Comparing Sky Camera Solar Forecasting Systems for PV/Diesel Systems

Colin Bonner, Antonin Braun, Thomas Loveard, Phil Maker and Marc Mueller-Stoffels. .

Abstract—**Always write abstract at the end, or is it the beginning. Regardless this is a trial.**

Index Terms—**LaTeX, Haskell, Hybrid Power System. Solar nowcasting, Sky Camera, PV/Diesel.**

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Colin Bonner is with Fulcrum 3D
Antonin Braun is with Reuniwatt
Thomas Loveard is with Pixel Sciences
Phil Maker is with Powerwater working on the SETuP project and also helps the Alaskan Center for Energy and Power occasionally. He has worked on a variety of embedded systems (inside peopel) and hybrid power systems including the RIWE and Coral Bay
Marc Mueller-Stoffels is with Alaskan Center for Energy and Power
Manuscript received April 19, 20015; revised December 27, 2012 via time travel.

I. Notes to Authors

This section is intended to summarise our cunning plan as authors, technology providers, grant agencies and utilities. It will not appear in the final report.

Please do not take the first draft too seriously.

A. How are we going to do it?

This paper/comparision is based on previous work comparing different control systems, in particular standard PV/Diesel control vs perfect prediction.

The method is to use Literate Haskell as a tool to buildup a framework for the comparision. We could of course use other methods (MATLAB) but its kind of nice to have something that is:

- Free/open source including the model, its engine and tools.
- Complete, at least in the full version of this paper including all the annexes.
- Short since its in a functional programming language (see Haskell [1]).
- At least reasonably well tested using property based testing, see QuickCheck [2].

B. Administration

1) The paper contents will be kept on github in a possibly private archive with access by all authors. All nominated authors have veto over the contents, if not the results. In addition we have a few other people providing input/review,.
2) They are formatted in LaTeX using an IEEE style for now.
3) In terms of editing using a text editor on `main.lhs` should get you a long way. If `git` is a bit painful send me the text or changes and I will merge them. Give me a ring for details.
4) More to follow.

C. Proposed Roles and Responsibilities

In terms of how this will work:

1) My guess is I'll do most of the editing/coding with approval from everyone else.
2) Perhaps a few sections can be shared amongst the other authors, my initial thoughts are:
   a) Colin/Antonin/Thomas: a review of the testing methodology itself.
   b) Marc: I'd like to bring in once we have a draft for a fresh pair of eyes (both physics/CS and academic). Marc has a few ideas and has already talked to Thomas.
   c) Phil: I'm just here to help but expect to do the coding and definitions for the comparision methodology with your kind assistance.

D. General Questions to the Authors

And finally some questions to the gentle authors:

1) What are the critical assumptions?
2) What do we want in the results?
3) How should we compare solutions?
4) When should the first phase of the trial finish?

II. Introduction

The use of sky camera solar forecasting within PV/Diesel systems promises significant improvements in system performance. The SETuP [1] project has funded a trial of this technology in the Northern Territory.

Usual fluff and buff.

Explain what goes where.

A. Characteristics of PV Systems

PV output is rapidly effected by cloud events, a Powerwater rule of thumb for a 300kW sized array is that a cloud event will reduce output from 100% to 20% within 6..10s occur and must be covered by station spinning reserve.

The justification for the 6s is that:

1) Solar output varies with wind speed, at altitude the speed might be 10m/s.
2) A centralised solar array of 300kW size might be 60m across.

Since typical diesel generater start and synchronise times are in the 20..90s range we then need to carry spare capacity (Spinning Reserve) capable of dealing with cloud events or use energy storage or demand management.

Do we want to deal with Schrodingers Law and stability here as well. (uhm perhaps not).

A typical weeks worth of `PvAvailP` data shows significant solar variation over the time scales discussed.It is also worth noting that 4 out of the 7 days are very stable ad whence suitable for running a smaller diesel.

