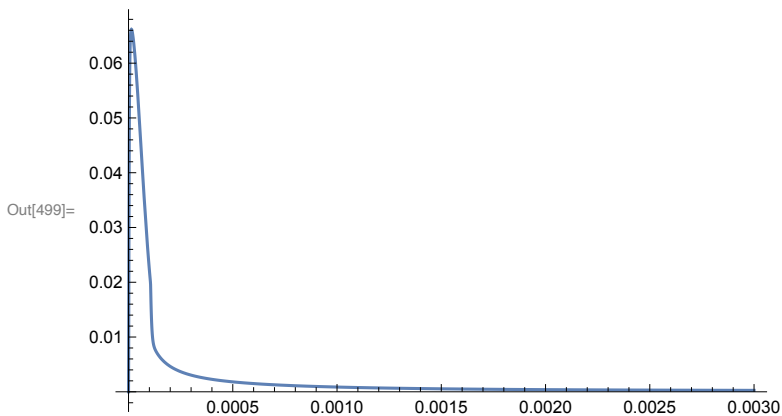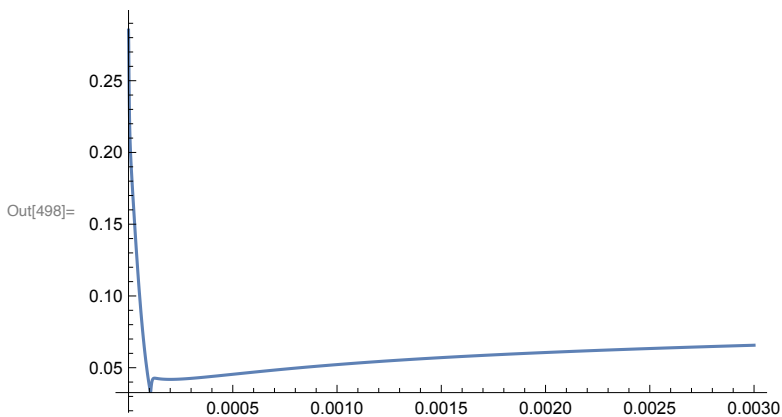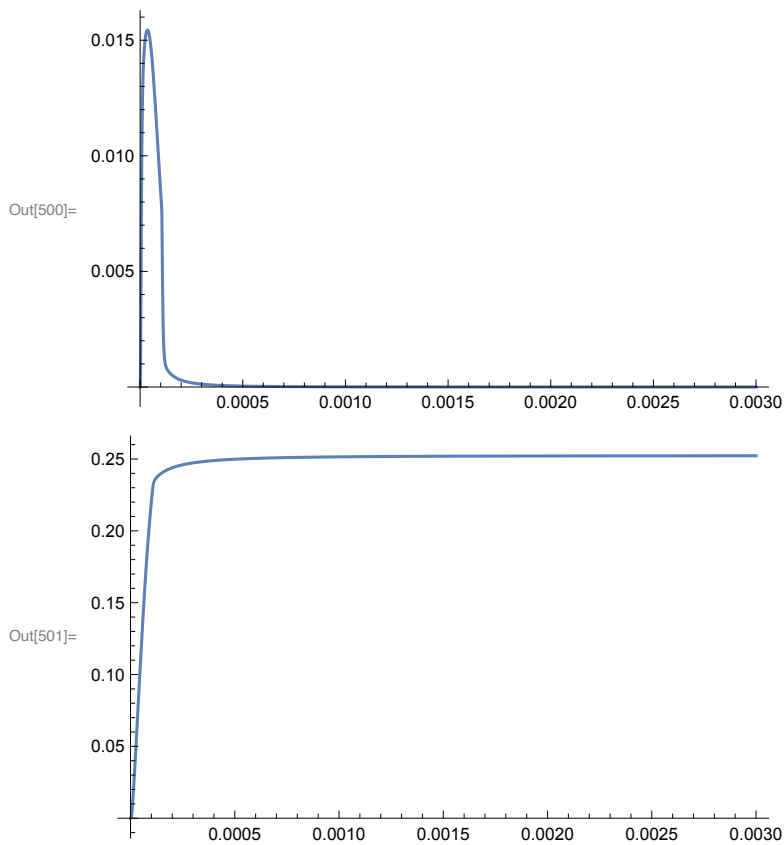
Plot EPSC

