

```
In[1]:= (* Mathematica code for Model3 of Eshra et al. eLife 2021 *)  
        (* Stefan Hallermann and Hartmut Schmidt Aug 2021 *)
```

# Import

---

## general

```
In[2]:= (* imported data all Ca in uM, all times in ms,  
        all amplitudes and Nv in vesicles *)  
  
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

```

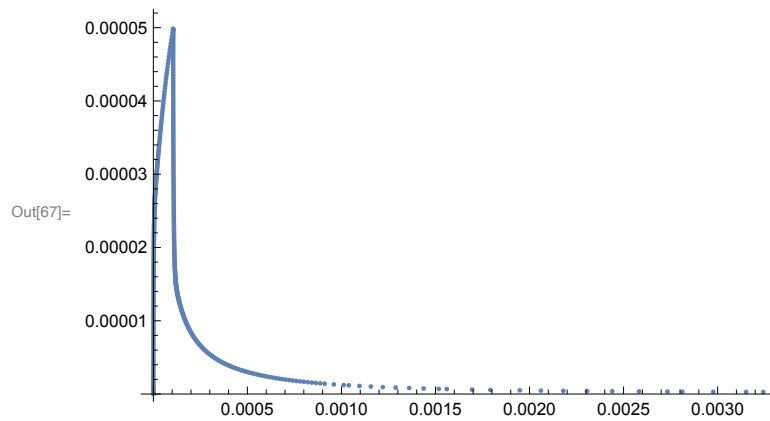
In[59]:= data = Import[dataFolder <> "all_t2_v02_D.txt", "Table"];

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]
```



# 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[]
```

```
Out[75]= 3.841510544996684 × 109
```

## 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 = CmToVesConversionFactor * 1.36937*^-14 / rrp(*cannot be 0*)
signalToNoiseRatioC5 = 1.;(*minimum s-to-n-ratio to attempt fitting*)
dtOfDataC5 = (1 / (1000 * samplingOfDataInKHzC5));

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

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

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

```
Out[82]= 0.217071
```

```
Out[86]= 0.265651
```

```
Out[90]= 0.0367584
```

## 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  $\text{Ca}^{2+}$  transitions. All calculations are repeated in a loop with increasing uncaging efficacy for three different DMN concentrations. The resulting free  $\text{Ca}^{2+}$  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 resting 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 * 10^8;
            kOffOG = 103.;]

          If[OGB5N == 1,
            kOnOG = 2.5 * 10^8;
            kOffOG = 6000. * 31.4 / 24;]

          If[OGB6F == 1,

```

```

    kOnOG = 3. * 108;
    kOffOG = 900.;]

If[Fluo5F == 1,
    kOnOG = 3. * 108;
    kOffOG = 249.;] (*Delvendahl PNAS; before: 432*)

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

(*ATP*)
kOnATP = 5. * 108;
kOffATP = 100000.;
kOnMgATP = 1. * 107; (*Bollmann Dissertation S. 59; *)
kOffMgATP = 1000.;

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

(*DM*)
If[MgT == 0.,
    kOnDM = 1.98 * 107; (*Faas Plos Biol, 2007*)
    kOffDM = 0.14;
    ,
    kOnDM = 2.9 * 107; (*Faas Biophys J, 2005*)
    kOffDM = 0.19;
]

(*Mg binding constants for DMn, DMf, DMs*)
kOnMg = 1.3 * 105; (*all values for Mg are from Faas et al., 2005*)
kOffMg = 0.2;

(*Ca binding constants for PP*)
kOnPP2 = kOnPP1 = kOnDM;
kOffPP2 = 3.6 * 103;
If[MgT == 0.,
    kOffPP1 = 7. * 104;;
    kOffPP1 = 6.9 * 104; ]

(*Mg binding constants for PP1,PP2*)
kOffMgPP = 3. * 102; (*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 * 108; (*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 nitrophenene*)
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 uncaging;
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 resting 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 * 108;
    kOffOG = 103.;]

If[OGB5N == 1,
    kOnOG = 2.5 * 108;
    kOffOG = 6000. * 31.4 / 24;]

If[OGB6F == 1,
    kOnOG = 3. * 108;
    kOffOG = 900.;]

If[Fluo5F == 1,
    kOnOG = 3. * 108;
    kOffOG = 249.;] (*Delvendahl PNAS; before: 432*)

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

(*ATP*)
kOnATP = 5. * 108;
kOffATP = 100000.;
kOnMgATP = 1. * 107; (*Bollmann Dissertation S. 59; *)
kOffMgATP = 1000.;

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

(*DM*)
If[MgT == 0.,
    kOnDM = 1.98 * 107; (*Faas Plos Biol, 2007*)
    kOffDM = 0.14;
    ,
    kOnDM = 2.9 * 107; (*Faas Biophys J, 2005*)
    kOffDM = 0.19;
]

(*Mg binding constants for DMn, DMf, DMs*)
kOnMg = 1.3 * 105; (*all values for Mg are from Faas et al., 2005*)
kOffMg = 0.2;

(*Ca binding constants for PP*)
kOnPP2 = kOnPP1 = kOnDM;
kOffPP2 = 3.6 * 103;
If[MgT == 0.,
    kOffPP1 = 7. * 104;,

```

```

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 nitrophenene*)
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 resting 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. * 108;
kOffATP = 100 000.;
kOnMgATP = 1. * 107; (*Bollmann Dissertation S. 59; *)
kOffMgATP = 1000.;

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

(*DM*)
If[MgT == 0.,
  kOnDM = 1.98 * 107; (*Faas Plos Biol, 2007*)
  kOffDM = 0.14;
  ,
  kOnDM = 2.9 * 107; (*Faas Biophys J, 2005*)
  kOffDM = 0.19;
]

(*Mg binding constants for DMn, DMf, DMs*)
kOnMg = 1.3 * 105; (*all values for Mg are from Faas et al., 2005*)
kOffMg = 0.2;

(*Ca binding constants for PP*)
kOnPP2 = kOnPP1 = kOnDM;
kOffPP2 = 3.6 * 103;
If[MgT == 0.,
  kOffPP1 = 7. * 104;;
  kOffPP1 = 6.9 * 104; ]

(*Mg binding constants for PP1,PP2*)
kOffMgPP = 3. * 102; (*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 * 108; (*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 nitrophenes*)
```

```

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]];

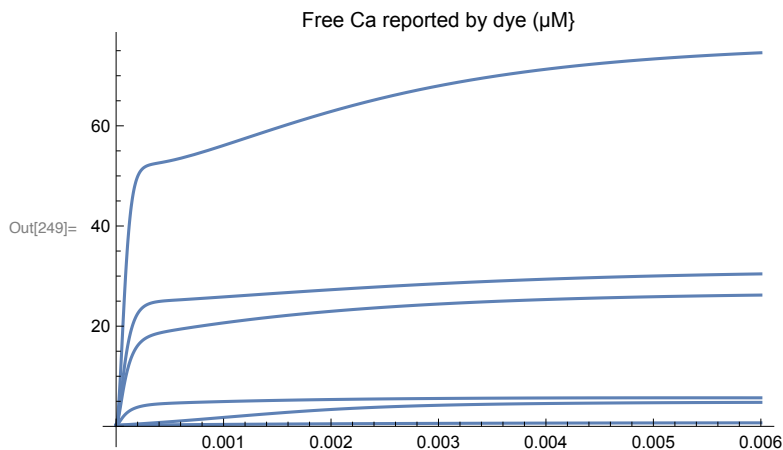
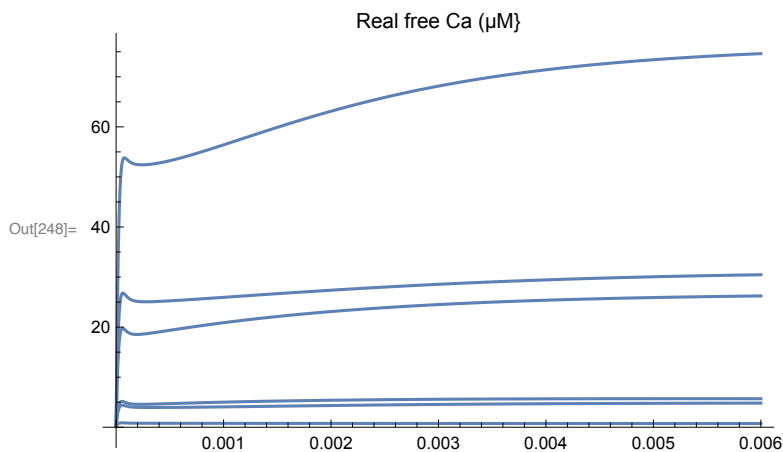
```

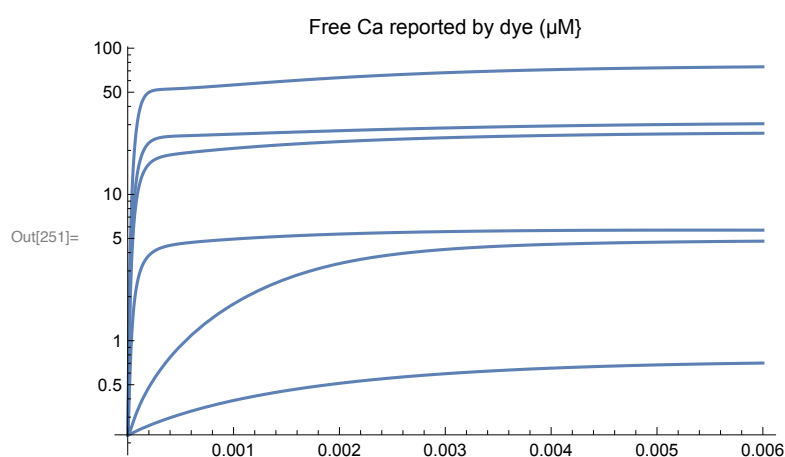
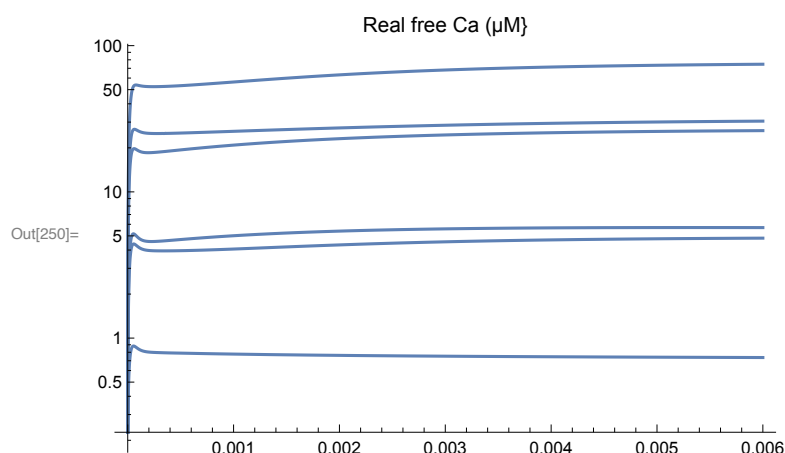
# Plot and further processing

```

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 needed to get rid of these brackets {{{}}})

```





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

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

# Release scheme and parameters

## Define release scheme A

In[254]:= nStates = 7;  
 matA = Table[0, {nStates}, {nStates}];

## forward rates

```

In[256]:= from = 1; (*0 ca bound*)
kk = 5 konA;
matA[[from + 1, from]] += kk; matA[[from, from]] += -kk;

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

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

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

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

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

```

## backwards rates

```

In[274]:= from = 2; (*1 ca bound*)
kk = koffA bA^0;
matA[[from - 1, from]] += kk; matA[[from, from]] += -kk;

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

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

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

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

```

## outflux from matrix

```

In[289]:= matA[[1, 1]] += -kunprimA - kprimB;
(*due to multiplication with ss1A[t] in mat.ss it results in:
-kunprimA ss1A[t] -kprimB ss1A[t]
*)

```

## influx in matrix

```

In[290]:= matA[[1, 1]] += kprimA (1 - ss1A[t]) / ss1A[t] + kunprimB ss1B[t] / ss1A[t];
(* due to multiplication with ss1A[t] in mat.ss it results in:
kprimA + kunprimB ss1B[t] *)

```



## Matrix

```
In[291]:= matA // TableForm
Out[291]/TableForm=
```

$-5 \text{ konA} - \text{kprimB} - \text{kunprimA} + \frac{\text{kprimA} (1 - \text{ss1A}[\text{t}])}{\text{ss1A}[\text{t}]} + \frac{\text{kunprimB} \text{ss1B}[\text{t}]}{\text{ss1A}[\text{t}]}$	koffA	0
5 konA	$-\text{koffA} - 4 \text{ konA}$	2 bA koff
0	4 konA	- 2 bA ko1
0	0	3 konA
0	0	0
0	0	0
0	0	0

## Define release scheme B

```
In[292]:= nStates = 7;
matB = Table[0, {nStates}, {nStates}];
```

### forward rates

```
In[294]:= from = 1; (*0 ca bound*)
kk = 5 konB;
matB[[from + 1, from]] += kk; matB[[from, from]] += -kk;

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

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

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

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

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

## backwards rates

```

In[312]:= from = 2; (*1 ca bound*)
kk = koffB bB^0;
matB[[from - 1, from]] += kk; matB[[from, from]] += -kk;

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

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

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

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

```

## outflux from matrix

```

In[327]:= matB[[1, 1]] += -kunprimB;
(* due to multiplication with ss1B[t] in mat.ss it results in:
   -kunprimB ss1B[t]
*)

```

## influx in matrix

```

In[328]:= matB[[1, 1]] += kprimB ss1A[t] / ss1B[t];
(* due to multiplication with ss1B[t] in mat.ss it results in:
   kprimB ss1A[t]
*)

```

## Matrix

```
In[329]:= matB // TableForm
Out[329]//TableForm=
```

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

## Parameters of release scheme

```
In[330]:= q10 = 2.3;
tempFact = q10 ^ ((37 - 24) / 10);

In[332]:= Clear[caFunc, konSchemeA, konSchemeB];
(*Clear is needed if the cell is executed for a 2nd time when
caFunc is already set to a value or an Interpolationfunction*)

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

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

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

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

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

```

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

  kprimA → kprimSchemeA ,
  kunprimA → kunprimSchemeA ,

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

  kprimB → kprimSchemeB ,
  kunprimB → kunprimSchemeB
};

```

```
In[351]:= tauOfDecayOfUncagedCa = 0.4;
```

---

## Initial occupancy

```

In[352]:= (*test initial equilibrium occupancy*)
caFunc[t_] := caTmp;
ss0AInitial = kprimSchemeA / kunprimSchemeA;
ss0BInitial = (kprimSchemeA / kunprimSchemeA) * (kprimSchemeB / kunprimSchemeB);
(ss0AInitial /. caTmp → 180*^-9)
(ss0AInitial /. caTmp → 30*^-9)
(ss0AInitial /. caTmp → 180*^-9) / (ss0AInitial /. caTmp → 30*^-9)
(ss0BInitial /. caTmp → 180*^-9)
(ss0BInitial /. caTmp → 30*^-9)
(ss0BInitial /. caTmp → 180*^-9) / (ss0BInitial /. caTmp → 30*^-9)

```

```
Out[355]= 1.
```

```
Out[356]= 1.
```

```
Out[357]= 1.
```

```
Out[358]= 0.836742
```

```
Out[359]= 0.26514
```

```
Out[360]= 3.15584
```

```

In[361]:= (*calualte initial equilibrium occupancy*)
caFunc[t_] := caRest;
ss0AInitial = kprimSchemeA / kunprimSchemeA;
ss0BInitial = (kprimSchemeA / kunprimSchemeA) * (kprimSchemeB / kunprimSchemeB);

```

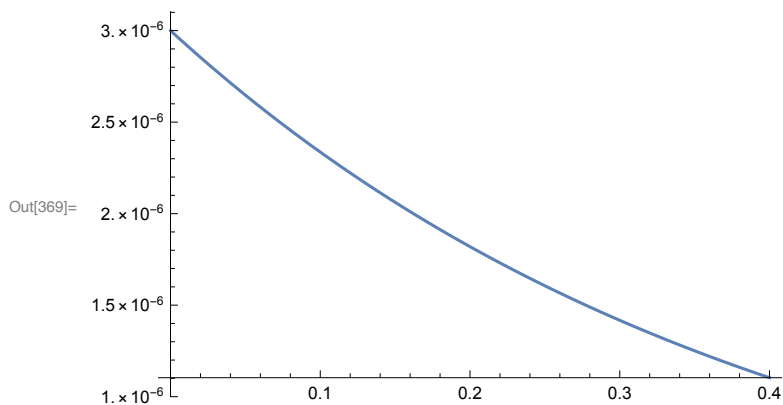
## Diff eq.

```
In[364]:= Clear[caFunc, eq]; (*Clear is needed if the cell is executed for a 2nd time
when caFunc is already set to a value or an Interpolationfunction*)
ssA[t_] = {ss1A[t], ss2A[t], ss3A[t], ss4A[t], ss5A[t], ss6A[t], ss7A[t]};
ssB[t_] = {ss1B[t], ss2B[t], ss3B[t], ss4B[t], ss5B[t], ss6B[t], ss7B[t]};
eq = {
  ssA'[t] == (matA /. repl).ssA[t],
  ssA[0] == {ss0AInitial, 0, 0, 0, 0, 0, 0},

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

## Solve all states

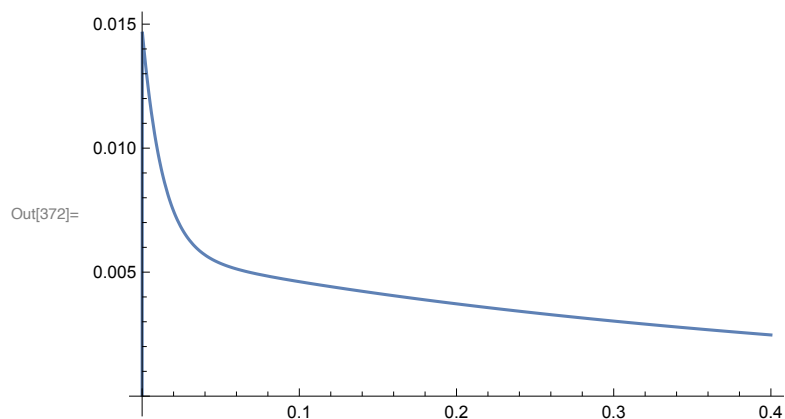
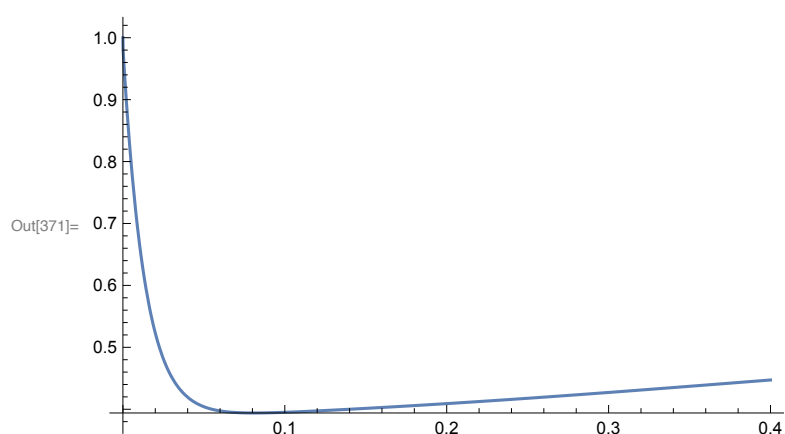
```
In[368]:= caFunc[t_] := 3*^-6 * Exp[-t / tauOfDecayOfUncagedCa];
Plot[caFunc[t], {t, 0, 0.4}, PlotRange -> All]
```

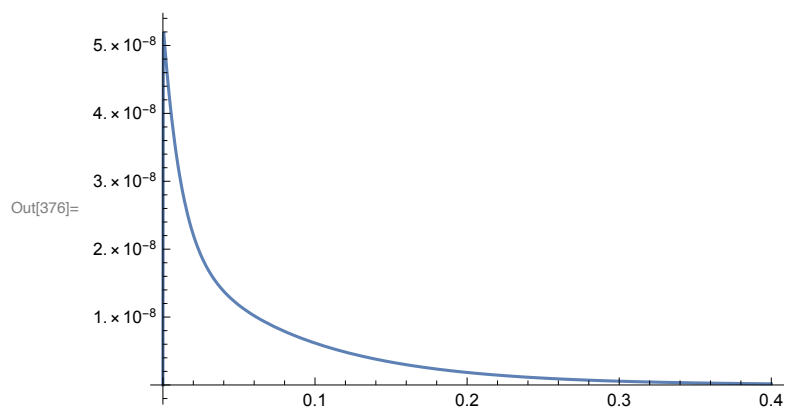
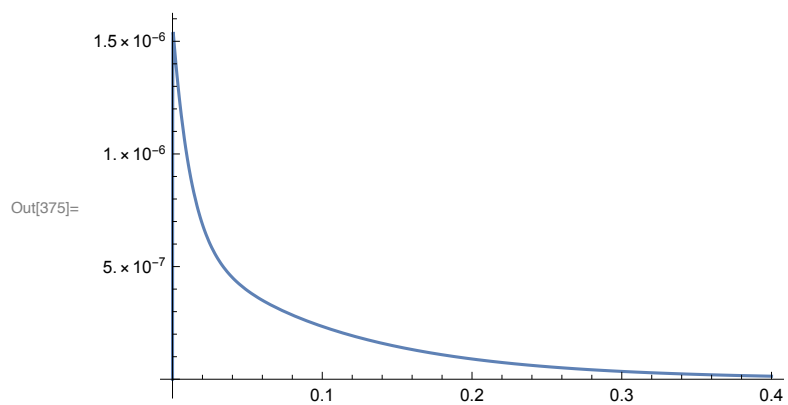
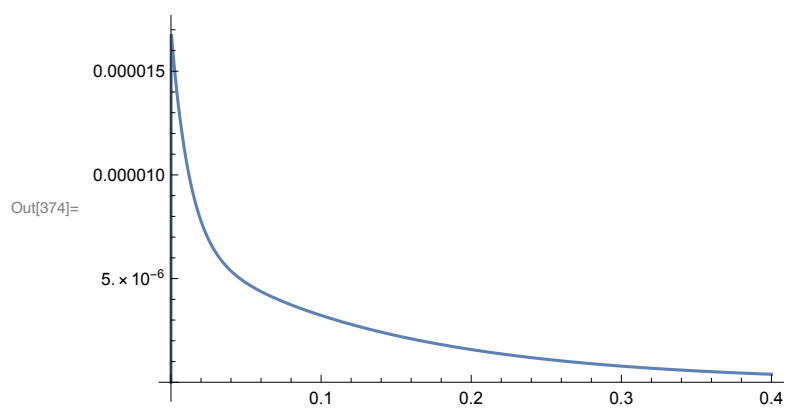
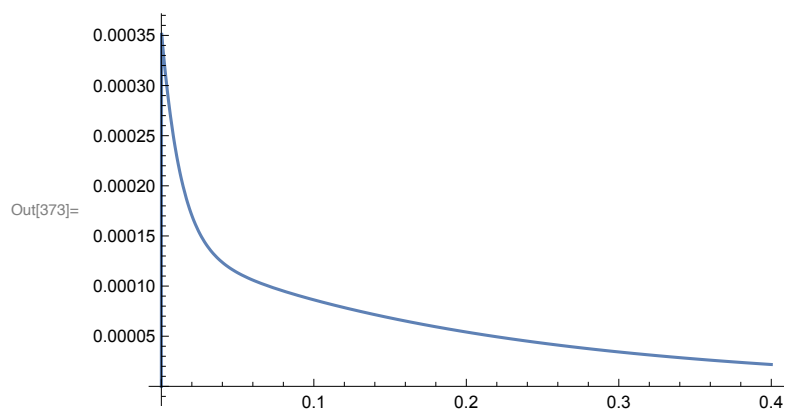