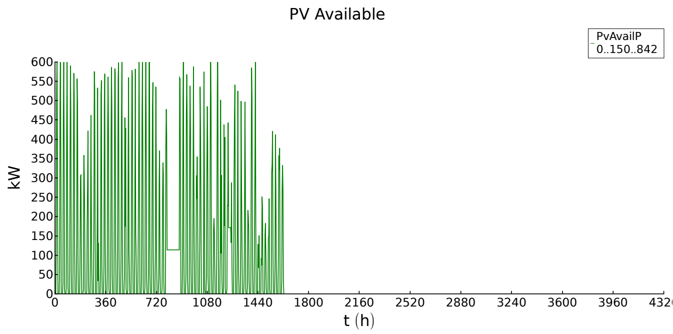

Fig. 1. PV Available

Given a diesel start, synchronisation and ramp up time for a diesel of 60s it is considered that a 120s (2 minute) prediction that PV is going to drop would be most useful. Figure 2 shows the maximum downwards variation in PV output over a 120s window.

```
sPvAvailMaxDown120P = maxdown 120 sPvAvailP
sPvAvailMaxDown120PAboveSpin = map aboveSpin sPvAvailMaxDown120P
  where aboveSpin x = if x < 50 then 0 else x
```

[1]SETuP is described as ...

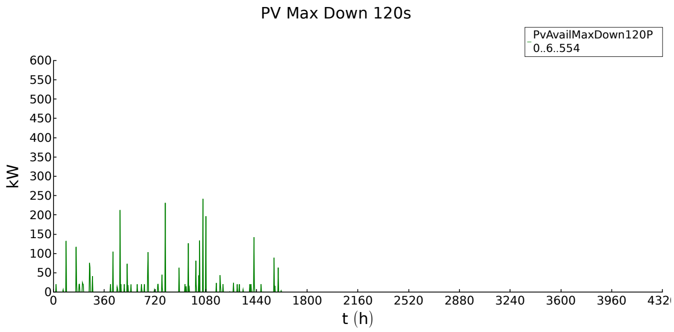

Fig. 2. PV Available Max Down over 120s

Figure 3 show those drops which are above the system spinning reserve of 50kW (`SpinPPa`).

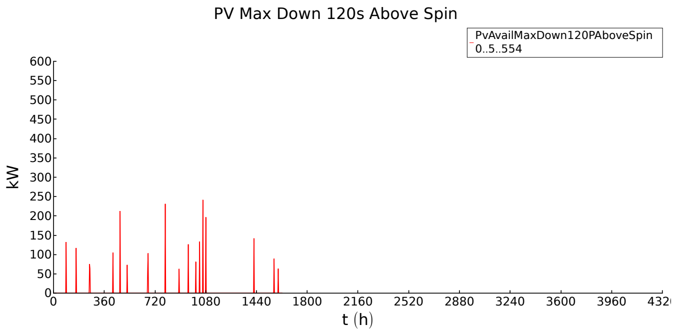

Fig. 3. PV Available Max Down over 120s above Spinning Reserve

B. Characteristics of Diesel Systems

Describe how they work, synchronisation, etc.

C. Characteristics of Diesel Generator Fuel Consumption

In this section we will describe the diesel fuel consumption model that will be used in this paper. This particular model whilst well known (see ...) is perhaps underused by some practitioners since diesel generator fuel consumption is usually presented as a non linear function as in fig 4. The summary of the model and its uses are:

1) Fuel consumption variation is nearly linear and varies with both load and capacity online.
2) Fuel consumption can be modelled as two independent components:
   a) `capCost` - the cost of maintaining capacity online regardless of load. This cost is independent of the current load and can be thought of as the standby losses for running the diesels at a fixed speed in a synchronous generater (e.g. 1500rpm).
   b) `energyCost` - the cost of energy production in L/kWh which is approximately the same across diesels sizes in the range 100kW to 1MW. There is an improvement as engine size scales but it is not considered significant across the range discussed.
3) This allows us to place bounds on possible performance improvements without doing any detailed modelling. For example:
   a) In a diesel only system any change to generator configuration can only effect the `capCost` component, not the `energyCost`.
   b) So for a system with N $\times$ 320kW generators a control system improvement that turns off a single generator 30 minutes earlier each day can only result in a saving of `capCost 320` which is around 12l/h¿ so that total saving is $12 \times 0.5 \times 365 = 2190$ L/year.
4) We can also provide estimates of the cost of keeping spinning reserve online in a simple manner based on `capCost` and typical system loadings.

Traditionally fuel consumption is presented using a kWh efficiency curve such as fig 4. This is clearly non-linear and shows that at low loads fuel consumption per kWh is typically twice that at full load.