In[502]:= ```
epscLowCa = D[(ss4[t] /. myNDSolveResults), t];
Plot[(ss4[t] /. myNDSolveResults), {t, 0, 2*^-3}, PlotRange → All];
Plot[epscLowCa, {t, 0, 0.5*^-3}, PlotRange → All]
```

Out[504]=

# High Ca

## Initial occupancy

```
In[505]:= (*calualte initial equilibrium occupancy*)
      caFunc[t_] := caRestHigh;
      kprimScheme
      kunprimScheme
      ss0Initial = kprimScheme / kunprimScheme
```

Out[506]= 3.07706

Out[507]= 3.65793

Out[508]= 0.841205

Diff eq.

```
In[509]:= 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]};
      eq = {ss'[t] == (mat /. repl).ss[t],
         ss[0] == {ss0Initial, 0, 0, 0}};
```

NDSolve

```
In[513]:= myNDSolveResults = NDSolve[eq, {ss1, ss2, ss3, ss4}, {t, 0, 0.003}];
      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]
```
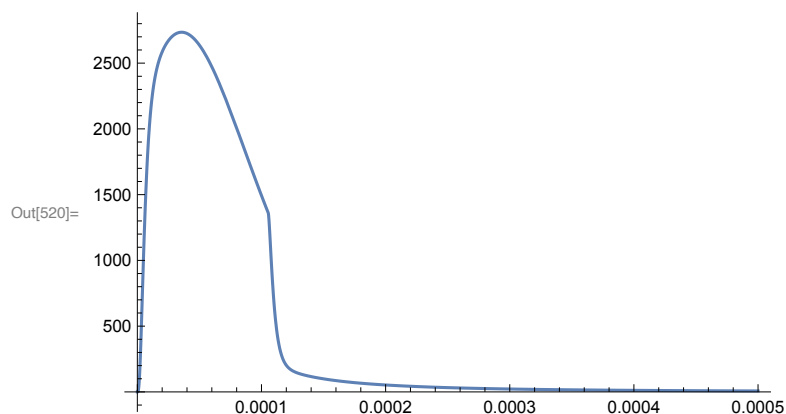
Out[514]=

Out[515]=

Out[516]=

Out[517]=

Plot EPSC

In[518]:= `epscHighCa = D[(ss4[t] /. myNDSolveResults), t];`
`Plot[(ss4[t] /. myNDSolveResults), {t, 0, 2*^-3}, PlotRange → All];`
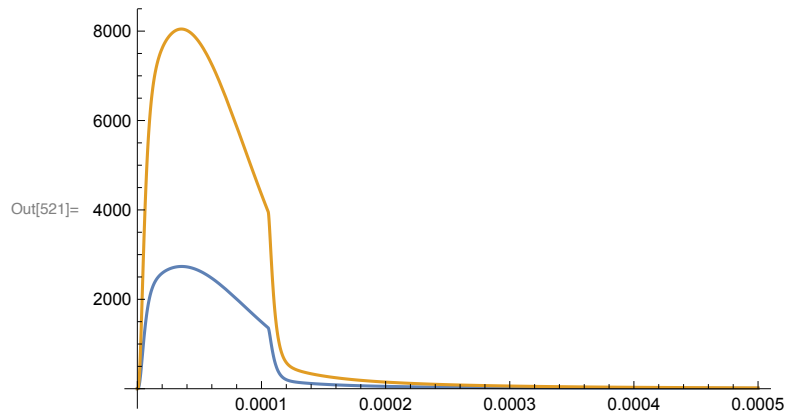`Plot[epscLowCa, {t, 0, 0.5*^-3}, PlotRange → All]`

Out[520]=

Compare

Plot both release rates

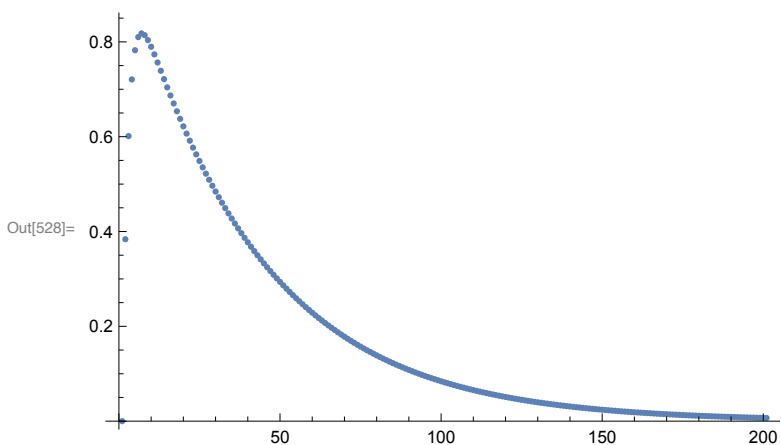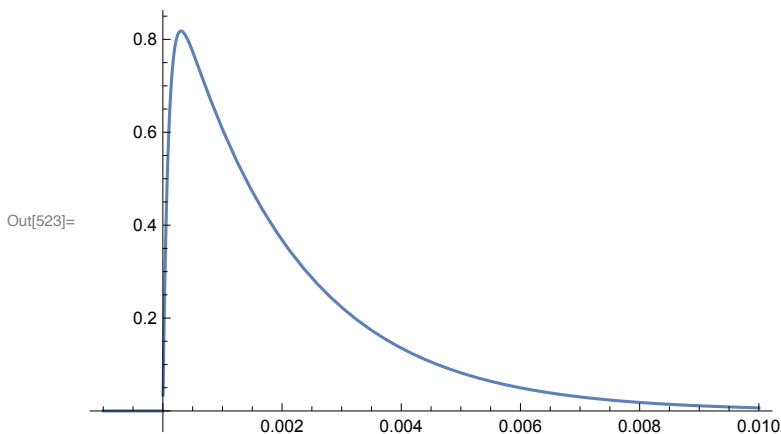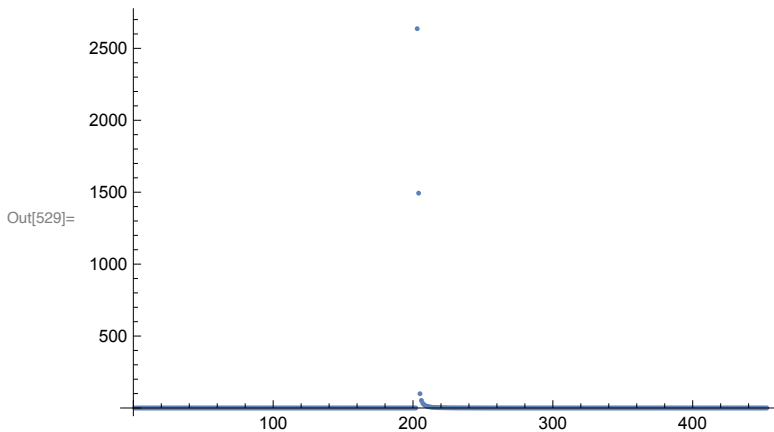In[521]:= `Plot[{epscLowCa, epscHighCa}, {t, 0, 0.5*^-3}, PlotRange → All]`

Out[521]=

Convolution RelRate => EPSC