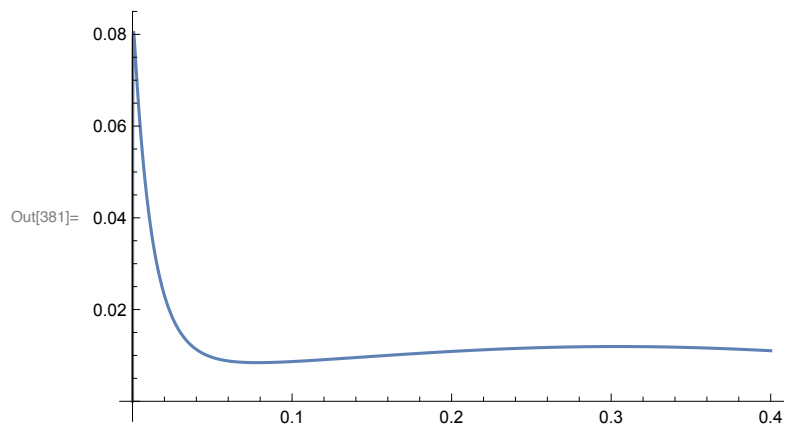
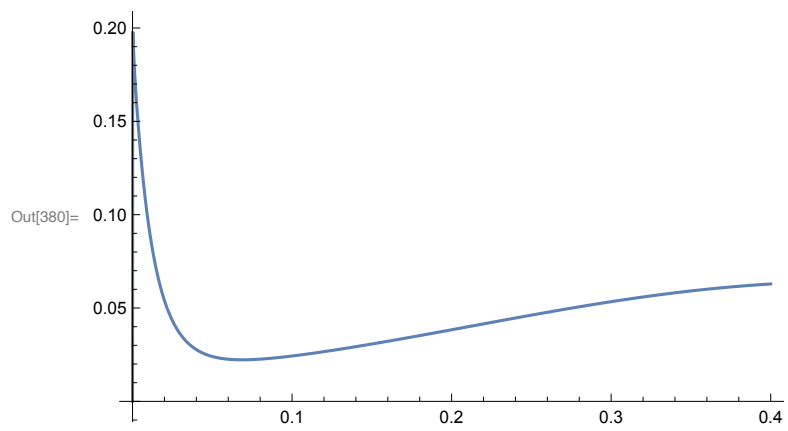
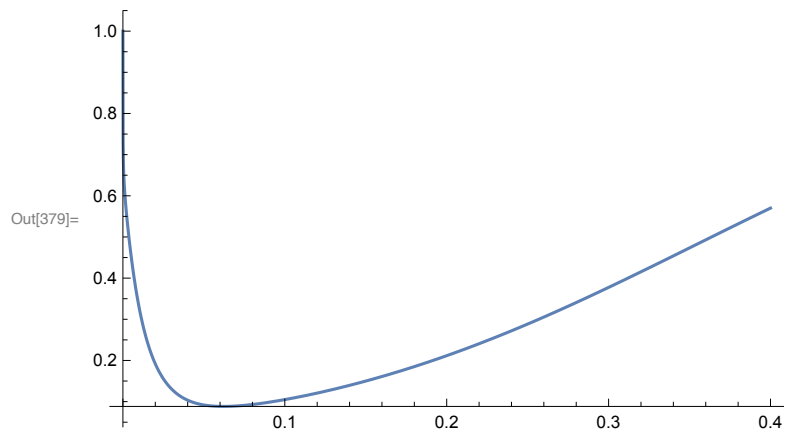
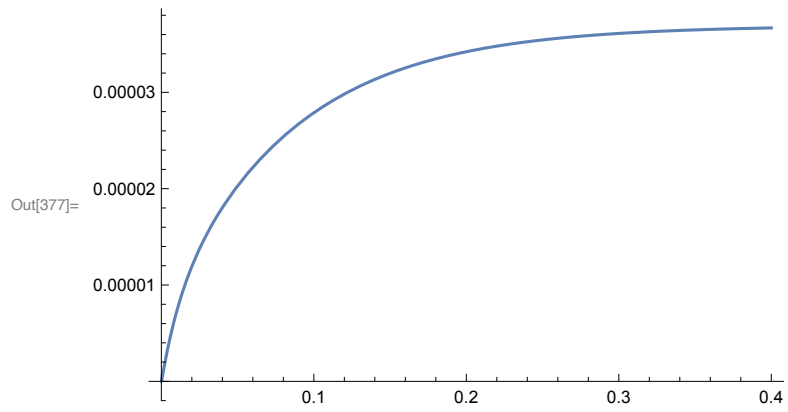
```

In[370]:= myNDSolveResults = NDSolve[eq, {ss1A, ss2A, ss3A, ss4A, ss5A,
      ss6A, ss7A, ss1B, ss2B, ss3B, ss4B, ss5B, ss6B, ss7B}, {t, 0, 10.}];
tEndForPlot = .4;
Plot[(ss1A[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange -> All]
Plot[(ss2A[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange -> All]
Plot[(ss3A[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange -> All]
Plot[(ss4A[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange -> All]
Plot[(ss5A[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange -> All]
Plot[(ss6A[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange -> All]
Plot[(ss7A[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange -> All]
Print[
  "-----"]
Plot[(ss1B[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange -> All]
Plot[(ss2B[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange -> All]
Plot[(ss3B[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange -> All]
Plot[(ss4B[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange -> All]
Plot[(ss5B[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange -> All]
Plot[(ss6B[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange -> All]
Plot[(ss7B[t] /. myNDSolveResults), {t, 0, tEndForPlot}, PlotRange -> All]

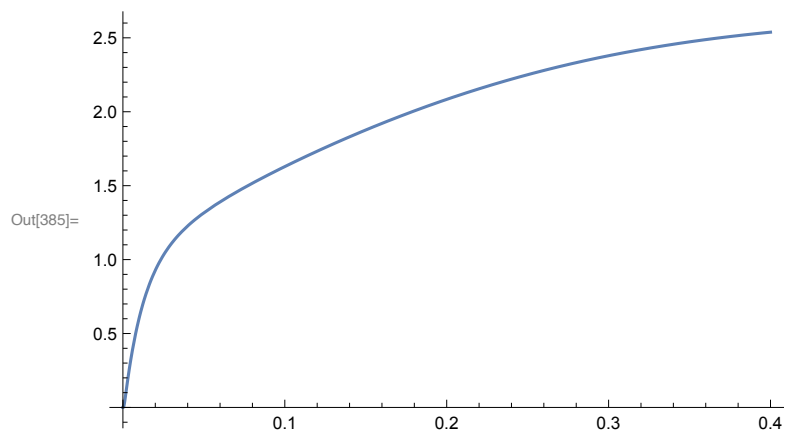
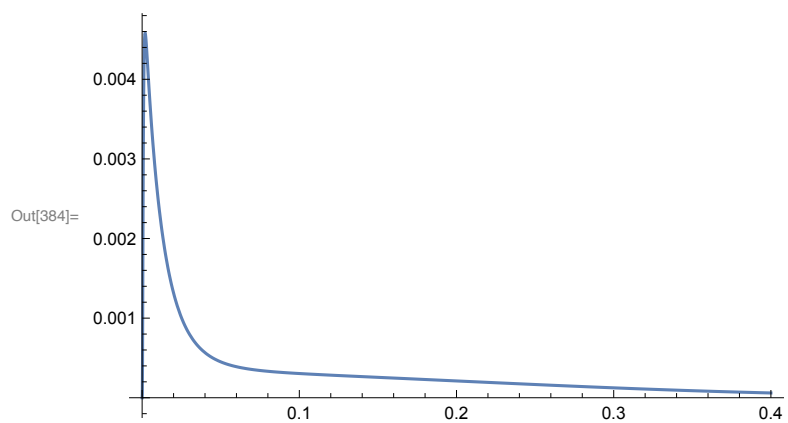
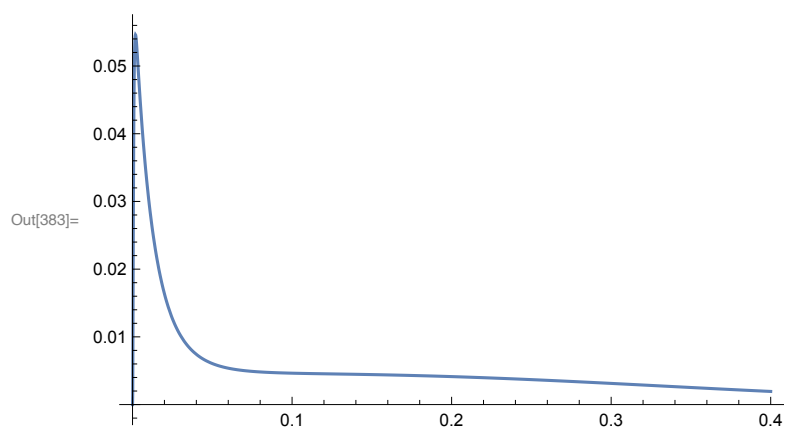
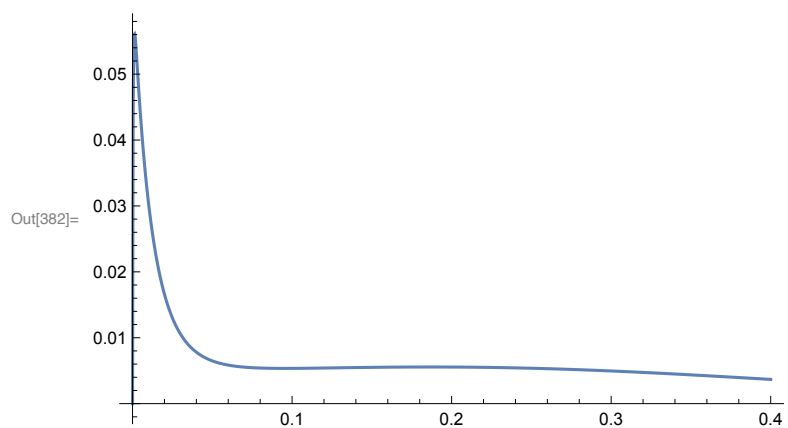
```











# Loop

## make lists for later

```
In[386]:= simCaList = Table[0, {runs}];
          simParamNv = Table[0, {7}, {runs}];
```

### C5

```
In[388]:= baselineC5 = Table[{ttt, 0}, {ttt, cursorStart, -1*^-6, dtOfDataC5}];
          (*List to save data within loop*)
          (*ca; tau; number of released vesilces at time=cursorEnd; delay*)
          simParamNoiseC5 = Table[0, {numberOfFitParamToBeSaved}, {noiseRepeats}];
          simParamMedianC5 = Table[0, {numberOfFitParamToBeSaved}, {runs}];
          simParamQuantile1C5 = Table[0, {numberOfFitParamToBeSaved}, {runs}];
          simParamQuantile2C5 = Table[0, {numberOfFitParamToBeSaved}, {runs}];
```

### C10

```
In[393]:= baselineC10 = Table[{ttt, 0}, {ttt, cursorStart, -1*^-6, dtOfDataC10}];
          (*List to save data within loop*)
          (*ca; tau; number of released vesilces at time=cursorEnd; delay*)
          simParamNoiseC10 = Table[0, {numberOfFitParamToBeSaved}, {noiseRepeats}];
          simParamMedianC10 = Table[0, {numberOfFitParamToBeSaved}, {runs}];
          simParamQuantile1C10 = Table[0, {numberOfFitParamToBeSaved}, {runs}];
          simParamQuantile2C10 = Table[0, {numberOfFitParamToBeSaved}, {runs}];
```

### D

```
In[398]:= baselineD = Table[{ttt, 0}, {ttt, cursorStart, -1*^-6, dtOfDataD}];
          (*List to save data within loop*)
          (*ca; tau; number of released vesilces at time=cursorEnd; delay*)
          simParamNoiseD = Table[0, {numberOfFitParamToBeSaved}, {noiseRepeats}];
          simParamMedianD = Table[0, {numberOfFitParamToBeSaved}, {runs}];
          simParamQuantile1D = Table[0, {numberOfFitParamToBeSaved}, {runs}];
          simParamQuantile2D = Table[0, {numberOfFitParamToBeSaved}, {runs}];
```

### Long

```
In[403]:= baselineLong = Table[{ttt, 0}, {ttt, -0.05, -1*^-6, dtOfDataLong}];
```

## Loop

```

In[404]:= For[r = 1, r ≤ runs, r += 1,
  myCaNow = CaListDyePeak[[r]];
  lastSimulatedCaListReal = CaListReal[[r]][[1]][[1]] /. tt → TimeWindow;
  (* "[1][1]" is needed to get rid of these brackets {} *)
  Print[
    "-----
    -----"];
  Print["----- Ca = ", 1*^6 myCaNow,
    " uM -----"];
  Print[
    "-----
    -----"];
  simCaList[[r]] = myCaNow;
  caFunc[ttt_] := If[ttt < TimeWindow,
    CaListReal[[r]][[1]][[1]] /. tt → ttt,
    (* tt because this is the symbol used in "Calculate Ca transients" *)
    lastSimulatedCaListReal * Exp[-ttt/tauOfDecayOfUncagedCa]
  ];
  (* solve Diff Eq.: *)
  myNDSolveResults = NDSolve[eq, {ss7A, ss7B}, {t, 0, 0.4}];
  (* plot results: *)
  fused[t_] := (ss7A[t] + ss7B[t]);
  Plot[(fused[t] /. myNDSolveResults),
    {t, 0, cursorEnd}, PlotRange → All] // Print;
  Plot[(fused[t] /. myNDSolveResults), {t, 0, cursorEndLong},
    PlotRange → All] // Print;
  If[exportYes == 1,
    toExport = Table[{t, (fused[t] /. myNDSolveResults)[[1]]},
      {t, 0., cursorEndLong, dtOfPlotsForExport}];
    Export["withinLoop r" <> ToString[r] <> " Ca" <> ToString[1*^6 myCaNow] <>
      " withoutNoise.txt", toExport, "Table"];
  ];
  (* sample data and add baseline: -----*)
  tmpCumRelC5 = Table[{ttt, (fused[ttt] /. myNDSolveResults)[[1]]},
    {ttt, 0., cursorEnd, dtOfDataC5}];
  tmpToFitC5 = Catenate[{baselineC5, tmpCumRelC5}];
  tmpCumRelC10 = Table[{ttt, (fused[ttt] /. myNDSolveResults)[[1]]},
    {ttt, 0., cursorEnd, dtOfDataC10}];
  tmpToFitC10 = Catenate[{baselineC10, tmpCumRelC10}];
  tmpCumRelD = Table[{ttt, (fused[ttt] /. myNDSolveResults)[[1]]},
    {ttt, 0., cursorEnd, dtOfDataD}];
  tmpToFitD = Catenate[{baselineD, tmpCumRelD}];
  tmpCumRelLong = Table[{ttt, (fused[ttt] /. myNDSolveResults)[[1]]},
    {ttt, 0., cursorEndLong, dtOfDataLong}];
  tmpToFitLong = Catenate[{baselineLong, tmpCumRelLong}];

```

```

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

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

(*----- Fitting of data,
saving of results, and plotting-----*)
(*-----*)
(*----- C5 -----*)
(*-----*)
(*-----*)
(*-----*)

Print[
  "-----
  C5"];
(*check that signal is large enough relative to noise to obtain
useful fit results. If not, do not do fitting and set everything to {}*)
If[simParamNv[[5, r]] > signalToNoiseRatioC5 myNoiseC5, (*add noise
several times and do the fitting and 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 criteria should be fulfilled:*)
  (*chi2 should improve(=decrease) by >10%*)
  (simParamNoiseC5[[6, noiseN]] > 1.04)
  &&
  (*tau1 and tau2 of bi fit should be factor of >3 different*)
  ((simParamNoiseC5[[11, noiseN]] / simParamNoiseC5[[10, noiseN]]) < 3.)
  &&
  (*relative amplitude of 1st component should be > 5% *)
  (((amp1 /. fitResultBi)[[1]] > 0.05)
  &&
  (*relative amplitude of 1st component should be < 95% *)
  (((amp1 /. fitResultBi)[[1]] < 0.95)
  ,
  (*take bi*)
  Print["take bi"];
  simParamNoiseC5[[12, noiseN]] = simParamNoiseC5[[7, noiseN]];
  (*delay*)
  simParamNoiseC5[[13, noiseN]] = simParamNoiseC5[[8, noiseN]];
  (*amp*)
  simParamNoiseC5[[14, noiseN]] = simParamNoiseC5[[9, noiseN]];
  (*amp1*)
  simParamNoiseC5[[15, noiseN]] = simParamNoiseC5[[10, noiseN]];
  (*tau1*)
  simParamNoiseC5[[16, noiseN]] = simParamNoiseC5[[11, noiseN]]; (*tau2*)
  ,
  (*take mono*)
  Print["take mono"];
  simParamNoiseC5[[12, noiseN]] = simParamNoiseC5[[3, noiseN]];
  (*delay*)

```

```

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

```

```

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

```

```

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

```





```

    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 criteria should be fulfilled:*)
  (*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 noisettrace*)
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 criteria should be fulfilled:*)
(*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"];

```



```

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

```

```

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

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

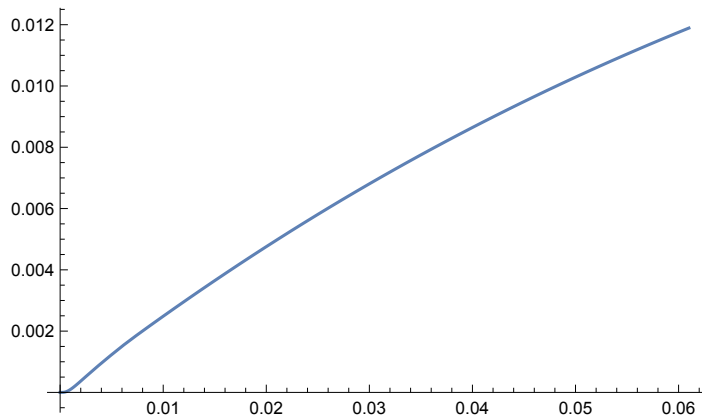
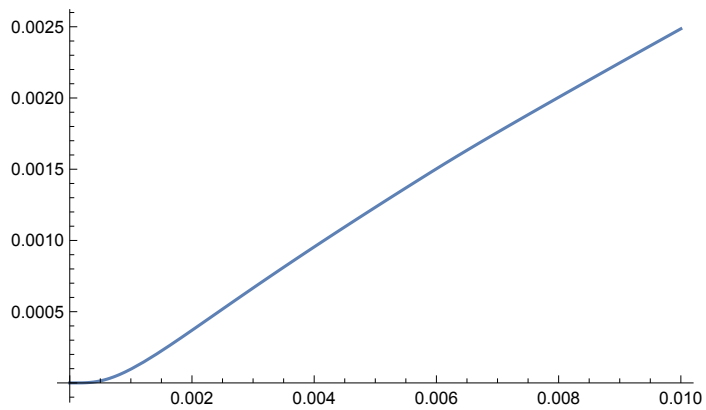
, (*else: signal is not large enough*)
For[p = 1, p ≤ numberOfFitParamToBeSaved, p += 1,
  simParamMedianD[[p, r]] = {};
  simParamQuantile1D[[p, r]] = {};
  simParamQuantile2D[[p, r]] = {};
];

```

```
];
];
```

```
----- Ca = 0.703073
```

```
uM
```



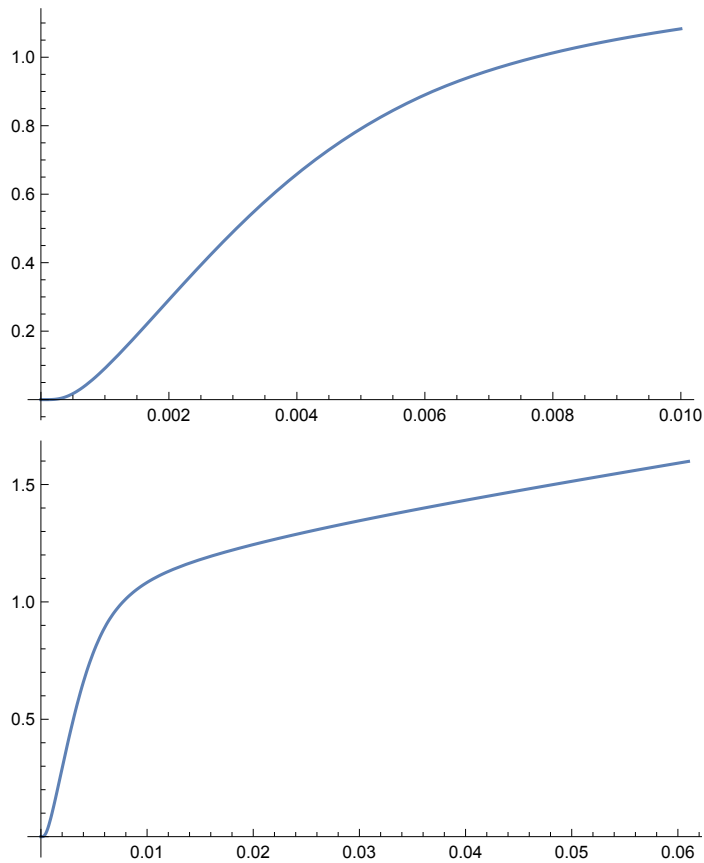
```
----- C5
```

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

```
----- D
```

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

```
uM
```



C5

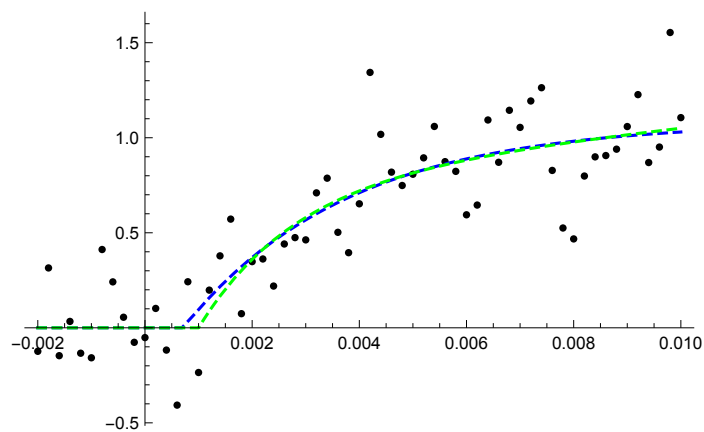
NonlinearModelFit: Failed to converge to the requested accuracy or precision within 100 iterations.

take mono

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

take mono

take mono



Mono:  $\chi^2 = 2.96309$        $d = 2.96309$        $a = 1.08303$        $t = 0.00307561$

Bi: ratio  $\chi^2_{\text{Mono}}/\chi^2_{\text{Bi}} = 1.00705$        $d = 0.001$        $a =$   
 $2.66801$        $a1 = 0.753421$        $t1 = 0.00177641$        $t2 = 0.0525808$

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

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

\*\*\* **Quantile:** Argument {} should be a non-empty list.

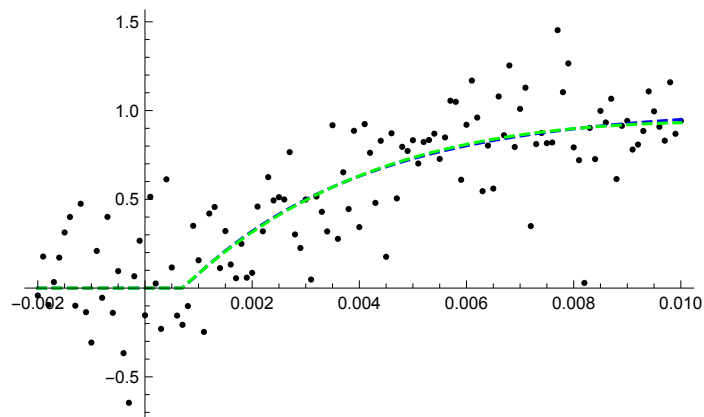
\*\*\* **General:** Further output of Quantile::empt will be suppressed during this calculation.

----- C10

take mono

take mono

take mono



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

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

----- D

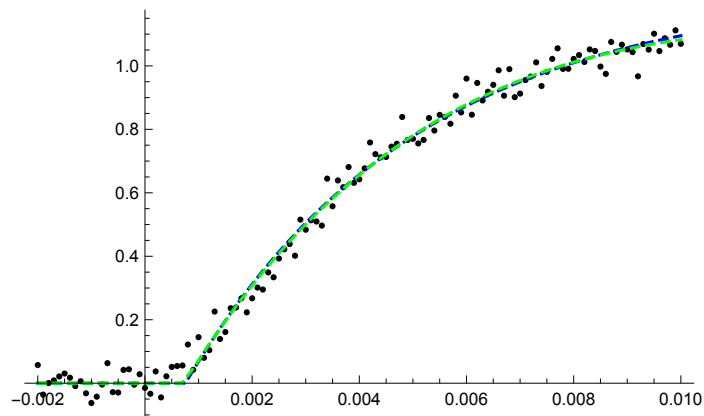
\*\*\* **NonlinearModelFit:** Failed to converge to the requested accuracy or precision within 100 iterations.