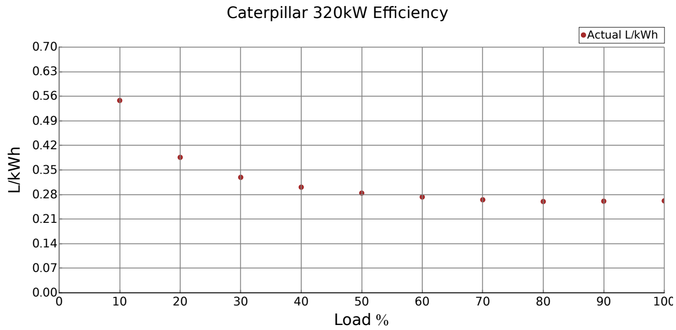

Fig. 4. Fuel Consumption in L/kWh vs Load is non linear

However the same data presented with L/h rather than L/kWh results in fig 5 which shows clearly that fuel consumption is close to linear for a given engine and that the losses at 0% load result in most of the efficiency decrease at low load in fig 4.

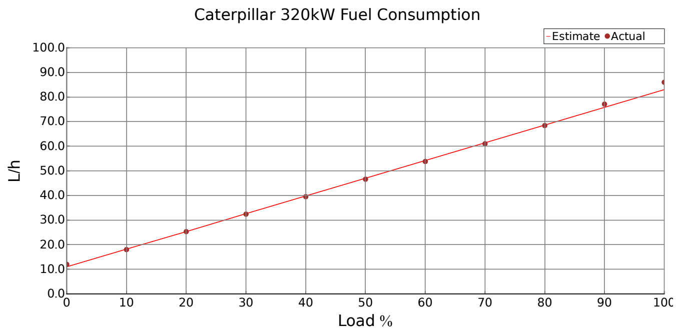

Fig. 5. Fuel Consumption vs Load is very close linear

From fig 5 it is apparent that the fuel consumption rate can be modelled using two constants `capCost` and `energyCost`:

1) `capCost`: is the fuel in L/h required at 0% load in order to run the engine at its synchronous speed. .
2) `energyCost`: the cost in L/kWh for producing a single kWh. Modern engines are fairly close in efficency terms at the same load factor.

The `capCost` for a generator is typically around 12% of its fuel consumption at full load. This numbers scales with generator size as is to be expected though a further study to develop detailed scaling laws for these systems would be most worthwhile.

It should also be noted that in most systems generaters do not spend a significant time at 100% load because of spinning reserve requirements so the non linearity at high loads is reduced.