```
In[522]:= 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[523]=

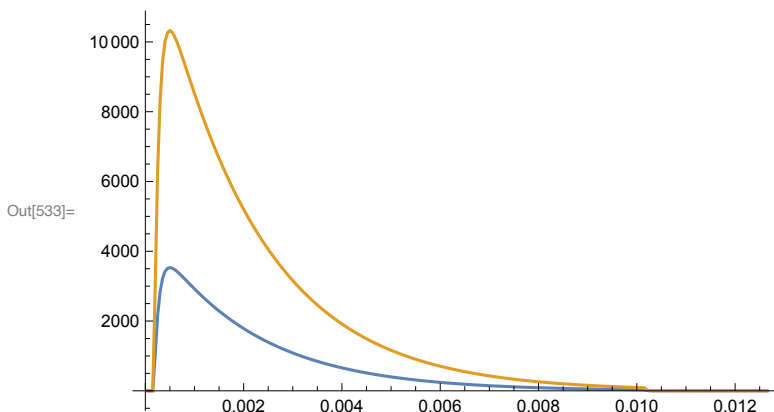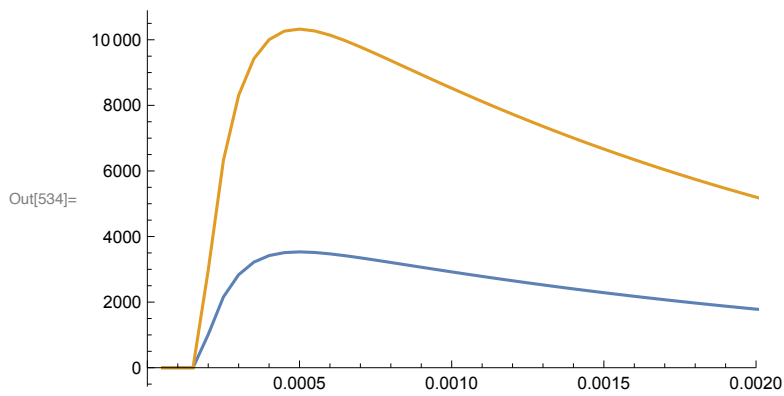Out[528]=

Out[529]=


```
In[530]:= 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"];
          ];
```
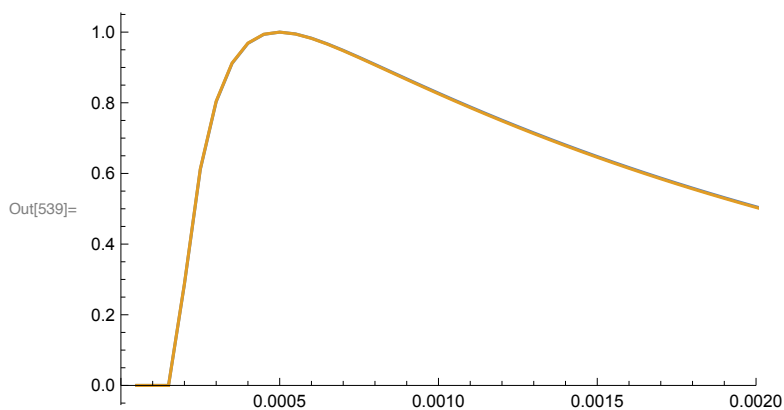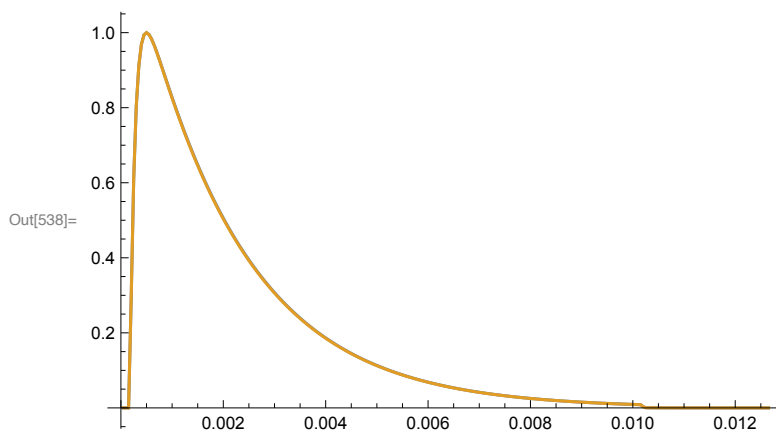
Out[533]=

Out[534]=

Out[535]= 3532.14

Out[536]= 10 325.2

maxHigh/maxLow = 2.92322

Out[538]=

Out[539]=

Timing

In[541]:= **timeEnd = AbsoluteTime[];**
(timeEnd - timeStart)/60.(* time of calculation in min *)

Out[542]= 0.280824