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

take mono

take mono

take mono

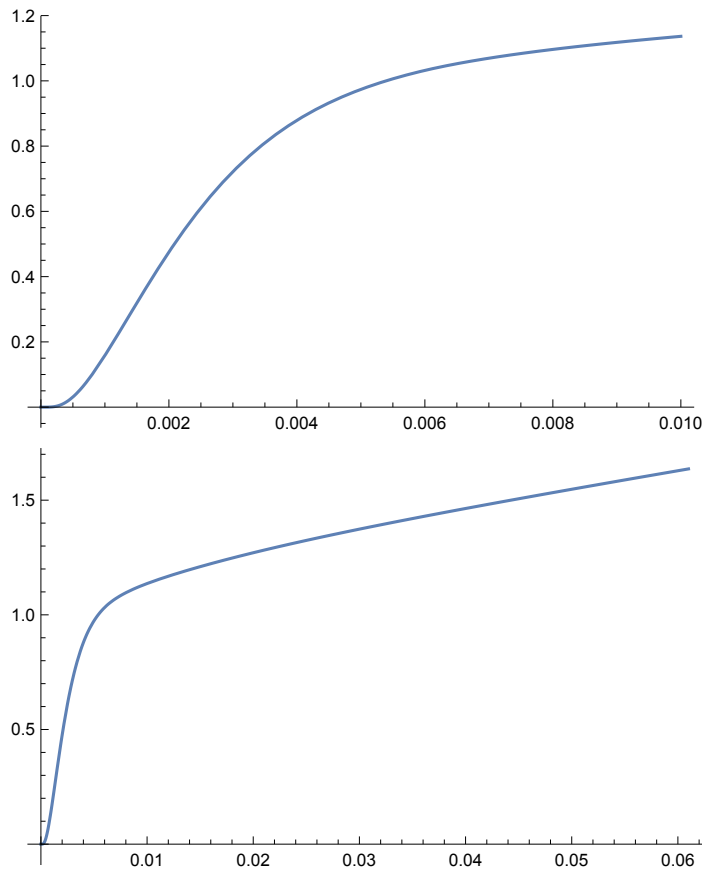


Mono: chi2 = 0.175917      d = 0.175917      a = 1.23782      t = 0.00428099

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

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

uM -----

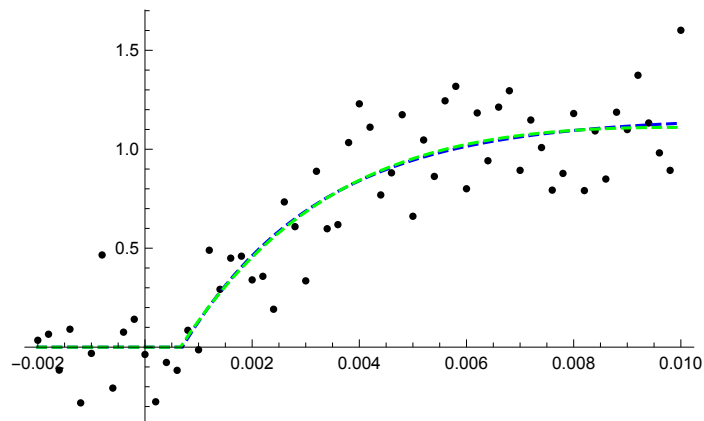


C5

take mono

take mono

take mono



Mono:  $\chi^2 = 2.47782$        $d = 2.47782$        $a = 1.16243$        $t = 0.00257036$

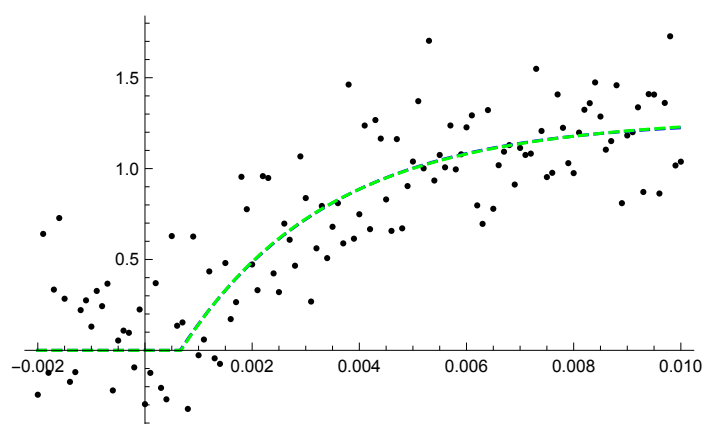
Bi: ratio  $\chi^2_{\text{Mono}}/\chi^2_{\text{Bi}} = 1.0028$        $d = 0.000663083$        $a =$   
 $0.962649$        $a1 = 5.53286$        $t1 = 0.00430774$        $t2 = 0.00528536$

C10

take mono

take mono

take mono



Mono:  $\chi^2 = 8.54551$        $d = 8.54551$        $a = 1.26908$        $t = 0.00278213$

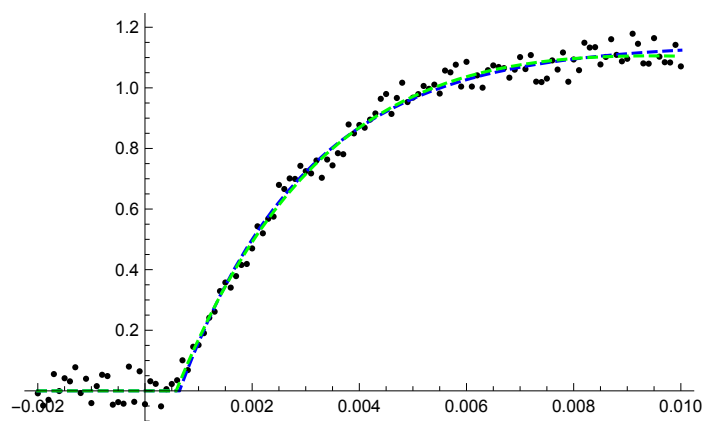
Bi: ratio  $\chi^2_{\text{Mono}}/\chi^2_{\text{Bi}} = 1.00005$        $d = 0.000672742$        $a =$   
 $1.3105$        $a1 = 1.08952$        $t1 = 0.00250859$        $t2 = 0.00668579$

----- D

take mono

take mono

take mono



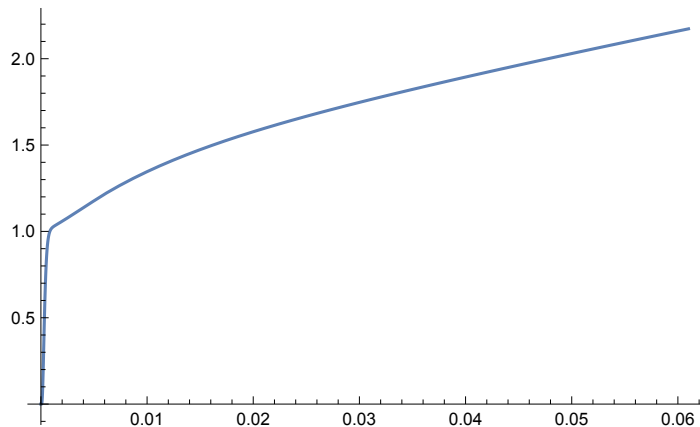
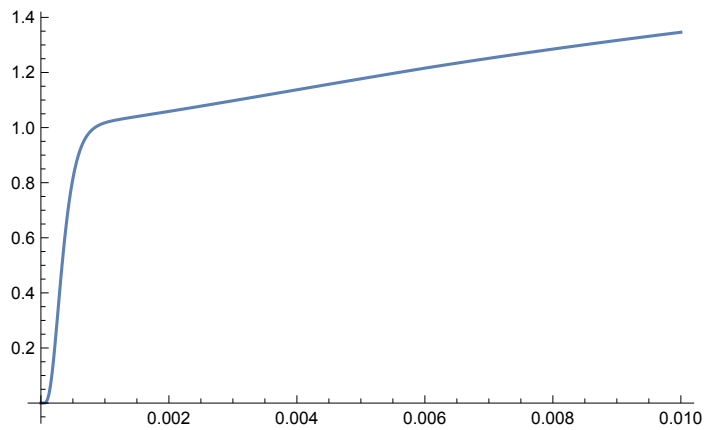
Mono:  $\chi^2 = 0.161382$        $d = 0.161382$        $a = 1.14624$        $t = 0.00236594$

Bi: ratio  $\chi^2_{\text{Mono}}/\chi^2_{\text{Bi}} = 1.04593$        $d = 0.000581982$        $a =$   
 $0.957308$        $a1 = 3.21524$        $t1 = 0.00382601$        $t2 = 0.00561116$

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

uM -----





C5

General:  $\text{Exp}[-788.402]$  is too small to represent as a normalized machine number; precision may be lost.

General:  $\text{Exp}[-1576.25]$  is too small to represent as a normalized machine number; precision may be lost.

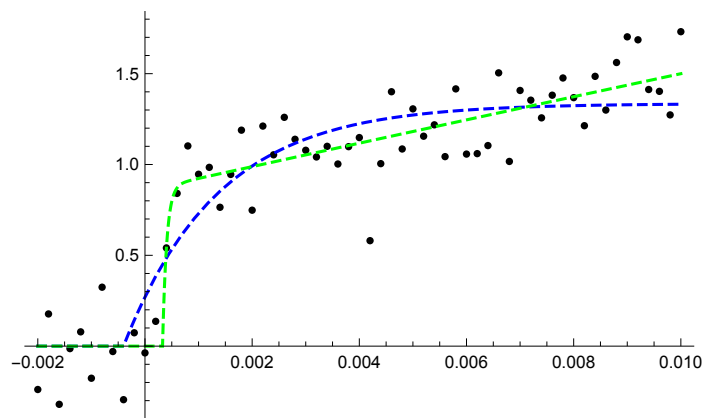
General:  $\text{Exp}[-2364.1]$  is too small to represent as a normalized machine number; precision may be lost.

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

take bi

take bi

take mono



Mono:  $\chi^2 = 2.70432$        $d = 2.70432$        $a = 1.33533$        $t = 0.00176702$

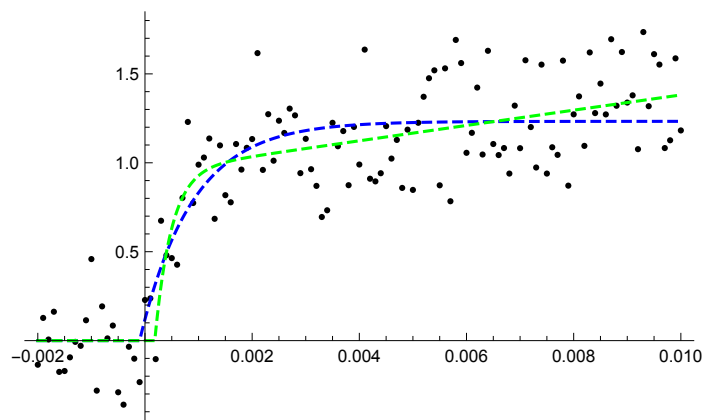
Bi: ratio  $\chi^2_{\text{Mono}}/\chi^2_{\text{Bi}} = 1.51932$        $d = 0.00032778$        $a =$   
        $21.0452$        $a1 = 0.8795$        $t1 = 0.0000772609$        $t2 = 0.30957$

C10

take bi

take bi

take bi



Mono:  $\chi^2 = 6.98349$        $d = 6.98349$        $a = 1.23306$        $t = 0.00097935$

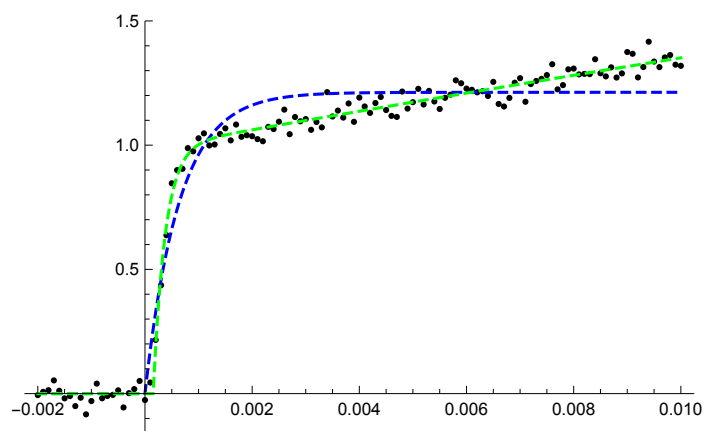
Bi: ratio  $\chi^2_{\text{Mono}}/\chi^2_{\text{Bi}} = 1.10818$        $d = 0.000183393$        $a =$   
 $19.0487$        $a1 = 0.957703$        $t1 = 0.000305464$        $t2 = 0.41421$

----- D

take bi

take bi

take bi



Mono:  $\chi^2 = 0.801939$        $d = 0.801939$        $a = 1.2127$        $t = 0.000633724$

Bi: ratio  $\chi^2_{\text{Mono}}/\chi^2_{\text{Bi}} = 5.47494$        $d = 0.000153394$        $a =$   
 $4.99995$        $a1 = 0.991888$        $t1 = 0.000223056$        $t2 = 0.104748$

-----

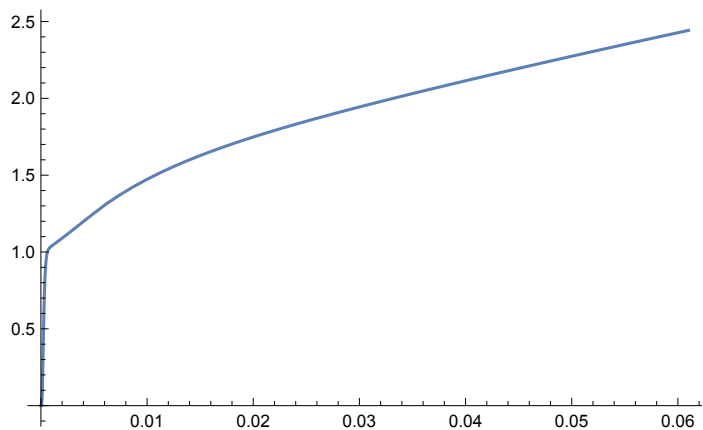
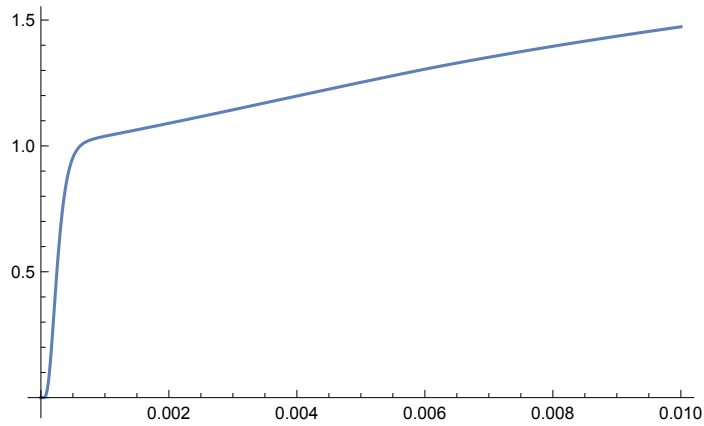
-----

----- Ca = 30.4385

uM -----

-----

-----



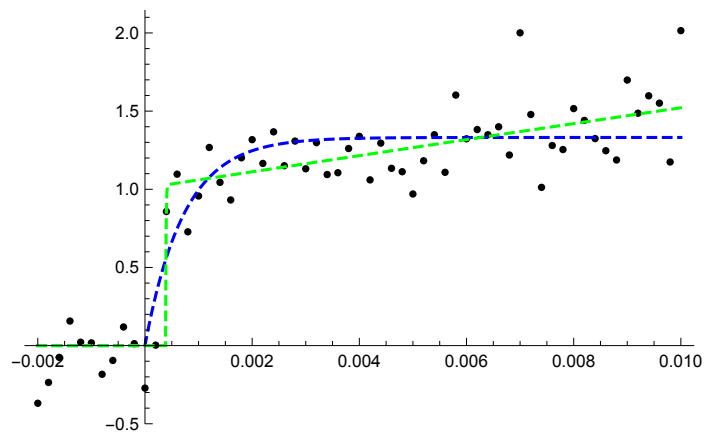
C5

```
take mono
```

```
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 = 2.85294      d = 2.85294      a = 1.33147      t = 0.000726217
```

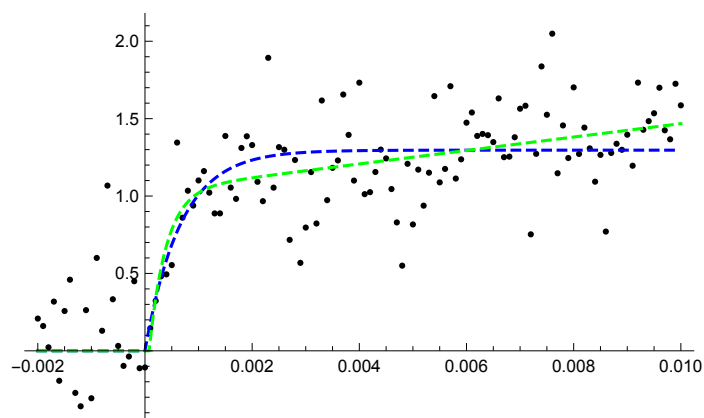
```
Bi: ratio chi2Mono/chi2Bi = 1.30575      d = 0.000384815      a =  
19.3501      a1 = 1.02791      t1 = 8.47328 × 10-6      t2 = 0.352122
```

C10

```
take mono
```

take mono

take mono



Mono:  $\chi^2 = 10.5612$        $d = 10.5612$        $a = 1.29638$        $t = 0.000664938$

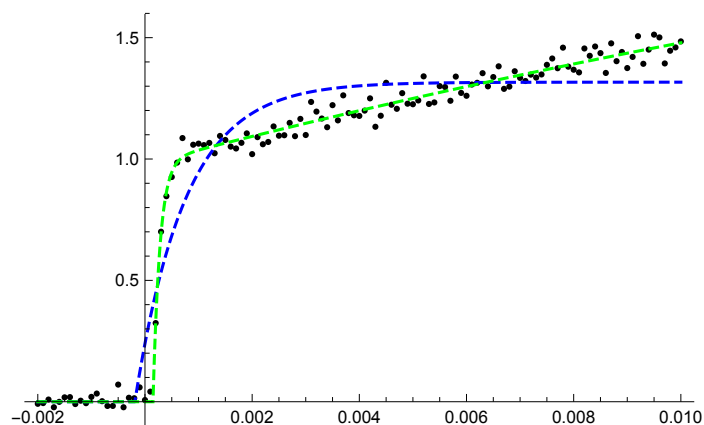
Bi: ratio  $\chi^2_{\text{Mono}}/\chi^2_{\text{Bi}} = 1.09687$        $d = 0.000077443$        $a =$   
 21.7689       $a1 = 1.03527$        $t1 = 0.000315449$        $t2 = 0.470737$

----- D

take bi

take bi

take bi



Mono:  $\chi^2 = 1.39955$        $d = 1.39955$        $a = 1.31664$        $t = 0.000939448$

Bi: ratio  $\chi^2_{\text{Mono}}/\chi^2_{\text{Bi}} = 8.83763$        $d = 0.000149554$        $a =$   
 2.78601       $a1 = 0.988914$        $t1 = 0.000128252$        $t2 = 0.0309468$

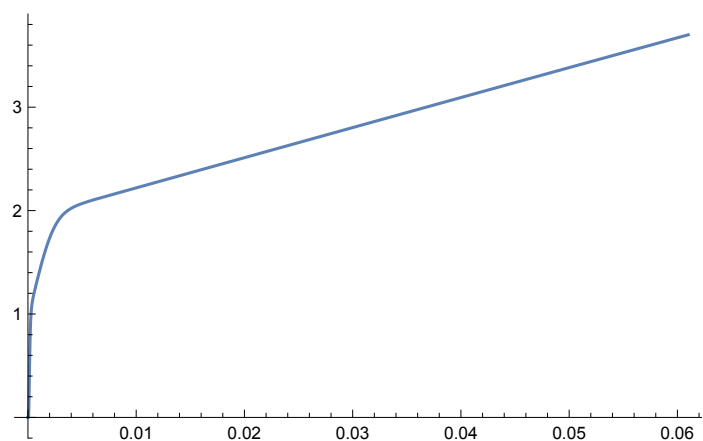
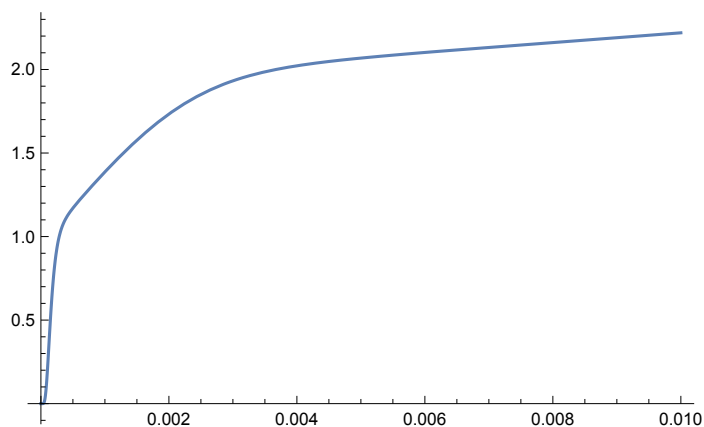
-----

----- Ca = 74.5861

uM -----

-----

-----

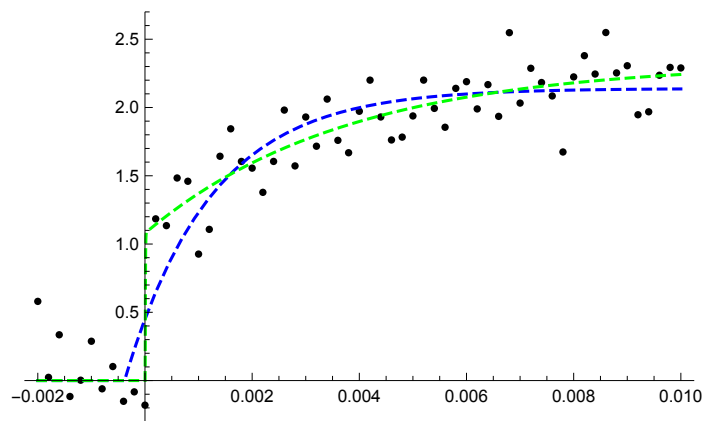


C5

```
take mono
```

```
take bi
```

```
take bi
```



```
Mono: chi2 = 3.78636      d = 3.78636      a = 2.13923      t = 0.00160829
```

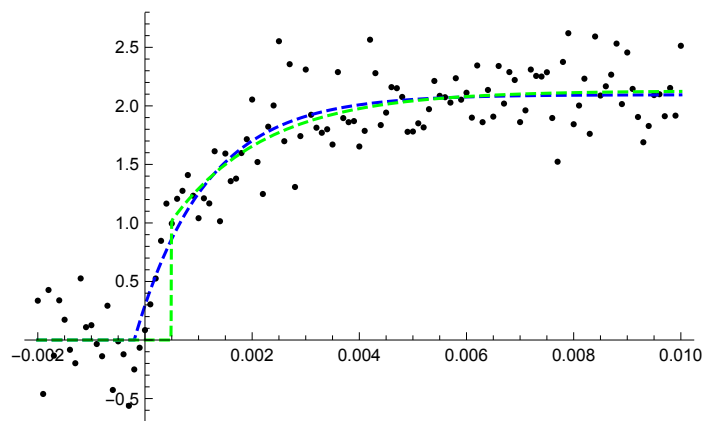
```
Bi: ratio chi2Mono/chi2Bi = 1.49132      d = 4.60679 × 10-6      a =  
2.32967      a1 = 1.08023      t1 = 3.89614 × 10-6      t2 = 0.00376319
```

C10

```
take mono
```

```
take bi
```

```
take mono
```



Mono:  $\chi^2 = 8.25029$        $d = 8.25029$        $a = 2.09501$        $t = 0.00131592$