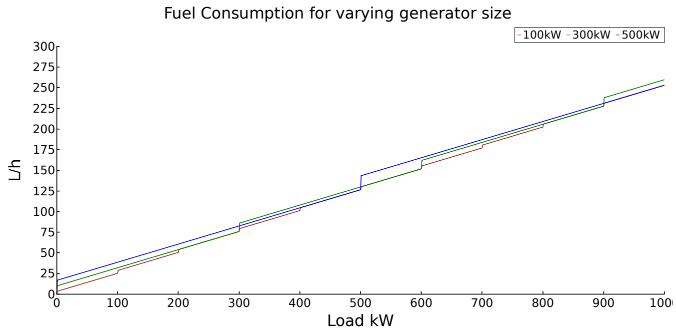

Fig. 6. Fuel Consumption vs Load and Generator Size

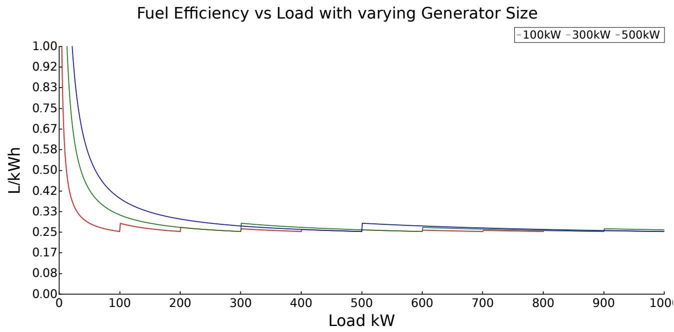

Fig. 7. Energy Cost vs Load and Generator Size

1) *Total fuel used calculation:* The total fuel used will be calculated by summing from the online `capacitys` and `loads`. The particular constants are based on the 328kW Caterpillar Generator.

```
-- Buglet if loads > capacitys no ngen generation ttt
fuelUsedTotal :: [Double] -> [Double] -> Double
fuelUsedTotal capacitys loads = (capCost * sum capacitys) +
                                (energyCost * sum loads)

fuelUsedAt :: Double -> Double -> Double
fuelUsedAt capacity load = (capCost * capacity) + (energyCost * load)

sfuelUsed :: [Double] -> [Double] -> [Double]
sfuelUsed [] [] = []
sfuelUsed (c:cs) (l:ls) = (fuelUsedAt c l):sfuelUsed cs ls
sfuelUsed cs ls = error $ "fatal: sfuelUsed has leftover values " ++
                  "cs " ++ show cs ++ " ls " ++ show ls


capCost = (fuelUsed0Pct/fuelUsedRating)/3600.0
energyCost = ((fuelUsed100Pct-fuelUsed0Pct)/fuelUsedRating)/3600.0

fuelUsedRating = 328.0 -- kW rating for diesel
fuelUsed0Pct = 11.0    -- L/h at 0% load
fuelUsed100Pct = 83.0  -- L/h at 100% load
```

2) *Fuel Usages Examples:* The following examples may be helpful

```
example 3600.0 * fuelUsed [0.0] [0.0] ->
        0.0 OK
example 3600.0 * fuelUsed [328.0] [0.0] ->
        11.0 OK
example 3600.0 * fuelUsed [328.0] [328.0] ->
        83.0 OK
```

3) *Diesel Generator Fuel Consumption Source Data:* The source data is in presented in the following table along with the values for a simple Estimate based on the described method.

Real kWe	Real %Load	Real L/h	Real L/kWh	Estimated L/h
328	100	86	0.262	83
295.2	90	77.1	0.261	75.8
262.4	80	68.4	0.260	68.6
229.6	70	61	0.265	61.4
196.8	60	53.8	0.273	54.2
164	50	46.6	0.284	47
131.2	40	39.5	0.301	39.8
98.4	30	32.4	0.329	32.6
65.5	20	25.3	0.386	25.3
32.8	10	18	0.548	18.2
0.0	0	12	undefined	11

TABLE I
CATERPILLAR 328KW GENERATOR DATA SHEET

4) *Fuel usage as generator size and kW load increases:* Using the proposed model for fuel consumption we end up with the following approximations for fuel consumption and energy cost vs Load and Generator size assuming a fleet of units of the same size.

The same data presented in L/kWh results in fig 7.

The final plots shows the cost in L/y versus Generator Capacity and load.

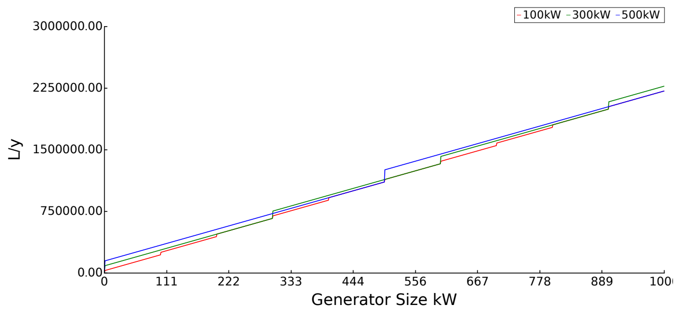

Fig. 8. Total Cost per year vs Load and Generator Size

D. *On Stability and distributed PV*

Do we need to talk about stability [3],
I think so..

E. *Benefits*

With 3 x 320kW Detroit Series 60 Diesels:

During 5 out of the 7 days we could run 1 rather than 2 for a typical load.If the minimum loading is ... then ...

```
-- A
```

III. METHOD

How are we going to do it.

A. *Trial Outline Plan*

B. *Trial Location and Timing*

C. *Completion of the Phase 1 Trial*

The Phase 1 trail will be completed when either:

- 6 months has elapsed.
- At least 100 significant events which demonstrate:
  - Drop in `PvAvailP` by at least 50% over 2 minutes.
  - The Drop occurs from a cloud free value of at least 60% of rated.
  - Events are separated by at least 30 minutes.

Preliminary results will be released at 3 months.

```
sKalkPvAvailP = take 10000000 $ readls "../data/KalkPvAvailP.ls"

events xs = if (length xs) > 2*6000
  then if d > 1 then d:events (drop 12000 xs)
       else events (drop 12000 xs)
  else []
  where d = before - after
        before = (minimum $ take 6000 xs)
        after = (minimum $ take 6000 $ take 6000 xs)
```

```
figKalkPvAvailP = fig "figKalkPvAvailP.pdf"
  [figLine (sKalkPvAvailP, "KalkPvAvailP", red)]
  figCfg
```

```
= (ramp 1 0.1 sPvAvailP)
```

D. Method 1

A

E. Method 2

A

IV. Results

What are the results.
In a series of tables etc.

A. Results A

B. Results B

V. Conclusion

So what.

VI. Acknowledgments

Thanks to my mother.

References

[1] Haskell programming language homepage. [Online]. Available: http://www.haskell.org

[2] K. Claessen and J. Hughes, "QuickCheck: a lightweight tool for random testing of Haskell programs," *ACM SIGPLAN Notices*, vol. 46, no. 4, pp. 53–64, Apr. 2011.

[3] E. Schrödinger, *What is life?* Cambridge: Cambridge University Press, 1944.

Colin Bonner is a director with Fulcrum 3D which ....

Antonin Braun is with Reuniwatt solving the worlds problems.

Thomas Loveard is very good looking

Phil Maker Phil Maker is working at Powerwater on the SETuP project as a controls engineer. His other role is Research Adjunct Professor at the Alaskan Center for Energy and Power (ACEP, acep.uaf.edu). Previous work includes the development of a distributed control system for high penetration power systems and work on an early embedded (inside people) system (4210 Defibrillator).

Dr. Marc Mueller-Stoffels is the Director for the Power Systems Integration Program at the Alaska Center for Energy and Power (ACEP, acep.uaf.edu) and Research Assistant Professor at the University of Alaska Fairbanks' Institute of Northern Engineering. Marc's research focuses on the integration of variable generation sources into isolated microgrids. Most recently, he has lead the testing of an inverter-battery system to enable diesel-off mode, and a flywheel energy storage system for power quality applications in high contribution renewable energy scenarios. Prior to joining ACEP, Marc has developed regional scale climate models with focus on Arctic sea ice, and has chaired a small software company specializing in optimization algorithms and scenario management. Marc holds graduate degrees in physics from the University of Alaska Fairbanks and Otago University, New Zealand.

APPENDIX A
SOMETHING INTERESTING

I'm not sure but perhaps an overview.

APPENDIX B
IMPLEMENTATION DETAILS

A. Characteristics of PV Generation

B. Predictor using the past for the future

```
pastPredict s t = 10
```

C. minRunTime

MinRunTime is intended

```
-- minRunTime period goes up instantly but remains at its level
--   until the timeout t expires
minRunTime t [] = []
minRunTime t (x:xs) = x:minRunTime_ t 1 x xs

minRunTime_ :: Int -> Int -> Double -> [Double] -> [Double]

minRunTime_ t c v [] = []
minRunTime_ t c v (x:xs) =
 if c < t && x <= v -- no timeout, no step up so stay at v
 then v:minRunTime_ t (c+1) v xs
 else if c < t && x > v -- no timeout, increased so go up to x
 then x:minRunTime_ t (c+1) x xs
 else if c >= t && x <= v -- timedout, no step so goto v
 then x:minRunTime_ t 1 x xs
 else {- c >= t && x > v -} -- timedout, increased so goto x
      x:minRunTime_ t 1 x xs


prop_minRunTime_ident_0 :: [Double] -> Bool
prop_minRunTime_ident_0 xs = xs == minRunTime 0 xs

prop_minRunTime_ident_1 :: [Double] -> Bool
prop_minRunTime_ident_1 xs = xs == minRunTime 1 xs

examples_minRunTime = do
 examples "examples_minRunTime.txt" display exprn isok tests
 where display ((t,xs),ok) = "minRunTime " ++ show t ++ " " ++ show xs
       exprn ((t,xs),ok) = minRunTime t xs
       isok ((t,xs),ok) = exprn((t,xs),ok) == ok
       tests = [((0,[1..10]),[1..10]),
                ((0,[5,4..1]),[5,4..1]),
                ((1,[5,4..1]),[5,4..1]),
                ((5,[5,4..1]),[5,5,5,5,5]),
                ((2,[5,4..0]),[5,5,3,3,1,1]),
                ((2,[1..10]),[1..10]),
                ((3,[10,9..1]),[10,10,10,7,7,7,4,4,4,1])
               ]
```