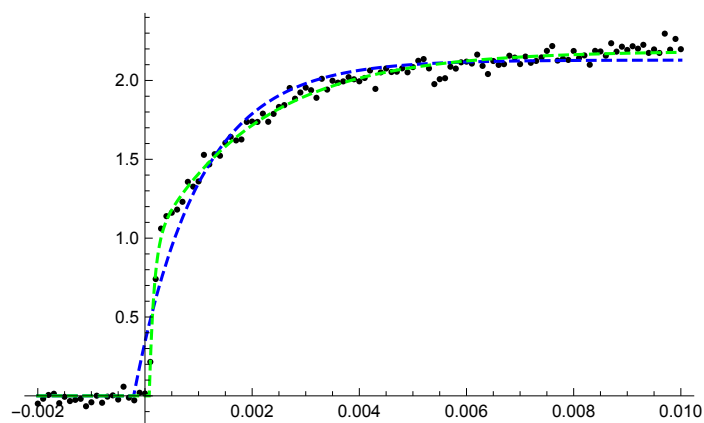
Bi: ratio  $\chi^2_{\text{Mono}}/\chi^2_{\text{Bi}} = 0.803446$        $d = 0.00048757$        $a =$   
 $2.12862$        $a1 = 1.01187$        $t1 = 3.35749 \times 10^{-6}$        $t2 = 0.00176021$

----- D

take bi

take bi

take bi



Mono:  $\chi^2 = 0.883126$        $d = 0.883126$        $a = 2.12918$        $t = 0.00120572$

Bi: ratio  $\chi^2_{\text{Mono}}/\chi^2_{\text{Bi}} = 4.48386$        $d = 0.0000795971$        $a =$   
 $2.18666$        $a1 = 0.952218$        $t1 = 0.0000883538$        $t2 = 0.0019903$

## Plots

In[405]:= **caFact = 1\*^6;**

In[406]:= **colorA = Green;**

**colorB = Red;**

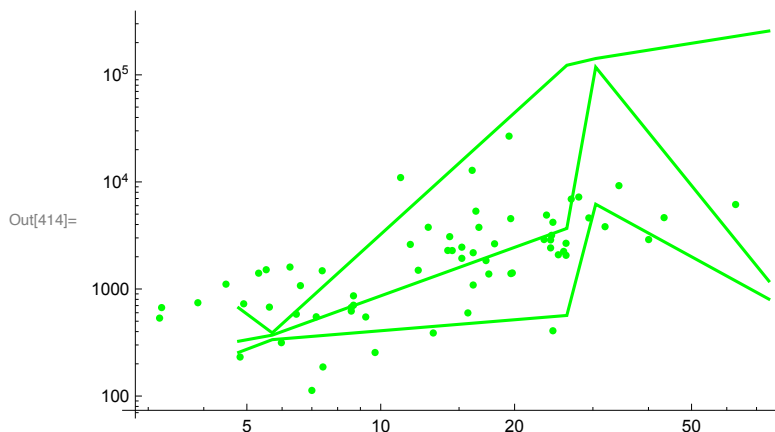
**colorC = Blue;**

## release rate $1/\tau_1$ (merge of mono $1/\tau_1$ and bi $1/\tau_1$ )

In[409]:= `simParam = 15;`

### C5

```
In[410]:= 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"];
];
```



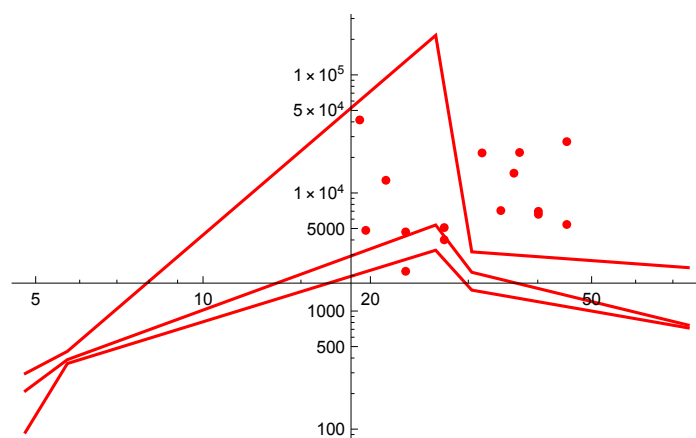
## C10

```

In[416]:= 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[420]=



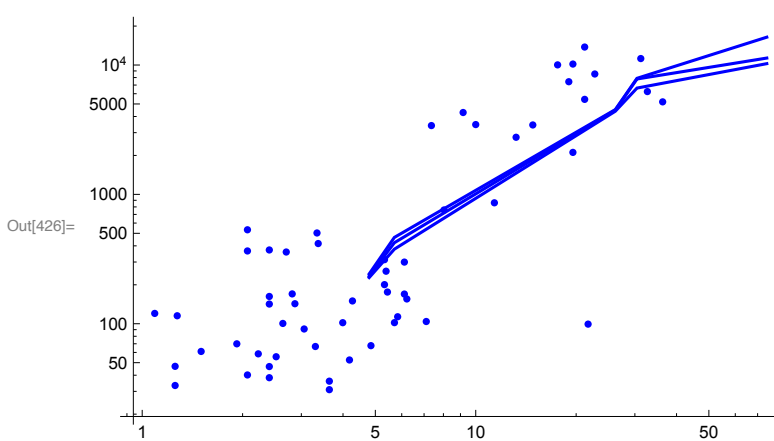


D

```

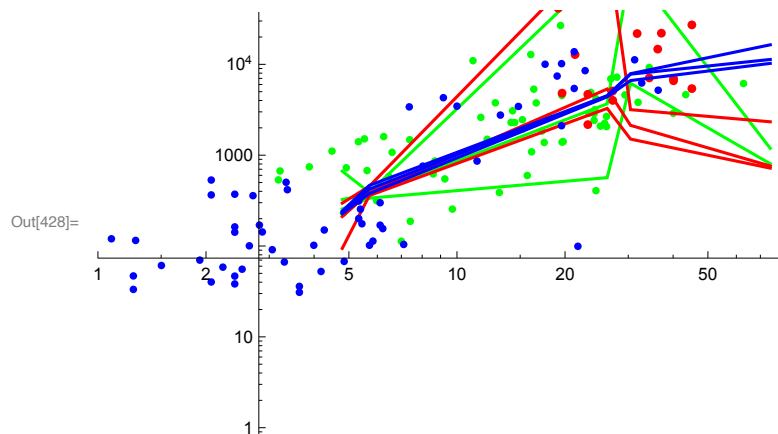
In[422]:= gr1c =
  ListLogLogPlot[Transpose[{dataT1DCa, dataT1DRelRate}], PlotStyle → {colorC}];
gr2c = ListLogLogPlot[Transpose[
  {caFact simCaList, simParamMedianD[[simParam, All]]}],
  PlotStyle → {colorC}, Joined → True, PlotRange → All];
gr3c = ListLogLogPlot[Transpose[{caFact simCaList,
  simParamQuantile1D[[simParam, 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"];
];

```



## C5 and C10 and D

```
In[428]:= Show[gr1a, gr2a, gr3a, gr4a, gr1b, gr2b, gr3b,
  gr4b, gr1c, gr2c, gr3c, gr4c, PlotRange -> {All, {0, 10}}]
```



```
In[429]:= Show[gr1a, gr1b, gr1c, PlotRange -> {All, {2, 10}}];
```

## delay (mono and bi merged)

```
In[430]:= simParam = 12;
```

```
In[431]:= Transpose[{caFact simCaList, simParamMedianC5[[simParam, All]]}] // TableForm
```

Out[431]//TableForm=

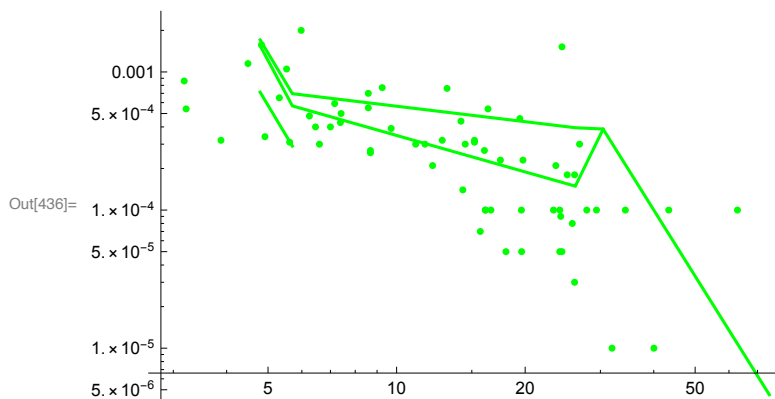
0.703073	
4.79194	0.0015475
5.70182	0.000567899
26.2105	0.000149017
30.4385	0.000384815
74.5861	-0.000183647

## C5

```

In[432]:= 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"];
];

```



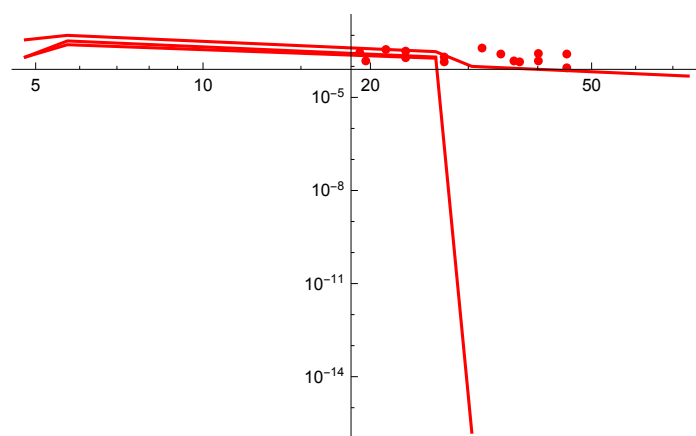
## C10

```

In[438]:= gr1b = ListLogLogPlot[
  Transpose[{dataT1C10Ca, dataT1C10Delay}], PlotStyle → {colorB}];
gr2b = ListLogLogPlot[Transpose[{caFact simCaList,
  simParamMedianC10[[simParam, All]]}],
  PlotStyle → {colorB}, Joined → True, PlotRange → All];
gr3b = ListLogLogPlot[Transpose[{caFact simCaList,
  simParamQuantile1C10[[simParam, All]]}],
  PlotStyle → {colorB}, Joined → True, PlotRange → All];
gr4b = ListLogLogPlot[Transpose[{caFact simCaList,
  simParamQuantile2C10[[simParam, All]]}],
  PlotStyle → {colorB}, Joined → True, PlotRange → All];
Show[gr1b, gr2b, gr3b, gr4b, PlotRange → All]
If[exportYes == 1,
  Export["plot delay C10 data.txt",
    Transpose[{dataT1C10Ca, dataT1C10Delay}], "Table"];
toExport = Transpose[{caFact simCaList, simParamQuantile1C10[[simParam, All]],
  simParamMedianC10[[simParam, All]],
  simParamQuantile2C10[[simParam, All]]}];
Export["plot delay C10 fit - quantiles and median.txt", toExport, "Table"];
];

```

Out[442]=

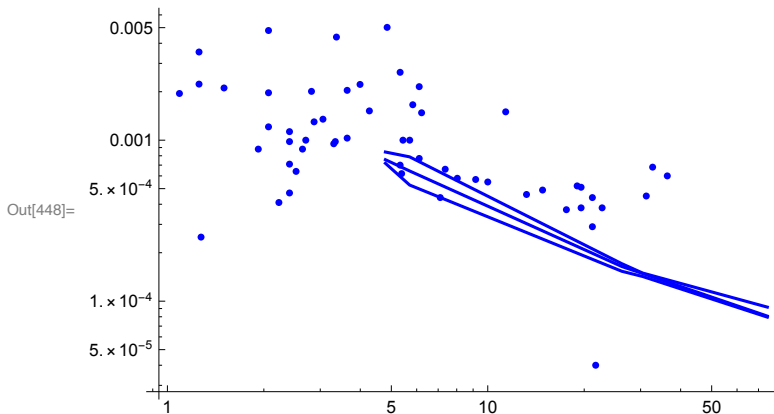


D

```

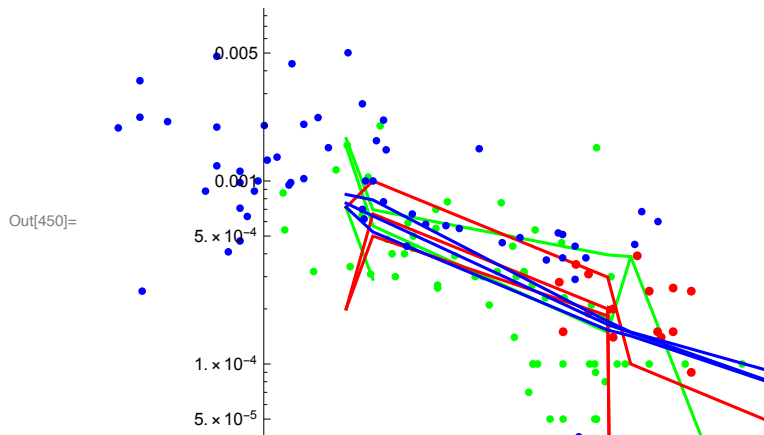
In[444]:= gr1c =
  ListLogLogPlot[Transpose[{dataT1DCa, dataT1DDelay}], PlotStyle → {colorC}];
gr2c = ListLogLogPlot[Transpose[
  {caFact simCaList, simParamMedianD[[simParam, All]]}],
  PlotStyle → {colorC}, Joined → True, PlotRange → All];
gr3c = ListLogLogPlot[Transpose[{caFact simCaList,
  simParamQuantile1D[[simParam, 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"];
];

```



## C5 and C10 and D

```
In[450]:= Show[gr1a, gr2a, gr3a, gr4a, gr1b, gr2b, gr3b,
  gr4b, gr1c, gr2c, gr3c, gr4c, PlotRange → {-10, -5}]
```



```
In[451]:= Show[gr1a, gr1b, gr1c, PlotRange → All];
```

---

## amp (merge of mono amp and bi amp)

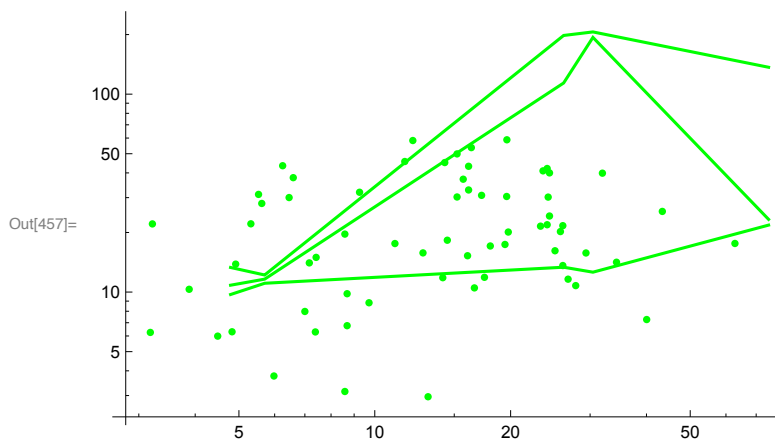
```
In[452]:= simParam = 13;
```

## C5

```

In[453]:= 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 ]

```

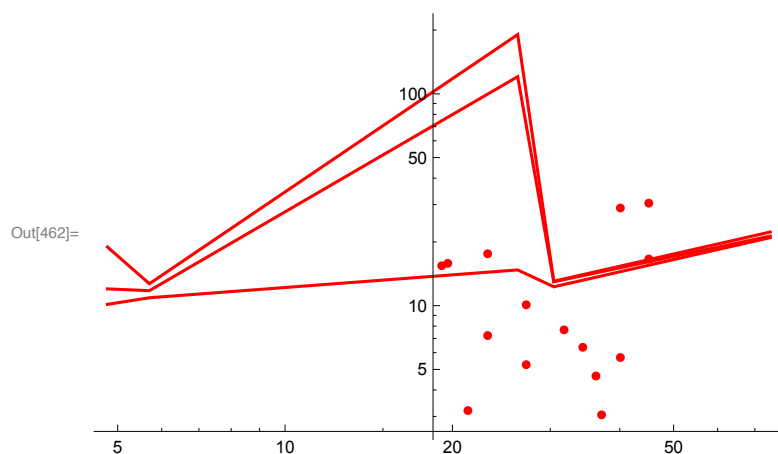


## C10

```

In[458]:= 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 ]

```



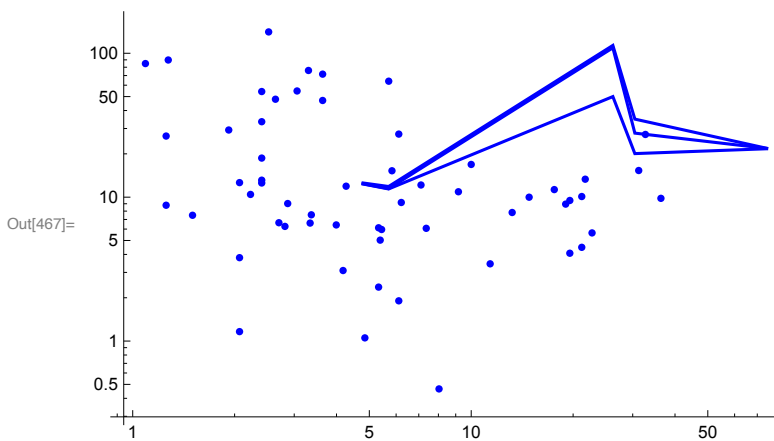


D

```

In[463]:= 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 ]

```

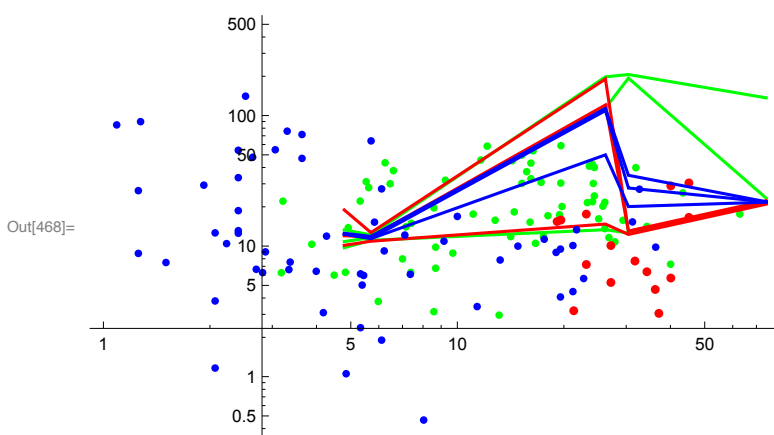


C5 and C10 and D

```

In[468]:= Show[gr1a, gr2a, gr3a, gr4a, gr1b, gr2b, gr3b,
  gr4b, gr1c, gr2c, gr3c, gr4c, PlotRange → {-1, 6} ]

```



release rate  $1/\tau^2$  of bi fits (if bi is justified)

```

In[469]:= simParam = 16;

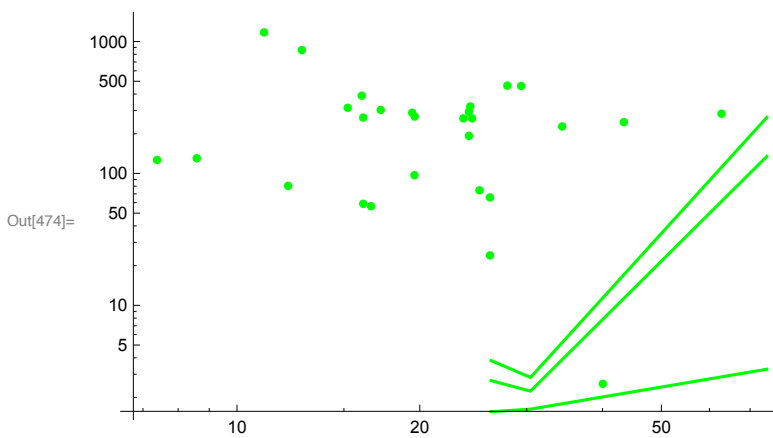
```

## C5

```

In[470]:= 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"];
];

```

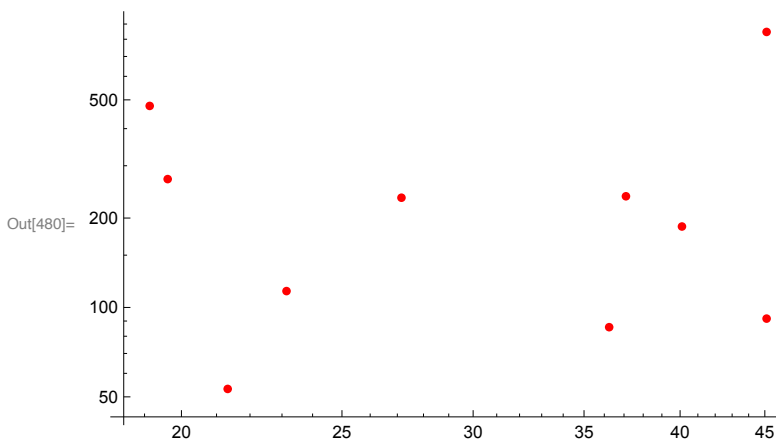


## C10

```

In[476]:= 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"];
];

```

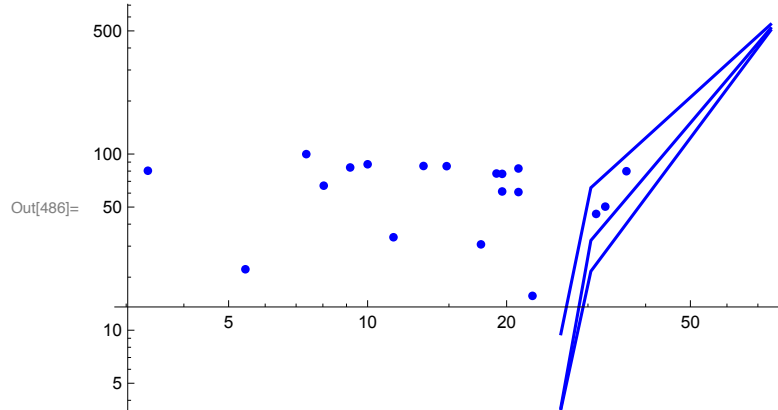


D

```

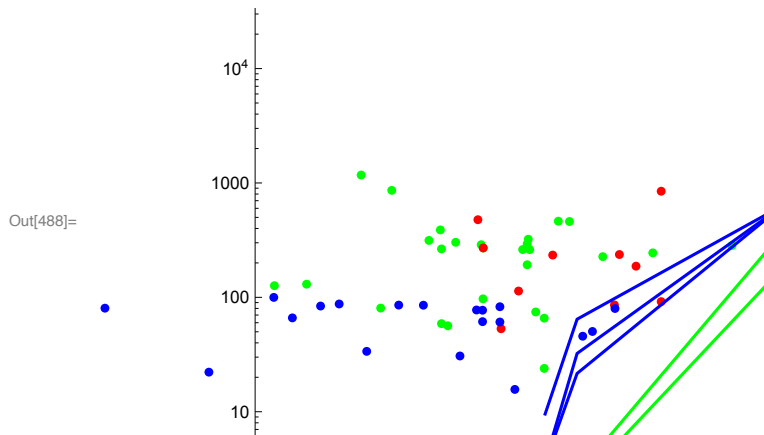
In[482]:= gr1c =
  ListLogLogPlot[Transpose[{dataT2DCa, dataT2DRelRate}], PlotStyle → {colorC}];
gr2c = ListLogLogPlot[Transpose[
  {caFact simCaList, simParamMedianD[[simParam, All]]}],
  PlotStyle → {colorC}, Joined → True, PlotRange → All];
gr3c = ListLogLogPlot[Transpose[{caFact simCaList,
  simParamQuantile1D[[simParam, 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"];
];

```



## C5 and C10 and D

```
In[488]:= Show[gr1a, gr2a, gr3a, gr4a, gr1b, gr2b, gr3b,  
gr4b, gr1c, gr2c, gr3c, gr4c, PlotRange -> {All, {2, 10}}]
```



```
In[489]:= Show[gr1a, gr1b, gr1c, PlotRange -> All];
```

---

## amp1 of bi fits (if bi is justified)

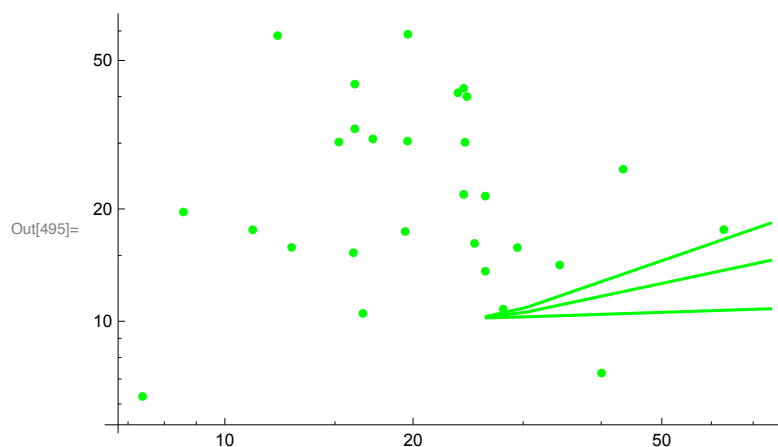
```
In[490]:= simParam = 14;
```

## C5

```

In[491]:= 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 ]

```

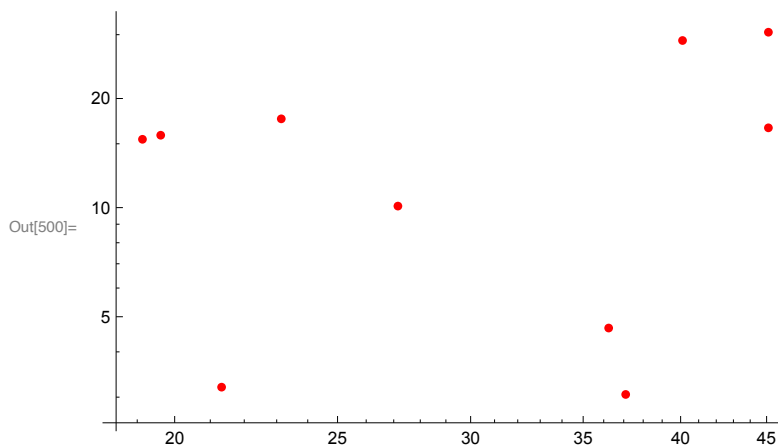


## C10

```

In[496]:= 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 ]

```

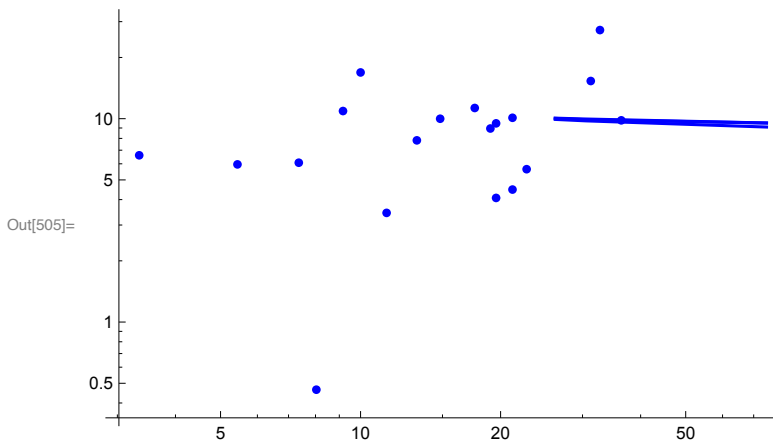


## D

```

In[501]:= 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]

```

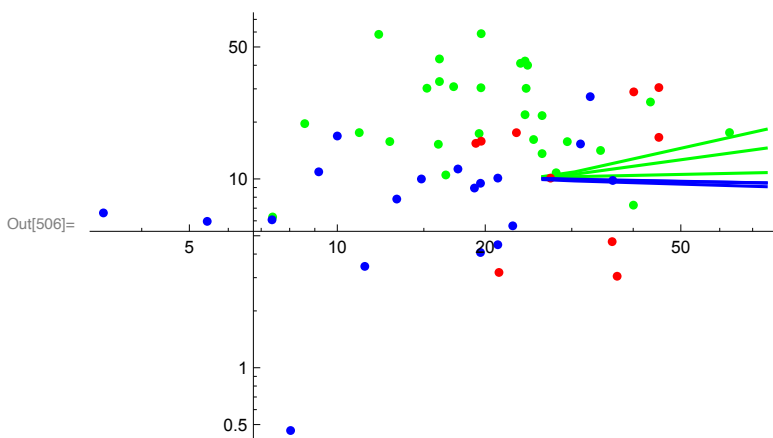


## C5 and C10 and D

```

In[506]:= Show[gr1a, gr2a, gr3a, gr4a, gr1b, gr2b,
  gr3b, gr4b, gr1c, gr2c, gr3c, gr4c, PlotRange → All]
Show[gr1a, gr1b, gr1c, PlotRange → All];

```





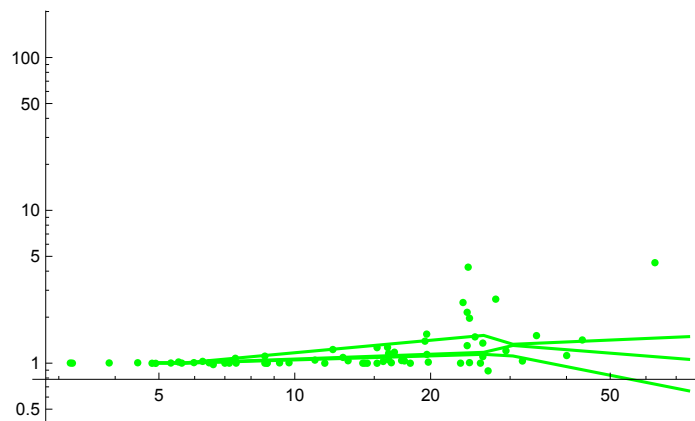
## chi2 mono/bi ratio

In[508]:= `simParam = 6;`

### C5

```
In[509]:= 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[513]=



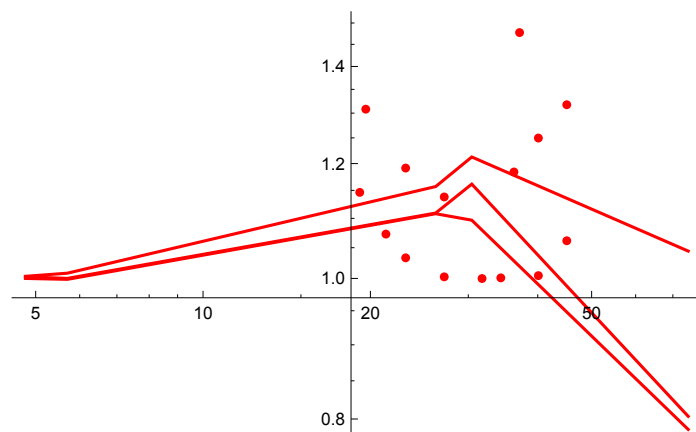
## C10

```

In[515]:= 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[519]=

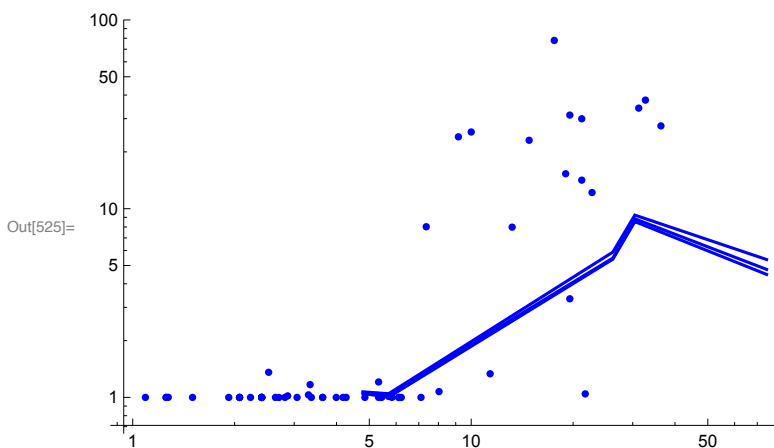


D

```

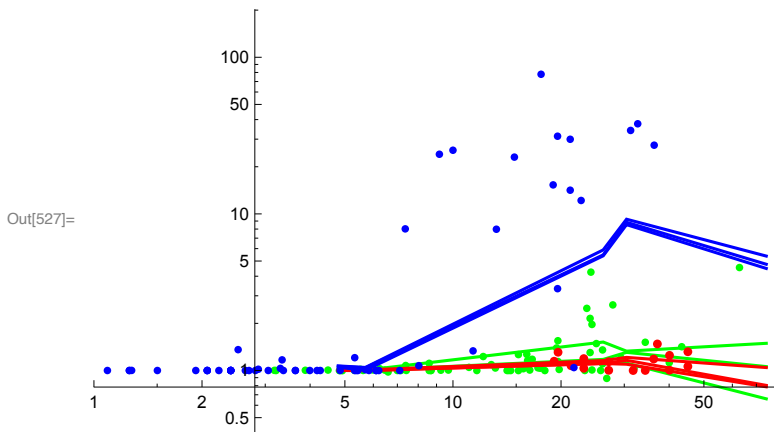
In[521]:= gr1c =
  ListLogLogPlot[Transpose[{dataT1DCa, dataT1DChiRatio}], PlotStyle → {colorC}];
gr2c = ListLogLogPlot[Transpose[
  {caFact simCaList, simParamMedianD[[simParam, All]]}],
  PlotStyle → {colorC}, Joined → True, PlotRange → All];
gr3c = ListLogLogPlot[Transpose[{caFact simCaList,
  simParamQuantile1D[[simParam, 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"];
];

```



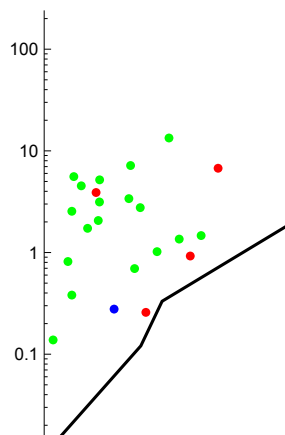
## C5 and C10 and D

```
In[527]:= Show[gr1a, gr2a, gr3a, gr4a, gr1b, gr2b, gr3b,
  gr4b, gr1c, gr2c, gr3c, gr4c, PlotRange → {All, {-0.8, 5}}]
Show[gr1a, gr1b, gr1c, PlotRange → All];
```

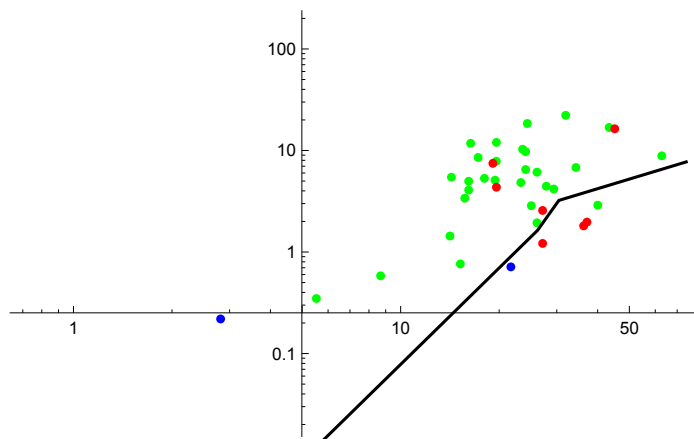


## Nv

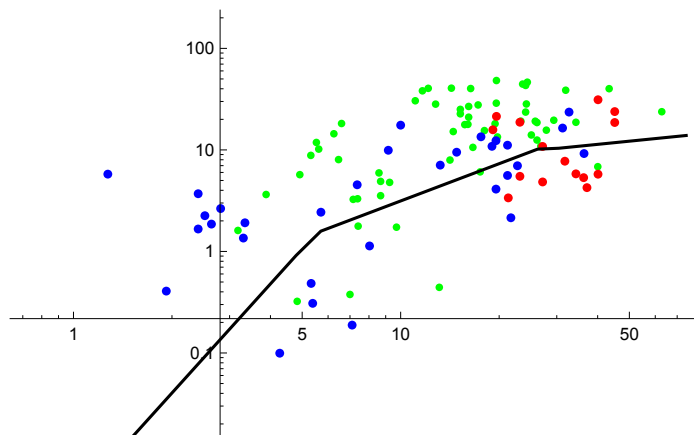
```
In[529]:= For[NvCount = 1, NvCount ≤ 7, NvCount += 1,
  Print[" time for Nv (ms) = ", 1000 * timeOfNv[NvCount]];
  gr1a = ListLogLogPlot[
    Transpose[{dataT1C5Ca, dataT1C5Nv[NvCount]}], PlotStyle → {colorA}];
  gr1b = ListLogLogPlot[Transpose[{dataT1C10Ca, dataT1C10Nv[NvCount]}],
    PlotStyle → {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
```



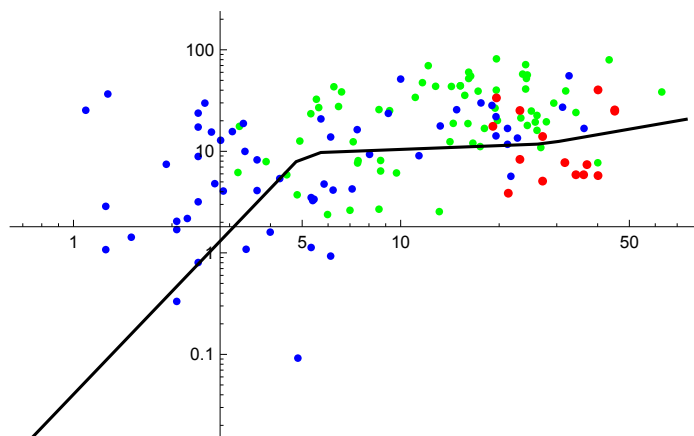
```
time for Nv (ms) = 0.2
```



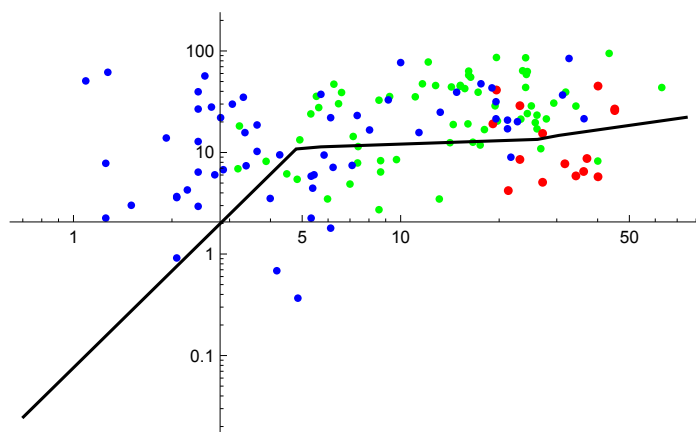
time for Nv (ms) = 1.



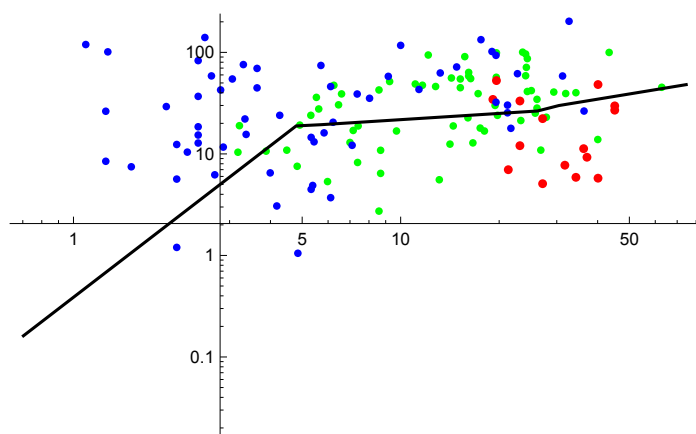
time for Nv (ms) = 5.



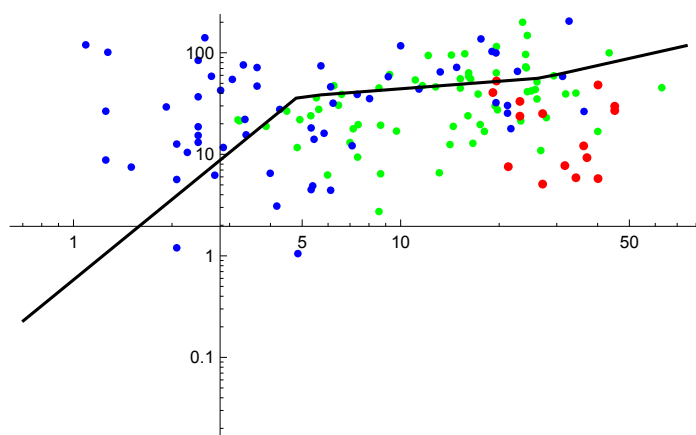
time for Nv (ms) = 10.



time for Nv (ms) = 100.



time for Nv (ms) = 400.



## sustained release 10 to 100 ms

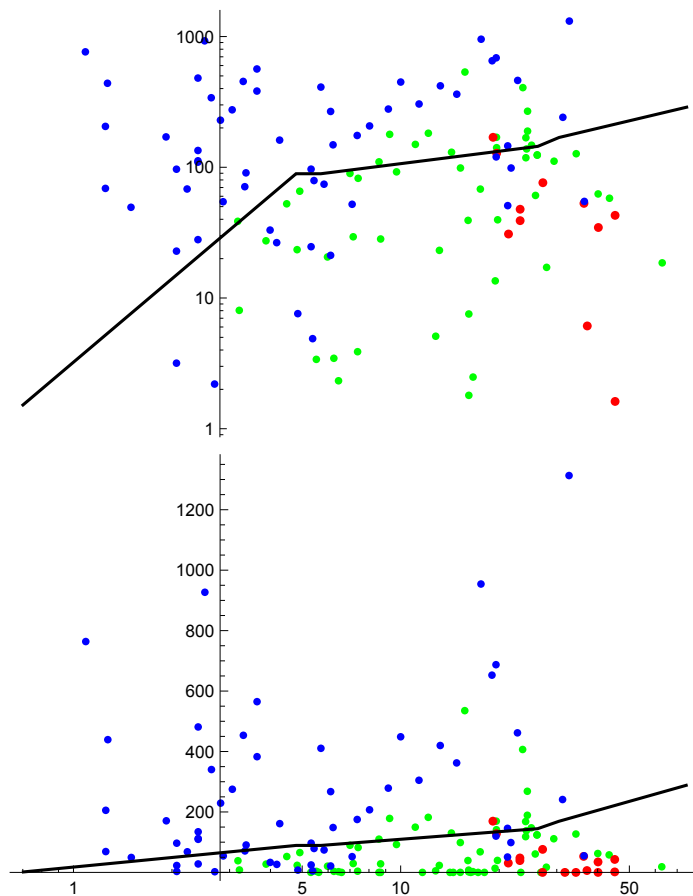
```

In[530]:= 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[538]:= If[exportYes == 1,
  Export["plot sustained release Cm5 data.txt", ttt1, "Table"];
  Export["plot sustained release Cm10 data.txt", ttt2, "Table"];
  Export["plot sustained release D data.txt", ttt3, "Table"];
  Export["plot sustained release sim.txt", ttt4, "Table"]
];
```

## Nv normalize to the value at 5 ms

```
In[539]:= timeOfNv
nromPos = 5;
1000 * timeOfNv[[nromPos]]
```

```
Out[539]= {0.0001, 0.0002, 0.001, 0.005, 0.01, 0.1, 0.4}
```

```
Out[541]= 10.
```

```
In[542]:= dataT1DNvNorm = Transpose[Transpose[dataT1DNv]/dataT1DNv[[nromPos]]];
dataT1C10NvNorm = Transpose[Transpose[dataT1C10Nv]/dataT1C10Nv[[nromPos]]];
dataT1C5NvNorm = Transpose[Transpose[dataT1C5Nv]/dataT1C5Nv[[nromPos]]];
```

```
In[545]:= dataT1C5NvNorm // TableForm
```

```
Out[545]//TableForm=
```

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

```
In[546]:= simParamNv // TableForm
```

```
simParamNvNorm = Transpose[Transpose[simParamNv]/simParamNv[[nromPos]]];
simParamNvNorm // TableForm
```

```
Out[546]//TableForm=
```

$1.15733 \times 10^{-8}$	0.0000223798	0.0000472366	0.0120748	0.0331758	0.189668
$4.0267 \times 10^{-7}$	0.000687618	0.0013287	0.163593	0.322237	0.77187
0.0000966667	0.0913074	0.158397	1.01743	1.03842	1.38687
0.00123275	0.791128	0.973861	1.17704	1.2524	2.06825
0.00248568	1.08299	1.13647	1.34602	1.47317	2.21951
0.0161613	1.8861	1.93952	2.64701	2.99457	4.81257
0.0229784	3.57675	3.85026	5.60294	6.15268	11.7524

```
Out[548]//TableForm=
```

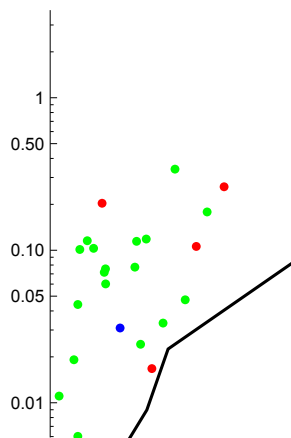
$4.656 \times 10^{-6}$	0.0000206648	0.0000415644	0.00897072	0.02252	0.0854549
0.000161996	0.000634927	0.00116915	0.121538	0.218737	0.347767
0.0388895	0.0843107	0.139377	0.755884	0.704891	0.624855
0.495941	0.730505	0.856919	0.874458	0.850142	0.931851
1.	1.	1.	1.	1.	1.
6.50175	1.74157	1.70662	1.96655	2.03274	2.16831
9.24432	3.30267	3.38792	4.1626	4.17649	5.29506



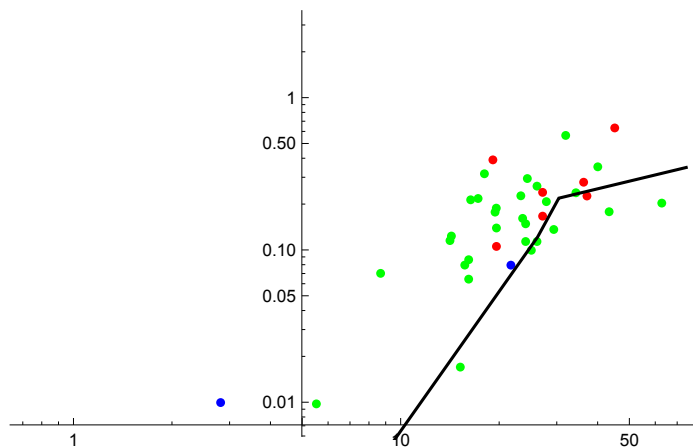
```

In[549]:= For[NvCount = 1, NvCount ≤ 7, NvCount += 1,
  Print[" time for Nv (ms) = ", 1000 * timeOfNv[NvCount]];
  gr1a = ListLogLogPlot[
    Transpose[{dataT1C5Ca, dataT1C5NvNorm[NvCount]}], PlotStyle → {colorA}];
  gr1b = ListLogLogPlot[Transpose[{dataT1C10Ca, dataT1C10NvNorm[NvCount]}],
    PlotStyle → {colorB}];
  gr1c = ListLogLogPlot[Transpose[{dataT1DCa, dataT1DNvNorm[NvCount]}],
    PlotStyle → {colorC}];
  gr2 = ListLogLogPlot[Transpose[{caFact simCaList, simParamNvNorm[NvCount,
    All]}], PlotStyle → {Black}, Joined → True, PlotRange → All];
  Show[gr1a, gr1b, gr1c, gr2, PlotRange → {All, {-5, 1}}] // Print;
];
time for Nv (ms) = 0.1