hysterisisDown

D. HysterisisDown

```
-- a signal must decrease by at least k before we accept it
hysterisisDown k [] = []
hysterisisDown k (x:xs) = x:hysterisisDown_ k x xs

hysterisisDown_ k v [] = []
hysterisisDown_ k v (x:xs) =
 if x >= v
 then x:hysterisisDown_ k x xs
 else if x <= v-k
 then x:hysterisisDown_ k x xs
 else v:hysterisisDown_ k v xs


examples_hysterisisDown = do
 examples "examples_hysterisisDown.txt" display exprn isok tests
 where display ((t,xs),ok) = "hysterisisDown " ++ show t ++ " " ++ show xs
       exprn ((t,xs),ok) = hysterisisDown t xs
       isok ((t,xs),ok) = exprn((t,xs),ok) == ok
       tests = [((0,[1..10]),[1..10]),
                ((0,[0,-24.0]),[0,-24.0]),
                ((0,[0,6.0]),[0,6.0]),
                ((10,[1..100]),[1..100]),
                ((10,[100,90..10]),[100,90..10])
               ]

prop_hysterisisDown_0 :: [Double] -> Bool
prop_hysterisisDown_0 xs = xs == hysterisisDown 0 xs
```

capacity

E. Capacity

```
capacity :: [Double] -> [Double] -> [Double]
capacity caps xs = map (capacity_ caps maxcap) xs
 where maxcap = maximum caps

capacity_ :: [Double] -> Double -> Double -> Double
capacity_ [] maxcap x = maxcap -- exceed maximum capacity
capacity_ (cap:caps) maxcap x =
 if x <= cap then cap
 else capacity_ caps maxcap x

capacityDefault = capacity [300,600..1000]
```

F. Tools

```
noise :: (Random a) => Int -> [a]
noise seed = randoms (mkStdGen seed)

step01 :: Int -> Int -> [Double]
step01 p q = cycle l
 where l = (replicate p 0.0) ++ (replicate q 1.0)

repeats xs p = cycle $ repeats_ xs p
repeats_ (x:xs) p = (replicate p x) ++ (repeats_ xs p)
repeats_ [] p = []

walk seed pu =
 intergrate sig
 where sig = scale pu $ offset (-0.5) $ noise seed

       offset k s = map (k+) s
scale k s  = map (k*) s
offset k s  = map (k+) s

diff :: [Double] -> [Double] -> [Double]
diff [] [] = []
diff (x:xs) (y:ys) = (x-y):diff xs ys
diff xs [] = xs
diff [] ys = ys

combine :: [Double] -> [Double] -> [Double]
combine [] [] = []
combine xs [] = xs
combine [] ys = ys
combine (x:xs) (y:ys) = (x+y: combine xs ys)

limitabove_s k s = map (max k) s
limitbelow_s k s = map (min k) s
limits low high s = map (limit low high) s
limit low high v = max low $ min high v

intergrate s = intergrate' 0.0 s
intergrate' p (x:xs) = (p+x):(intergrate' (p+x) xs)
intergrate' s [] = []

intergrate_within v low high = intergrate_within_ 0.0 v low high
intergrate_within_ s (x:xs) low high = sum:(intergrate_within_ sum xs low high)
 where sum = limit low high (x+s)
intergrate_within_ s [] low high = []
```

G. `smooth` – smoothing data

A common process in all control is the smoothing of data in order to eliminate:

1) Instrument noise.
2) Synchronisation noise where a signal is ...

The `smooth` function implements a simple single pole recursive low pass filter which behaves in the same way as a simple RC circuit. See DSP guide *http://www.dspguide.com/ch18/2.htm* for an introduction to these filters.

```
smooth :: Double -> [Double] -> [Double]
smooth k xs = smooth_ xs (head xs) ((tau k))
smooth_ (x:xs) prev k = f:(smooth_ xs f k)
 where f = (k * x) + ((1 - k) * prev)
smooth_ [] prev k = []

tau t = if t > 0 then (1 - exp((-1)/t)) else 1.0

examples_smooth = do
 examples "examples_smooth.txt" display exprn isok tests
 where display ((a,b),ok) = "smooth " ++ show a ++ " " ++ show b
       exprn ((a,b),ok) = smooth a b
       isok ((a,b),ok) = exprn((a,b),ok) == ok
       tests = [((0.0,[60..65]),[60..65])]

prop_smooth_ident_0 :: [Double] -> Bool
prop_smooth_ident_0 xs = (smooth 0 xs) ~==~ xs

prop_smooth_ident_1 :: Double -> Bool
prop_smooth_ident_1 x = (smooth 1 (replicate 100 x)) ~==~ (replicate 100 x)

prop_smooth_ident_2 :: Double -> Bool
prop_smooth_ident_2 x = (smooth 60 (replicate 100 x)) ~==~ (replicate 100 x)

prop_tau_limits :: Double -> Bool
prop_tau_limits k = 0 <= tau k && tau k <= 1
```