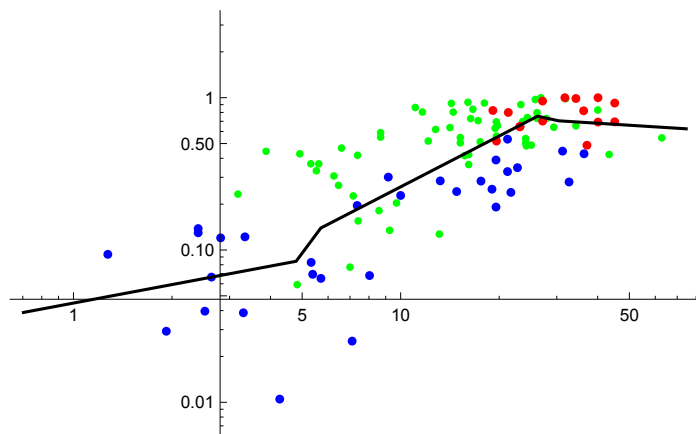
```



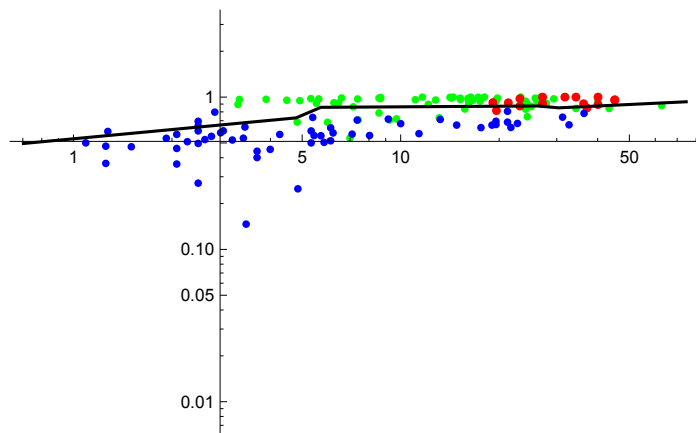
```
time for Nv (ms) = 0.2
```



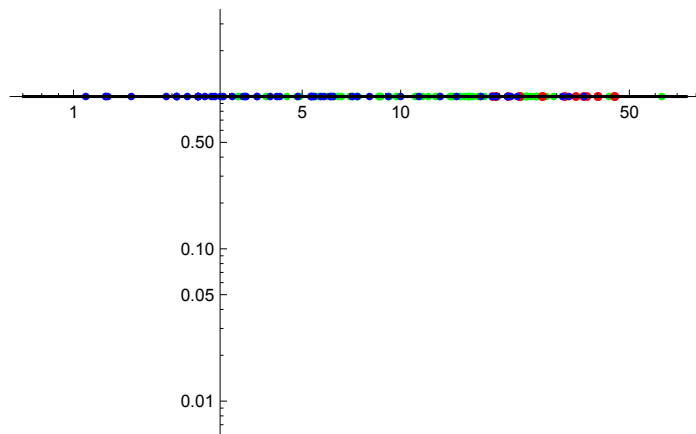
```
time for Nv (ms) = 1.
```



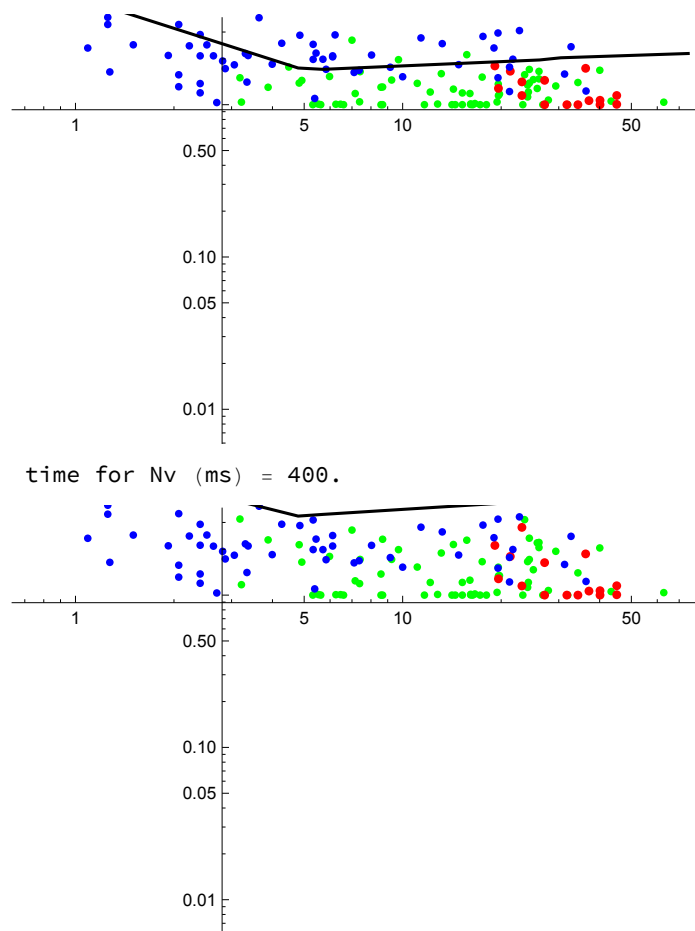
time for Nv (ms) = 5.



time for Nv (ms) = 10.



time for Nv (ms) = 100.



## Export Nv

```
In[550]:= 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[551]:= Transpose[simParamMedianC5] // TableForm
Transpose[simParamQuantile1C5] // TableForm
Transpose[simParamQuantile2C5] // TableForm
```

Out[551]//TableForm=

0	2.5573	0.0015475	1.08303	325.139	1.00194	0.001514	0.
0	2.50461	0.000567899	1.16243	370.508	1.0028	0.000574885	2.
0	2.59839	0.000363778	1.22787	6471.19	1.17733	0.00032778	19
0	3.2711	0.000180307	1.26031	5693.99	1.30575	0.000386272	20
0	3.27193	-0.000267639	2.13923	804.964	1.05917	$4.60679 \times 10^{-6}$	2.

Out[552]//TableForm=

0	2.51606	0.000713634	0.971947	255.849	1.00121	0.001
0	2.47782	0.000290398	1.1077	336.311	1.00004	0.000370448
0	2.49861	-0.0004	1.17334	565.925	1.14234	0.000149017
0	2.85294	$9.32334 \times 10^{-17}$	1.25169	1377.	1.11441	0.000384815
0	2.85416	-0.000380157	2.10942	621.78	0.659316	-0.000183647

Out[553]//TableForm=

0	2.96309	0.00170589	1.33129	665.904	1.00705	0.00168772
0	2.75443	0.000696149	1.21975	389.051	1.0049	0.000663083
0	2.70432	0.000399859	1.33533	$3.93924 \times 10^6$	1.51932	0.000394608
0	3.43923	0.000198141	1.33147	6190.3	1.32917	0.000394903
0	3.78636	-0.000255979	2.1793	831.044	1.49132	0.000588848

## C10

```
In[554]:= Transpose[simParamMedianC10] // TableForm
Transpose[simParamQuantile1C10] // TableForm
Transpose[simParamQuantile2C10] // TableForm
```

Out[554]//TableForm=

0	7.67019	0.0002	1.20006	210.934	1.00267	0.0002	1
0	7.83462	0.000660373	1.17692	387.588	1.00005	0.000672742	1
0	8.86808	-0.0001	1.23306	1219.14	1.10937	0.0002	1
0	8.52541	$1.62524 \times 10^{-16}$	1.29638	2129.89	1.16152	0.000077443	2
0	8.25029	-0.000296348	2.12344	719.972	0.803446	0.00048757	2

Out[555]//TableForm=

0	7.13307	0.0002	1.01403	94.1928	1.	0.0002	
0	7.7142	0.0005	1.08961	359.436	0.998535	0.0005	
0	6.98349	-0.000133562	1.22277	1021.09	1.10818	0.000183393	
0	7.78874	-0.0000334745	1.22767	1503.9	1.09687	0.0000409327	
0	6.56636	-0.000358176	2.09501	703.854	0.786879	0.000048708	

Out[556]//TableForm=

0	9.18484	0.000718121	1.89261	296.268	1.00335	0.0007	1.88
0	8.54551	0.001	1.26908	455.74	1.00837	0.00107778	15.6
0	10.2785	0.0002	1.24519	2142.35	1.15701	0.000298236	19.0
0	10.5612	0.0001	1.30379	3170.57	1.21262	0.000163589	36.0
0	10.0302	-0.0002	2.13033	759.926	1.04435	0.000497848	2.22

## D

```
In[557]:= Transpose[simParamMedianD] // TableForm
Transpose[simParamQuantile1D] // TableForm
Transpose[simParamQuantile2D] // TableForm
```

Out[557]//TableForm=

0	0.182211	0.000756457	1.23782	233.591	1.06421	0.000729688	0
0	0.161382	0.000643901	1.14624	422.665	1.04016	0.000581982	0
0	0.801939	0.0000430933	1.21328	1803.62	5.47494	0.00016417	1
0	1.30632	-0.000152542	1.31013	1131.46	8.83763	0.000148537	2
0	0.883126	-0.000194146	2.12802	845.567	4.76792	0.0000807884	2

Out[558]//TableForm=

0	0.175917	0.00071855	1.23443	227.397	1.04785	0.000679706	
0	0.148427	0.000526628	1.1405	377.03	1.00324	0.000473957	
0	0.78905	$-3.23016 \times 10^{-16}$	1.2127	1577.97	5.39822	0.000153394	
0	1.30204	-0.000188535	1.302	1064.45	8.53879	0.000142348	
0	0.803753	-0.000210221	2.12229	829.381	4.48386	0.0000795971	

Out[559]//TableForm=

0	0.192411	0.000844248	1.2548	242.66	1.07016	0.000816872	0
0	0.161997	0.000787771	1.18599	464.712	1.04593	0.000777558	1
0	0.817046	0.0000776317	1.21352	1882.15	5.88857	0.000170968	1
0	1.39955	-0.000105043	1.31664	1260.85	9.24529	0.000149554	3
0	0.917844	-0.000192623	2.12918	854.09	5.37949	0.0000918707	2

Nv

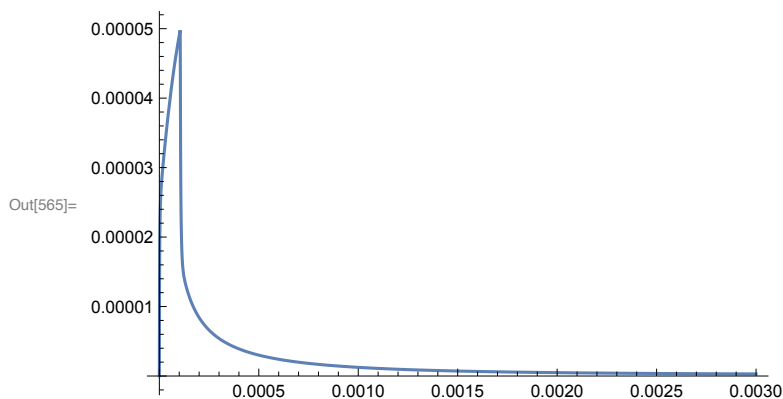
In[560]:= Transpose[simParamNv] // TableForm

Out[560]//TableForm=

$1.15733 \times 10^{-8}$	$4.0267 \times 10^{-7}$	0.0000966667	0.00123275	0.00248568	0.01616
0.0000223798	0.000687618	0.0913074	0.791128	1.08299	1.8861
0.0000472366	0.0013287	0.158397	0.973861	1.13647	1.93952
0.0120748	0.163593	1.01743	1.17704	1.34602	2.64701
0.0331758	0.322237	1.03842	1.2524	1.47317	2.99457
0.189668	0.77187	1.38687	2.06825	2.21951	4.81257

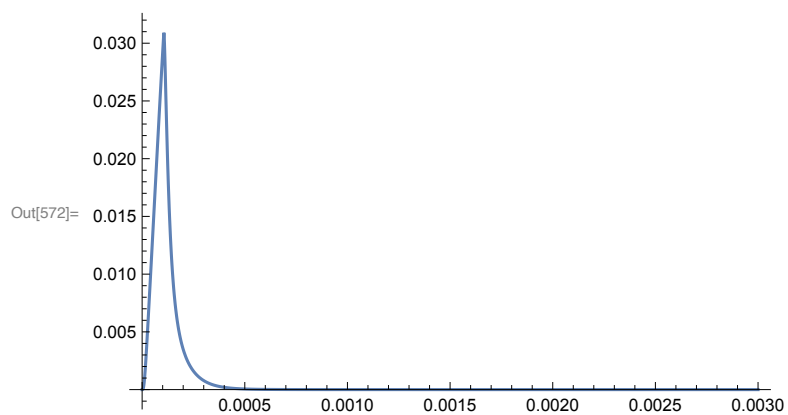
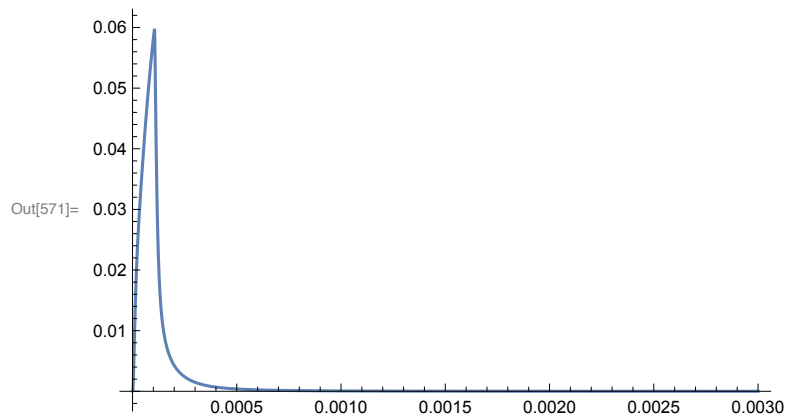
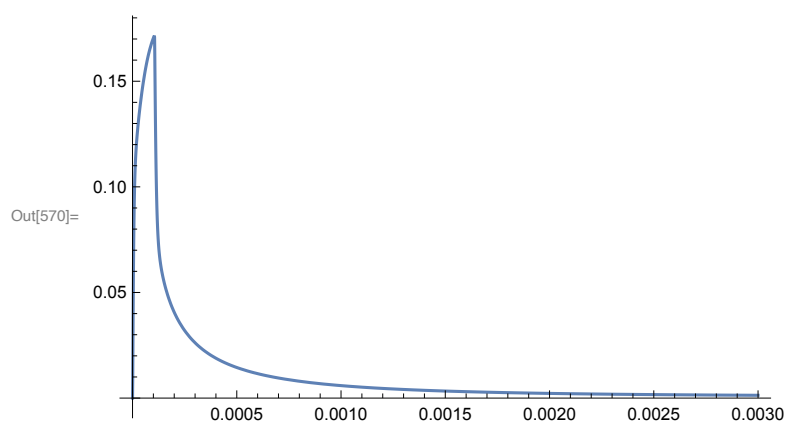
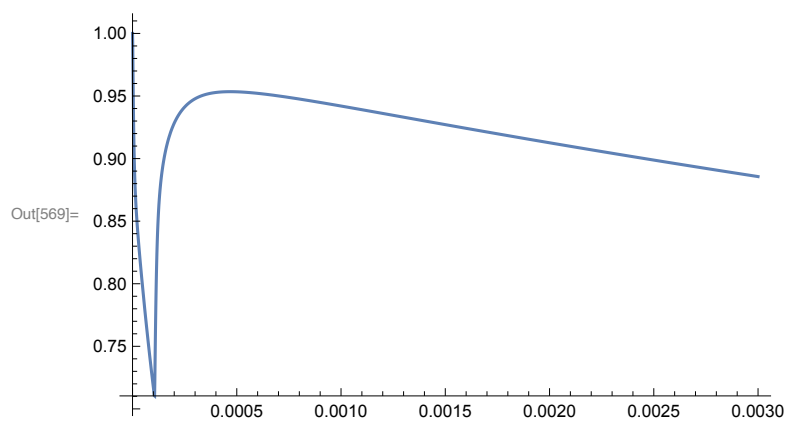
## EPSC with different caRest

```
In[561]:= 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]
```

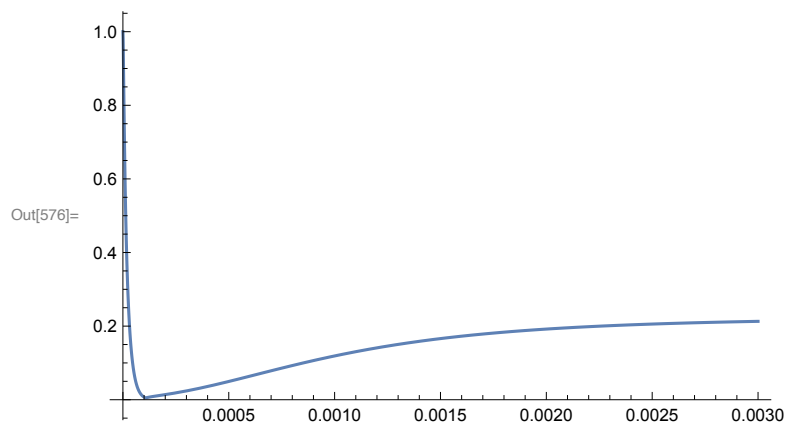
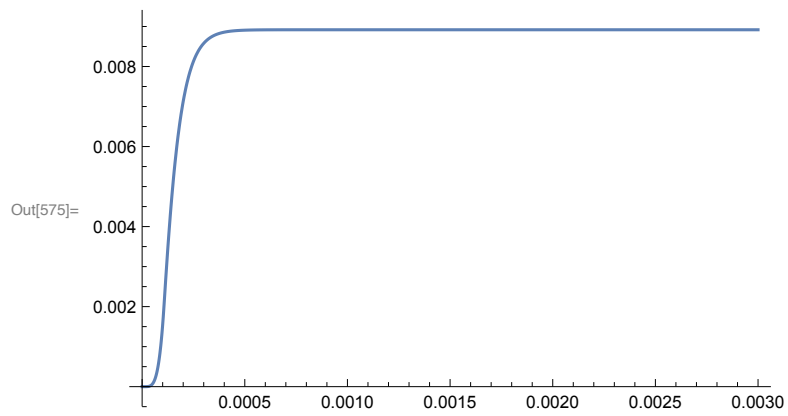
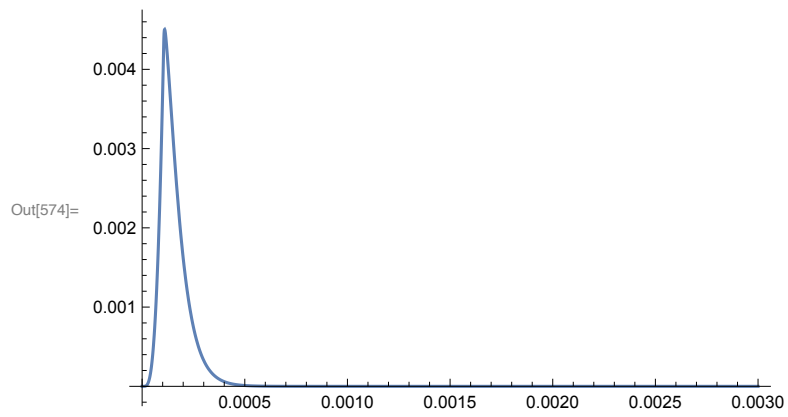
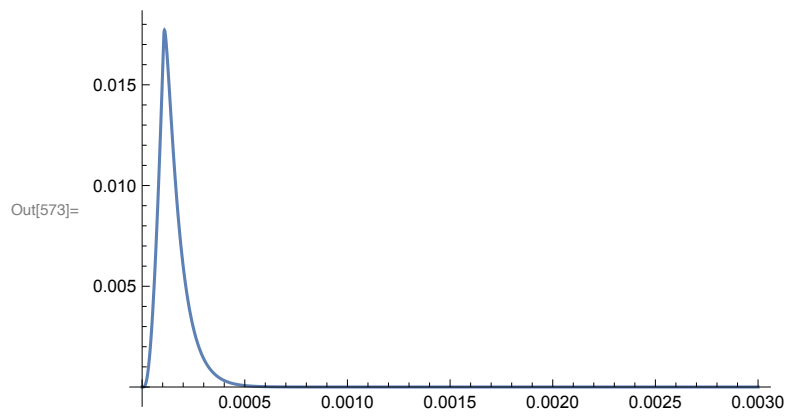


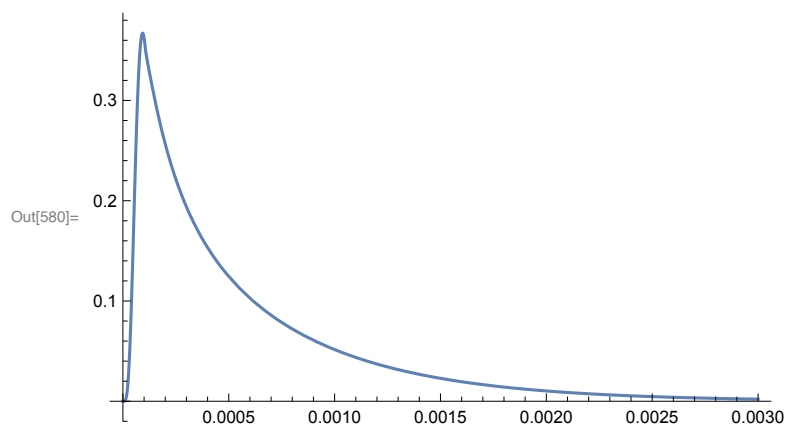
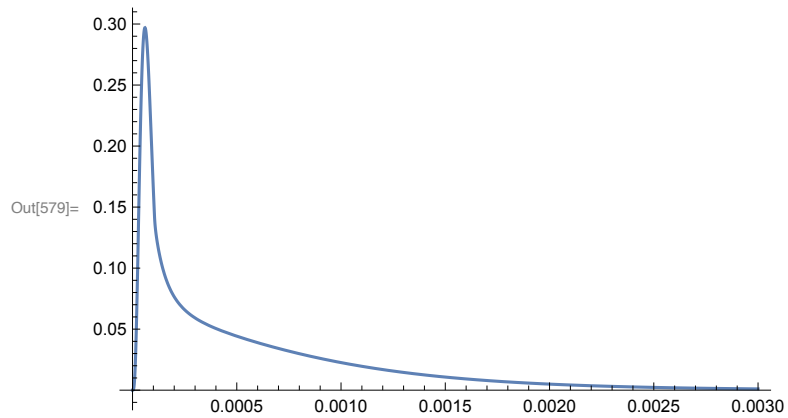
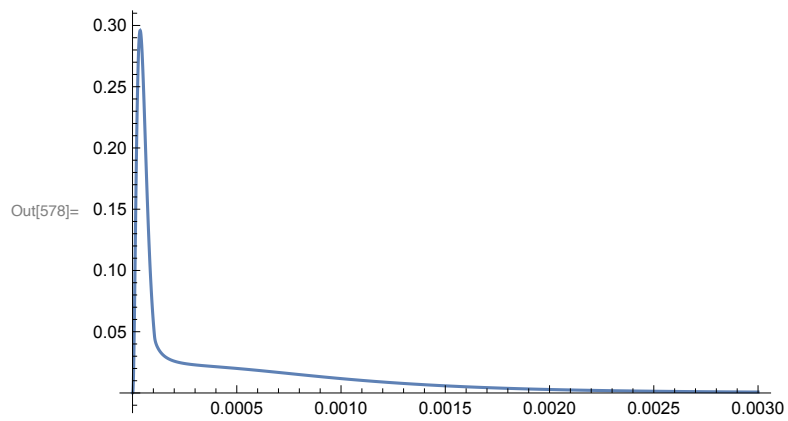
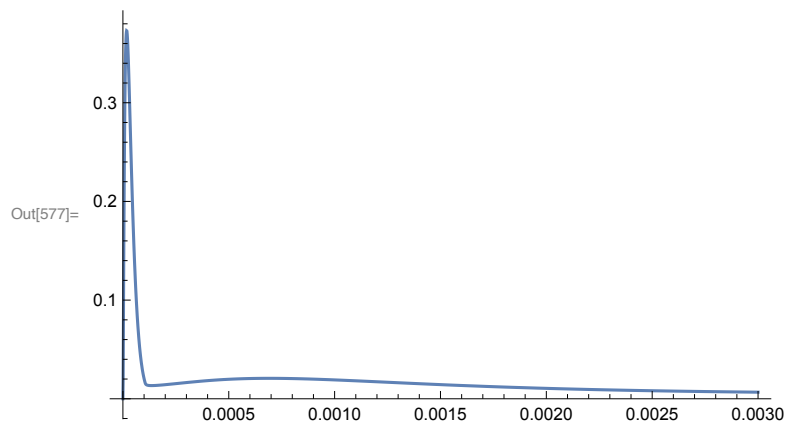
## NDSolve

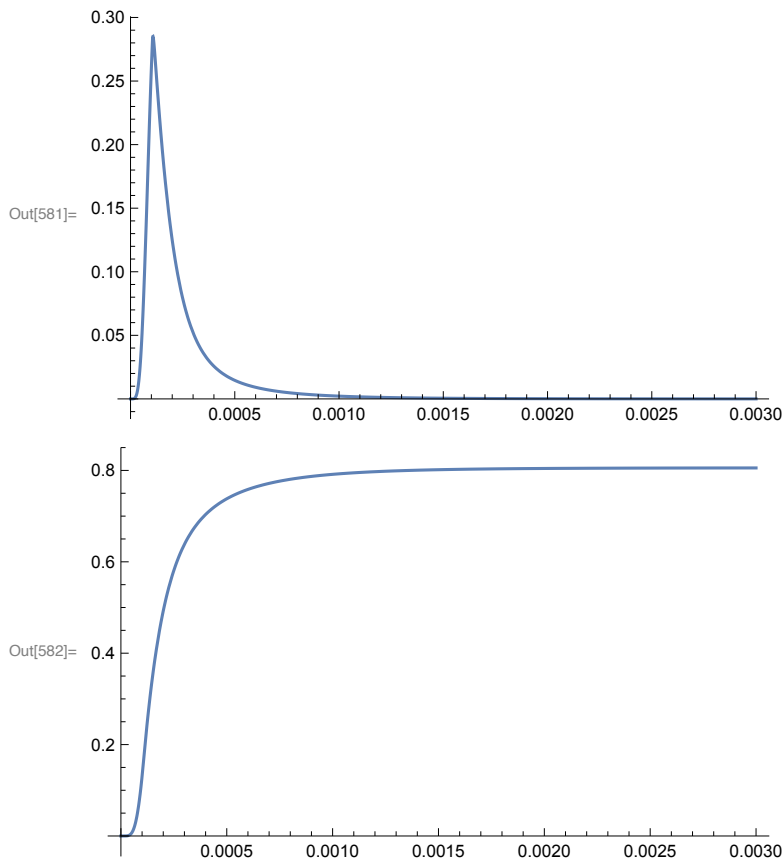
```
In[566]:= timeStartForPlot = 0.0;
timeEndForPlot = 0.003;
myNDSolveResults = NDSolve[eq, {ss1A, ss2A, ss3A, ss4A, ss5A, ss6A,
    ss7A, ss1B, ss2B, ss3B, ss4B, ss5B, ss6B, ss7B}, {t, 0, 0.003}];
Plot[(ss1A[t] /. myNDSolveResults), {t, timeStartForPlot, timeEndForPlot},
    PlotRange → All]
Plot[(ss2A[t] /. myNDSolveResults),
    {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss3A[t] /. myNDSolveResults),
    {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss4A[t] /. myNDSolveResults),
    {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss5A[t] /. myNDSolveResults),
    {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss6A[t] /. myNDSolveResults),
    {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss7A[t] /. myNDSolveResults),
    {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss1B[t] /. myNDSolveResults),
    {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss2B[t] /. myNDSolveResults),
    {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss3B[t] /. myNDSolveResults),
    {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss4B[t] /. myNDSolveResults),
    {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss5B[t] /. myNDSolveResults),
    {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss6B[t] /. myNDSolveResults),
    {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss7B[t] /. myNDSolveResults),
    {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
```











## different caRest

```
In[583]:= caRestLow = 30*^-9;
          caRestHigh = 180*^-9;
```

## Low Ca

### Initial occupancy

```
In[585]:= (*calculate initial equilibrium occupancy*)
          caFunc[t_] := caRestLow;
          ss0AInitial = kprimSchemeA / kunprimSchemeA
          ss0BInitial = (kprimSchemeA / kunprimSchemeA) * (kprimSchemeB / kunprimSchemeB)
```

```
Out[586]= 1.
```

```
Out[587]= 0.26514
```

## Diff eq.

```

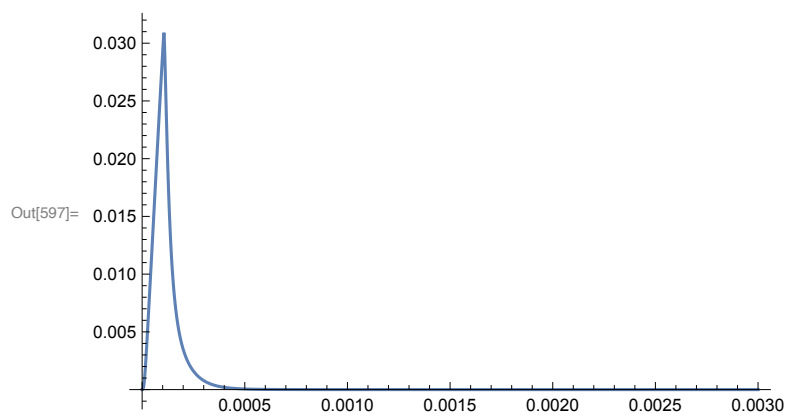
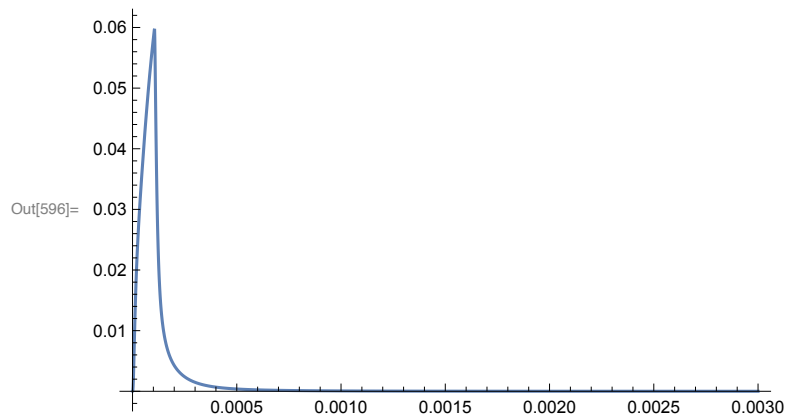
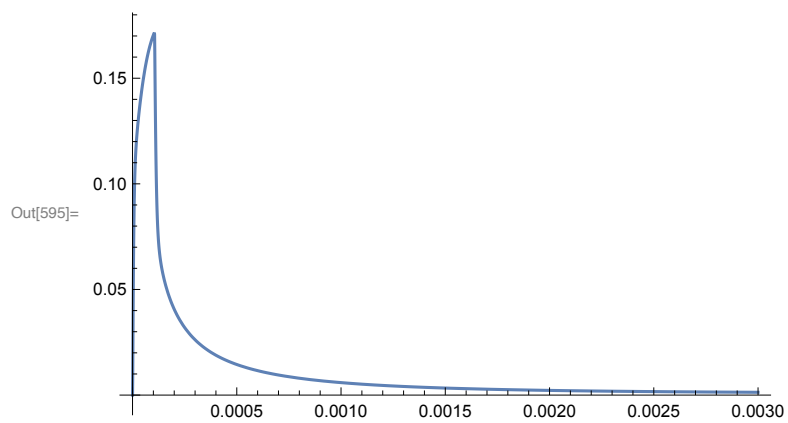
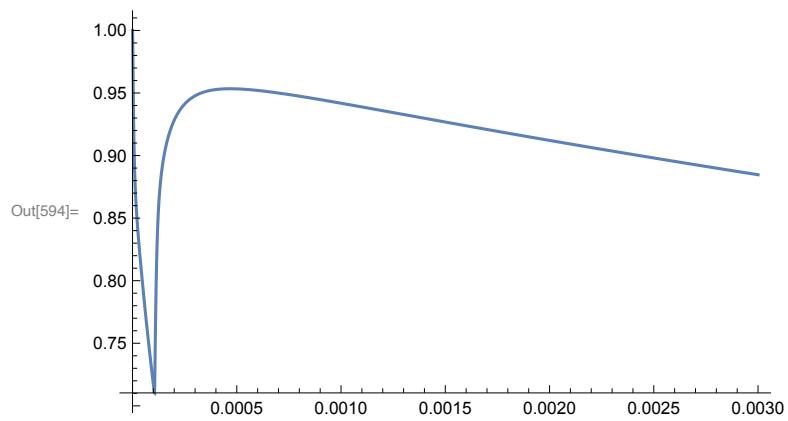
In[588]:= Clear[caFunc, eq];(*Clear is needed if the cell is executed for a 2nd time
when caFunc is already set to a value or an Interpolationfunction*)
caFunc[t_] := interpolFunc[t];
ssA[t_] = {ss1A[t], ss2A[t], ss3A[t], ss4A[t], ss5A[t], ss6A[t], ss7A[t]};
ssB[t_] = {ss1B[t], ss2B[t], ss3B[t], ss4B[t], ss5B[t], ss6B[t], ss7B[t]};
eq = {
  ssA'[t] == (matA /. repl).ssA[t],
  ssA[0] == {ss0AInitial, 0, 0, 0, 0, 0, 0},

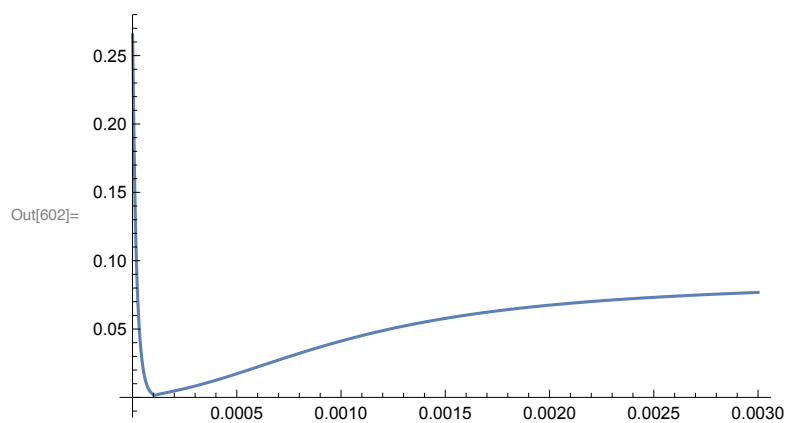
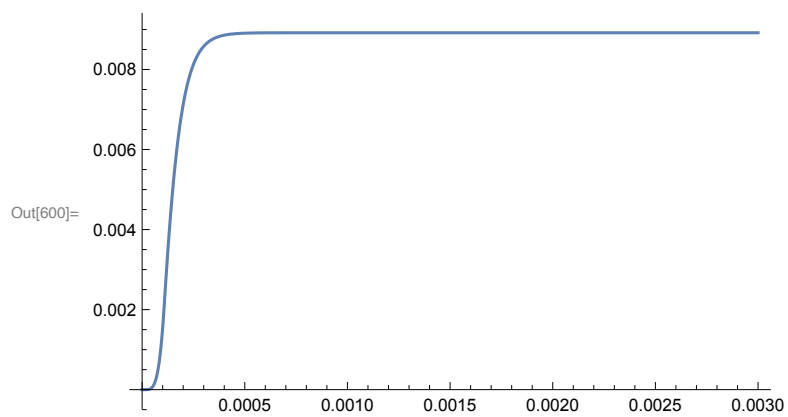
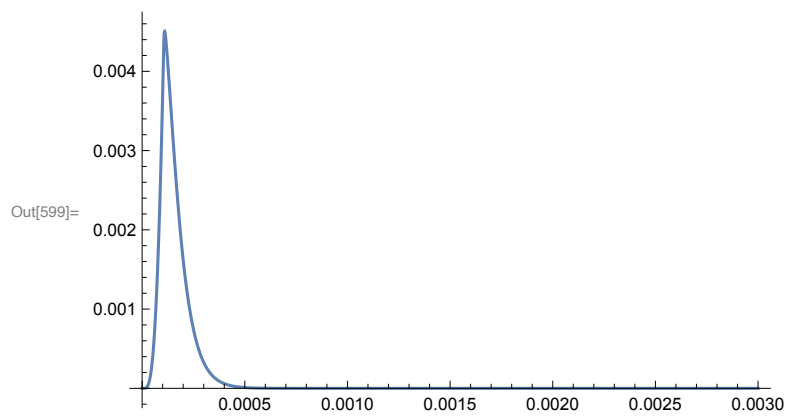
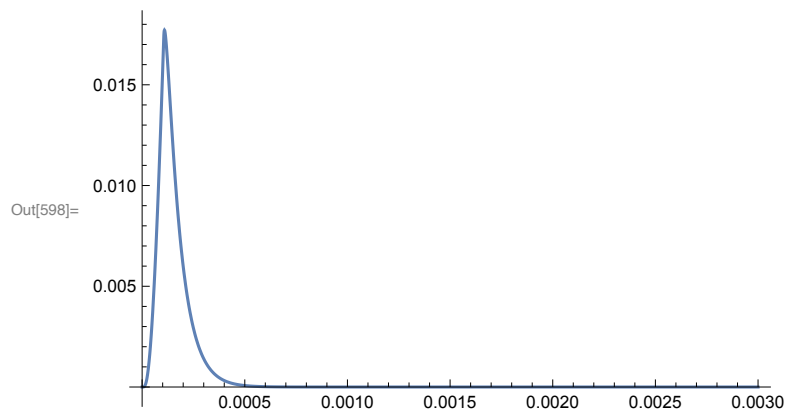
  ssB'[t] == (matB /. repl).ssB[t],
  ssB[0] == {ss0BInitial, 0, 0, 0, 0, 0, 0}
};

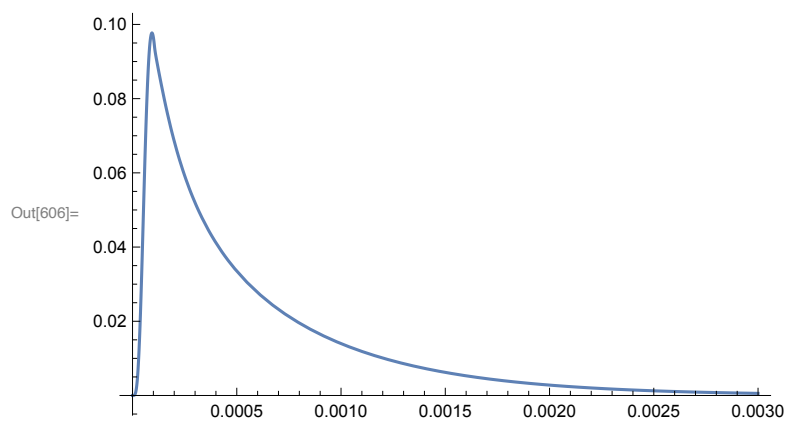
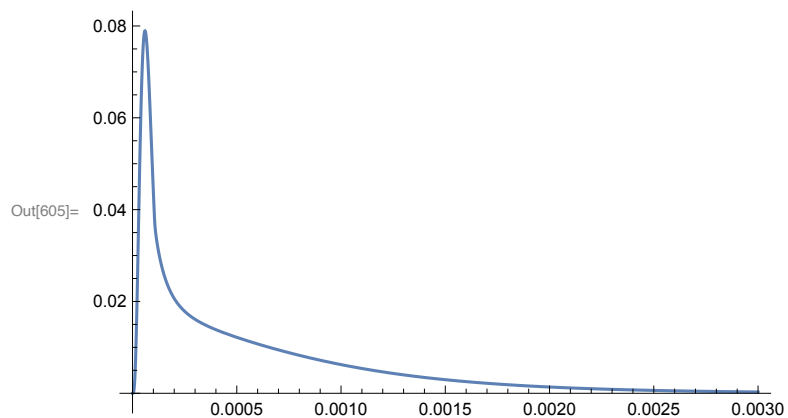
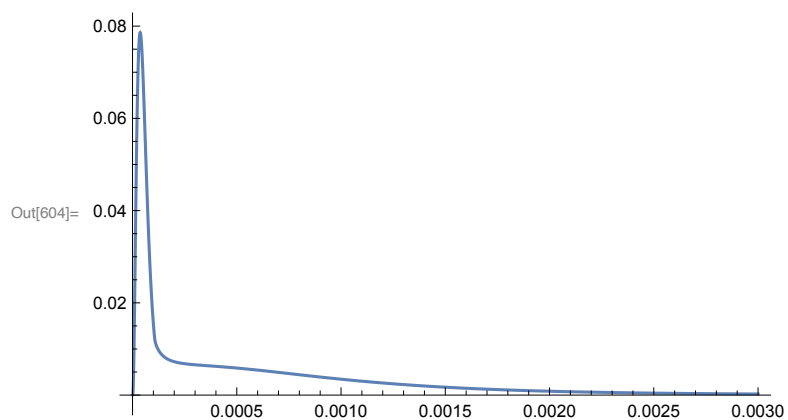
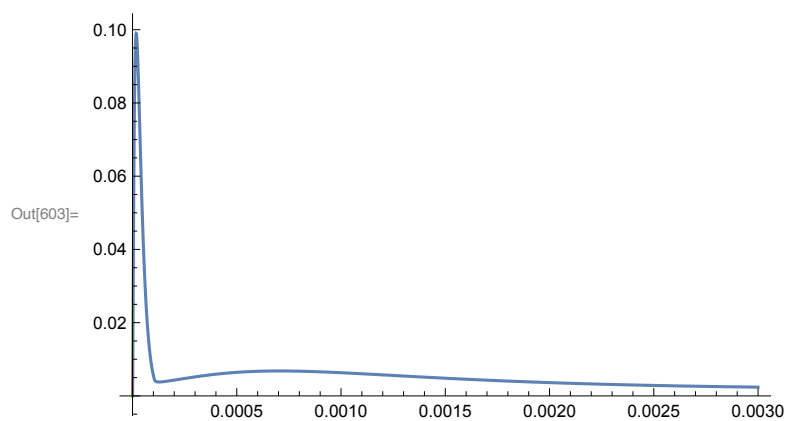
```

## NDSolve

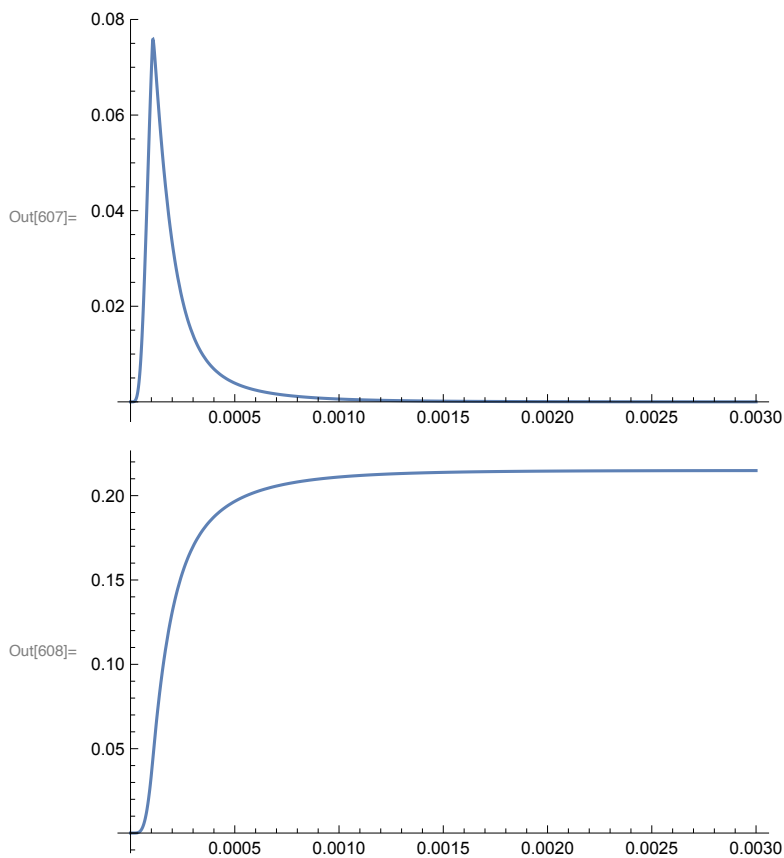
```
In[593]:= myNDSolveResults = NDSolve[eq, {ss1A, ss2A, ss3A, ss4A, ss5A, ss6A,
      ss7A, ss1B, ss2B, ss3B, ss4B, ss5B, ss6B, ss7B}, {t, 0, 0.003}];
Plot[(ss1A[t] /. myNDSolveResults), {t, timeStartForPlot, timeEndForPlot},
  PlotRange → All]
Plot[(ss2A[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss3A[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss4A[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss5A[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss6A[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss7A[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Print["-----"];
Plot[(ss1B[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss2B[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss3B[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss4B[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss5B[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss6B[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss7B[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
```





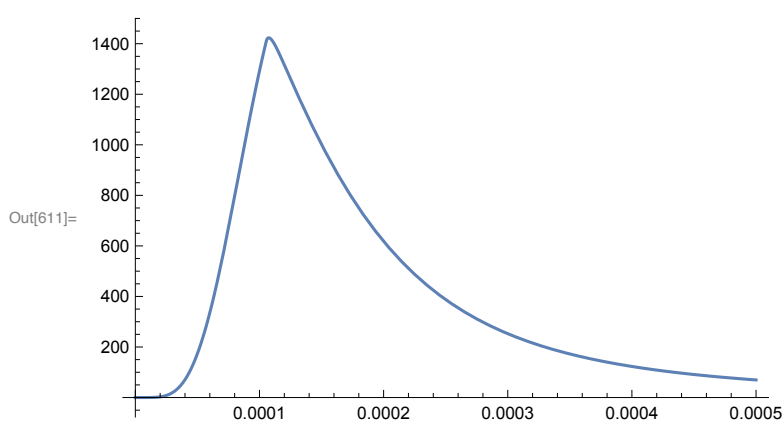






## Plot EPSC

```
In[609]:= epscLowCa = D[(ss7A[t] + ss7B[t]) /. myNDSolveResults], t];
Plot[epscLowCa, {t, 0, 2.*^-3}, PlotRange -> All];
Plot[epscLowCa, {t, 0, 0.5*^-3}, PlotRange -> All]
```



# High Ca

## Initial occupancy

```
In[612]:= (*calualte initial equilibrium occupancy*)
caFunc[t_] := caRestHigh;
ss0AInitial = kprimSchemeA / kunprimSchemeA
ss0BInitial = (kprimSchemeA / kunprimSchemeA) * (kprimSchemeB / kunprimSchemeB)
```

Out[613]= 1.

Out[614]= 0.836742

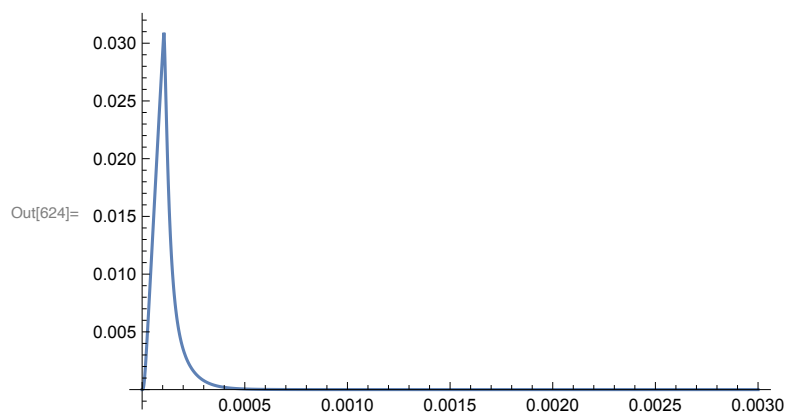
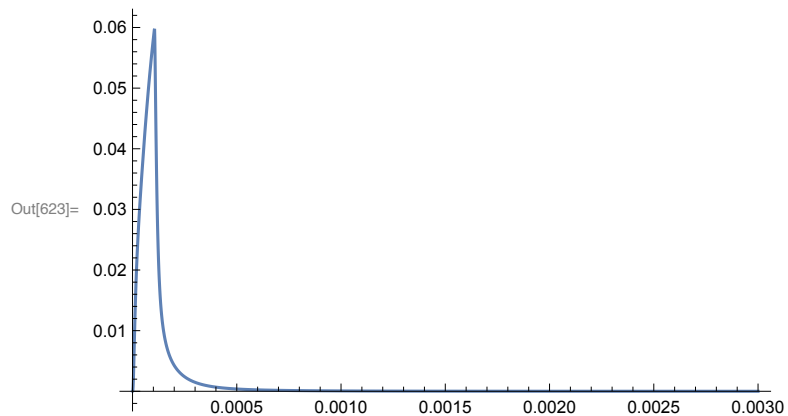
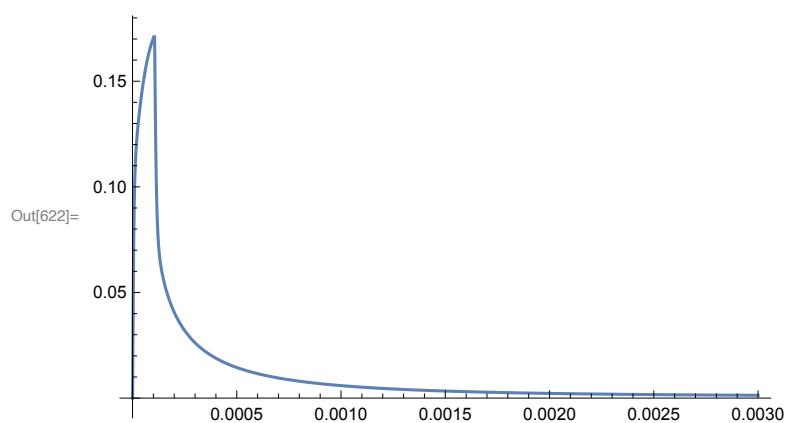
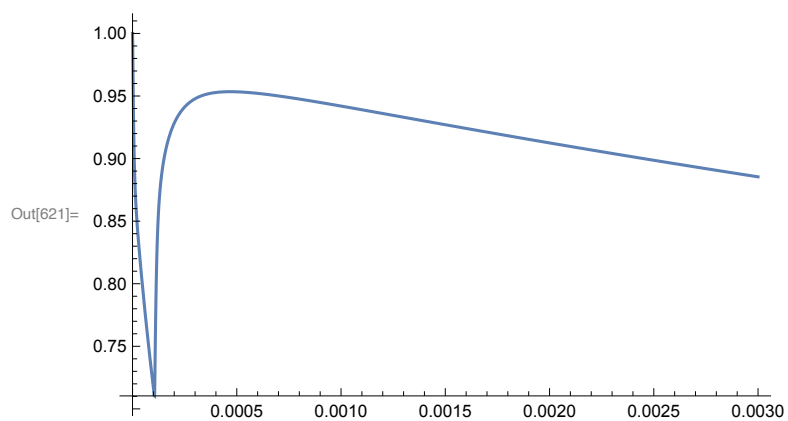
## Diff eq.

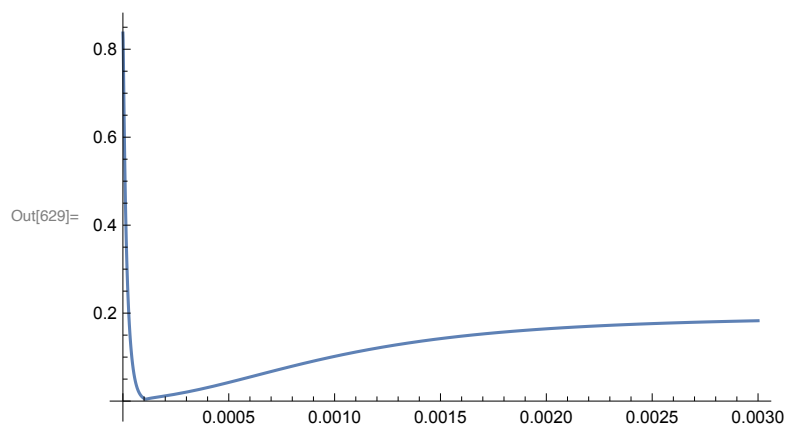
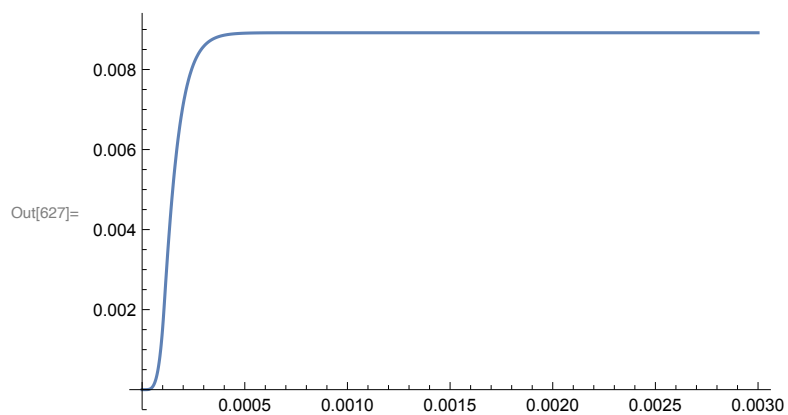
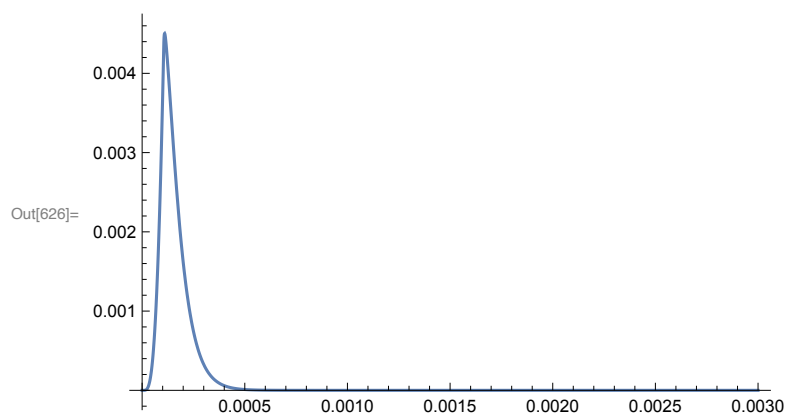
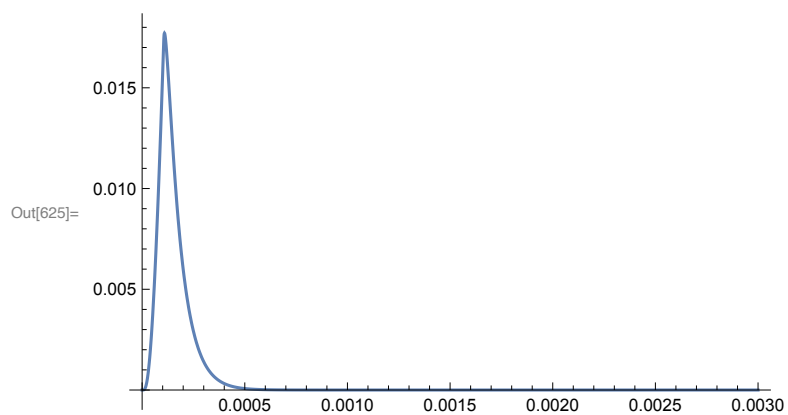
```
In[615]:= Clear[caFunc, eq];(*Clear is needed if the cell is exected for a 2nd time
when caFunc is already set to a value or an Interpolationfunction*)
caFunc[t_] := interpolFunc[t];
ssA[t_] = {ss1A[t], ss2A[t], ss3A[t], ss4A[t], ss5A[t], ss6A[t], ss7A[t]};
ssB[t_] = {ss1B[t], ss2B[t], ss3B[t], ss4B[t], ss5B[t], ss6B[t], ss7B[t]};
eq = {
  ssA'[t] == (matA /. repl).ssA[t],
  ssA[0] == {ss0AInitial, 0, 0, 0, 0, 0, 0},

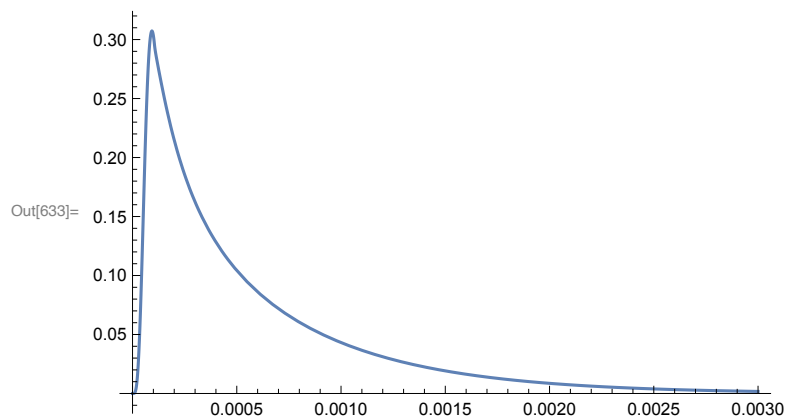
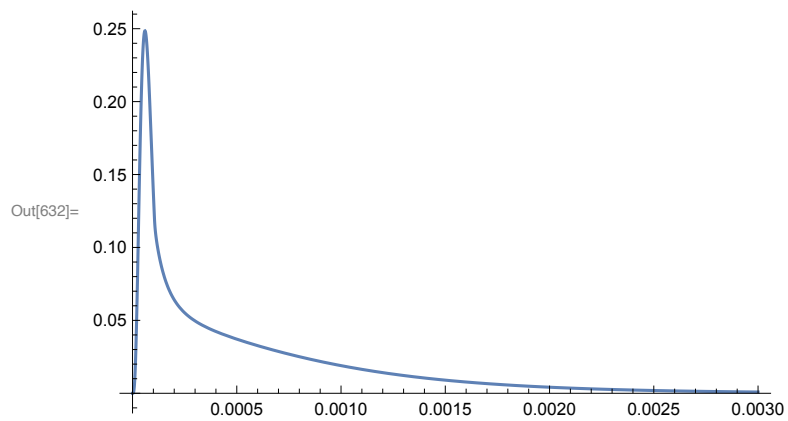
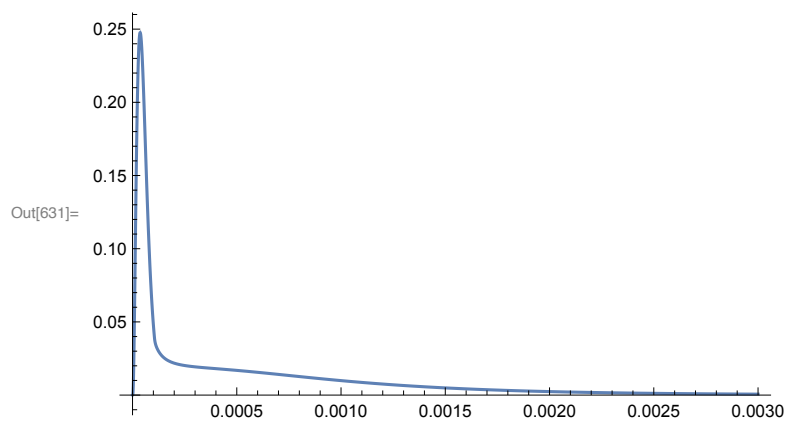
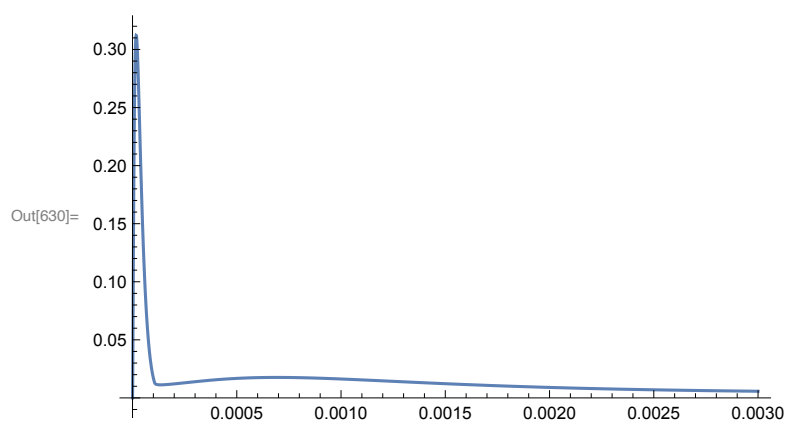
  ssB'[t] == (matB /. repl).ssB[t],
  ssB[0] == {ss0BInitial, 0, 0, 0, 0, 0, 0}
};
```

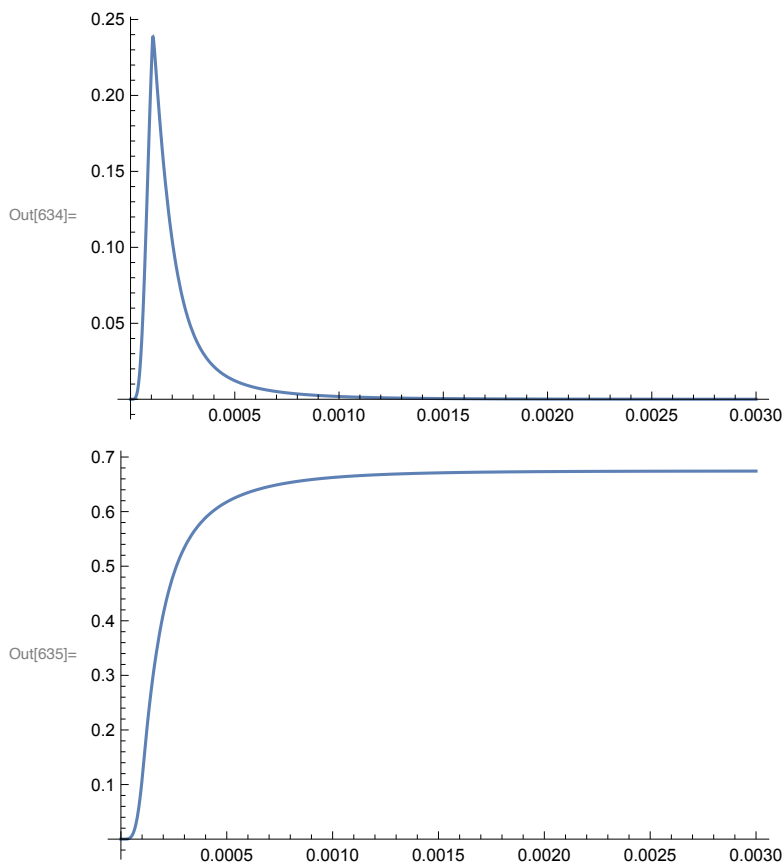
## NDSolve

```
In[620]:= myNDSolveResults = NDSolve[eq, {ss1A, ss2A, ss3A, ss4A, ss5A, ss6A,
      ss7A, ss1B, ss2B, ss3B, ss4B, ss5B, ss6B, ss7B}, {t, 0, 0.003}];
Plot[(ss1A[t] /. myNDSolveResults), {t, timeStartForPlot, timeEndForPlot},
  PlotRange → All]
Plot[(ss2A[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss3A[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss4A[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss5A[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss6A[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss7A[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Print["-----"];
Plot[(ss1B[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss2B[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss3B[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss4B[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss5B[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss6B[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
Plot[(ss7B[t] /. myNDSolveResults),
  {t, timeStartForPlot, timeEndForPlot}, PlotRange → All]
```



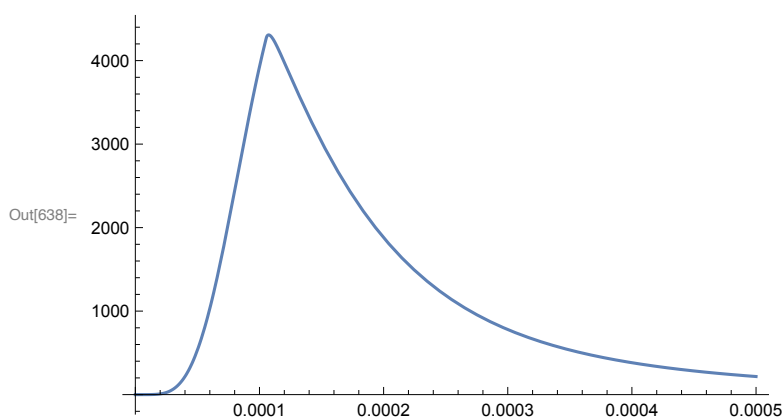






## Plot EPSC

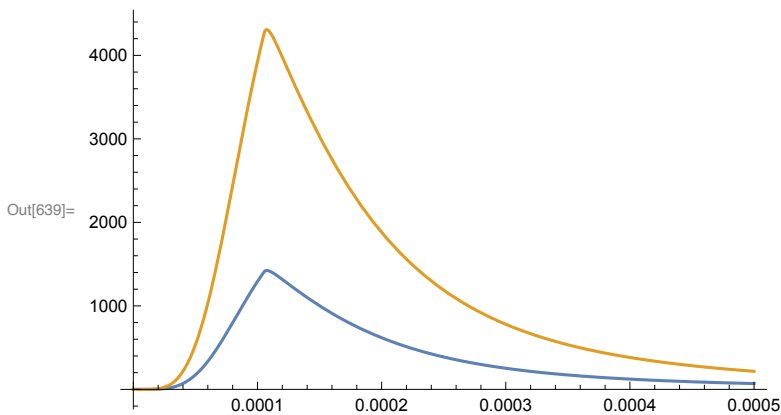
```
In[636]:= epscHighCa = D[(ss7A[t] + ss7B[t]) /. myNDSolveResults], t];
Plot[epscHighCa, {t, 0, 2*^-3}, PlotRange -> All];
Plot[epscHighCa, {t, 0, 0.5*^-3}, PlotRange -> All]
```



## Compare

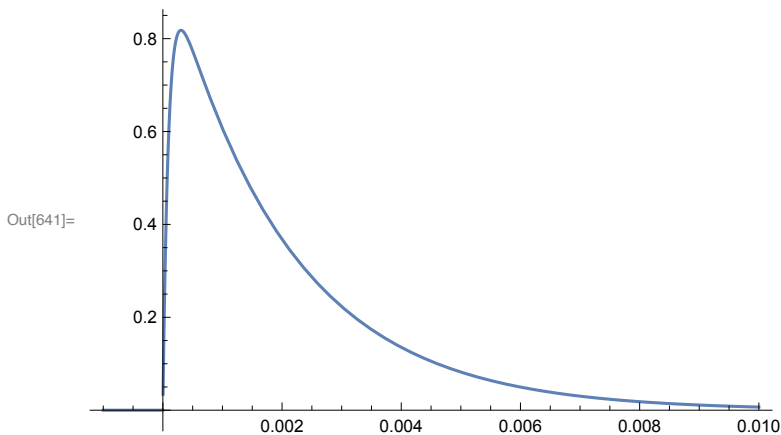
### Plot both release rates

```
In[639]:= Plot[{epscLowCa, epscHighCa}, {t, 0, 0.5*^-3}, PlotRange -> All]
```

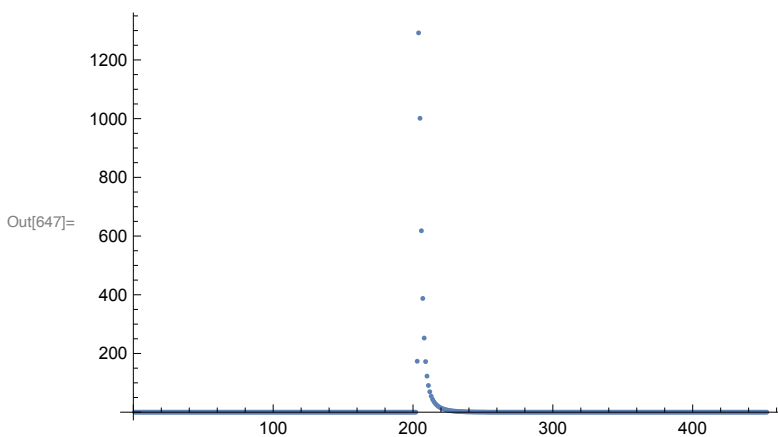
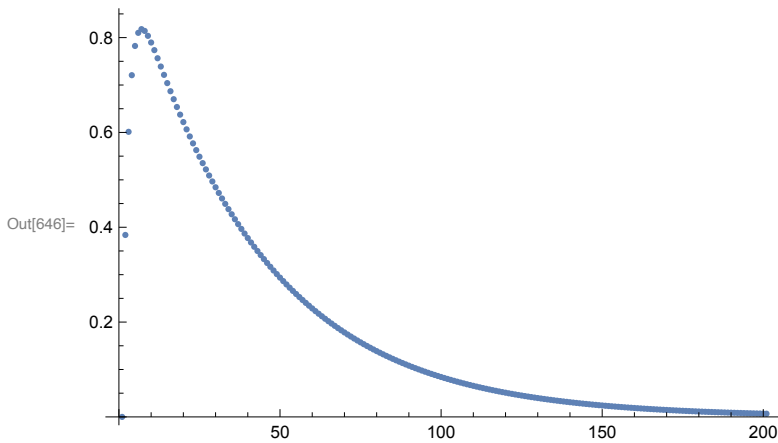


## Convolution RelRate => EPSC

```
In[640]:= 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]
```



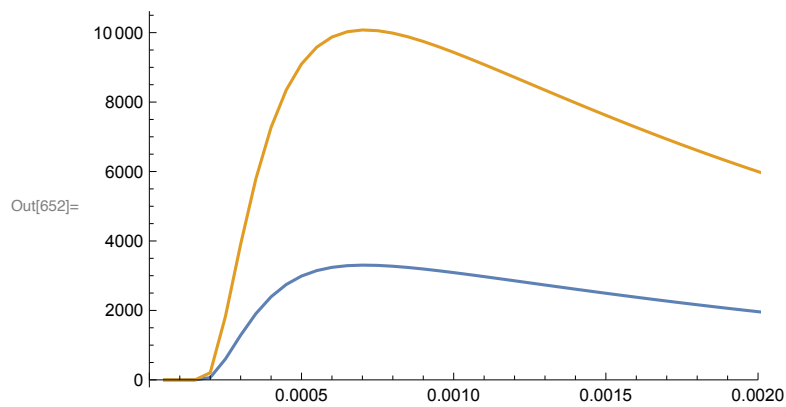
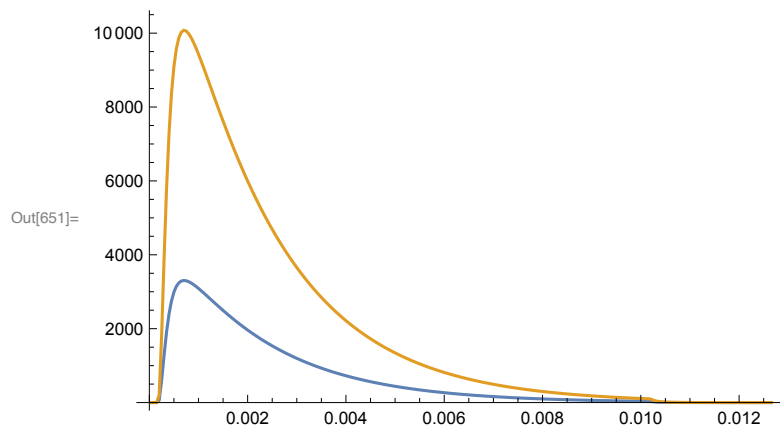




```

In[648]:= 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"];
];

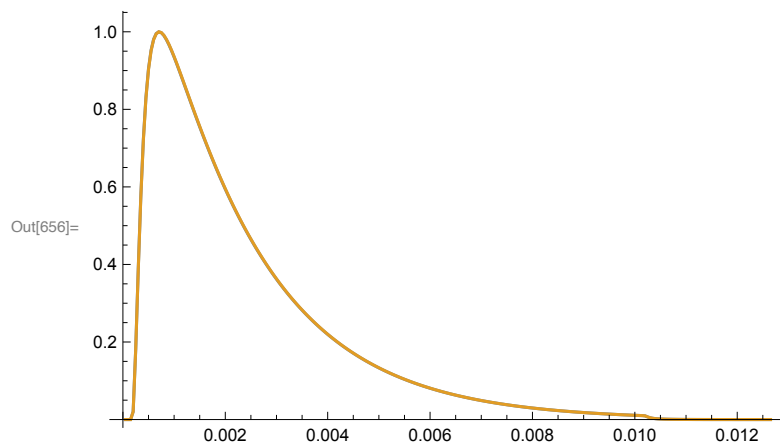
```

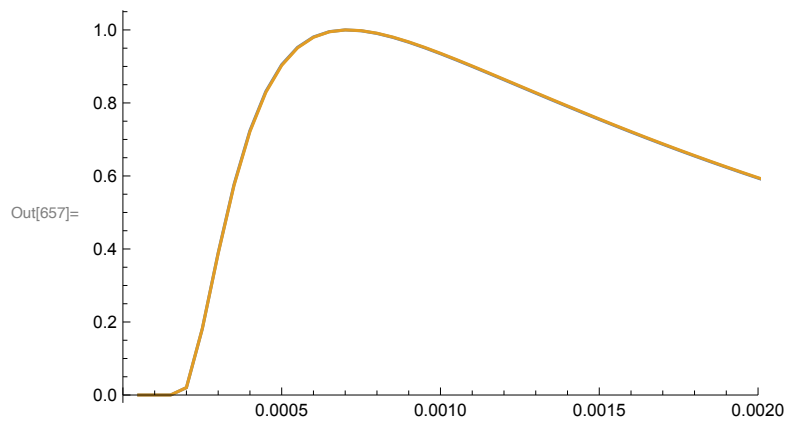


Out[653]= 3304.39

Out[654]= 10 076.8

maxHigh/maxLow = 3.0495





# Timing

```
In[659]:= timeEnd = AbsoluteTime[]
           (timeEnd - timeStart) / 60. (*time of calculation in min*)
```

Out[659]=  $3.841510563799333 \times 10^9$

Out[660]= 0.313377