Examples for

```
examples_tau = do
 examples "examples_tau.txt" display exprn isok tests
 where display (a,ok) = "tau " ++ show a
       exprn (a,ok) = tau a
       isok (a,ok) = (abs (exprn(a,ok) - ok)) < 0.01
       tests = [(0.0,1), -- straight thru
                (1.0,0.632),
                (10.0,0.0951),
                (60,0.0165),
                (600.0,0.001)]

figSmooth1 = fig "figSmooth1.pdf"
   [figLine (impulse,"step", black),
    figLine (smooth 1 $ impulse,"t=1", red),
    figLine (smooth 10 $ impulse,"t=10", green),
    figLine (smooth 60 $ impulse,"t=60", blue),
    figLine (smooth 600 $ impulse,"t=600", brown)
    ]
   figCfg
 where impulse = take 110 $ step01 10 100
```

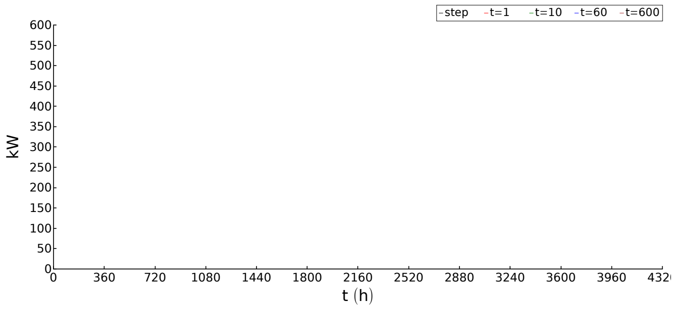

Fig. 9. Smoothing

H. Others

```
generate :: [Double] -> Vector Double
generate vs = fromList vs :: Vector Double
```

I. `maxup`

```
maxup :: Int -> [Double] -> [Double]
maxup n [] = []
maxup n (x:xs) = v:(maxup n xs)
 where v = if xs == []
           then 0
           else max 0 ((maximum $ take n xs)-x)
time :: IO t -> IO t
time a = do
   start <- getCPUTime
   v <- a
   end   <- getCPUTime
   let diff = (fromIntegral (end - start)) / (10^12)
   printf "cputime = %0.3fs\n" (diff :: Double)
   return v

make t f = do
  s <- getArgs
  if makeOk t s
    then do
       putStr $ "MAKING " ++ t ++ "\t"
       time f
       return 0
    else do
         -- putStrLn $ "SKIPPING " ++ t
         return 0
  return 0

makeOk t [] = False
makeOk t ["all"] = True
makeOk t (x:xs) =
  if t == x || isInfixOf x t
  then True
  else makeOk t xs
prop_maxup_0 :: Int -> Property
prop_maxup_0 n = n > 0 ==> (maxup n [1,1,1,1]) == [0,0,0,0]

prop_maxup_1 :: Int -> Property
prop_maxup_1 n = n > 0 ==> (maxmaxup n [1,1,2,1]) == 1

prop_maxup_2 :: Int -> Property
prop_maxup_2 n = n > 0 ==> (maxmaxup n [1,1,0,1]) == 1

prop_maxup_3 :: Int -> Property
prop_maxup_3 n = n > 0 ==> (maxmaxup n [1,1,1,100]) == 99
```

J. `maxmaxup`

```
maxmaxup n [] = 0.0
maxmaxup n xs = maximum (maxup n xs)
```

K. `maxdown`

```
maxdown :: Int -> [Double] -> [Double]
maxdown n [] = []
maxdown n (x:xs) = v:(maxdown n xs)
 where v = if xs == []
           then 0.0
           else abs (x - (minimum $ take n $ xs))
```

L. `maxmaxdown`

```
maxmaxdown n [] = 0.0
maxmaxdown n xs = maximum (maxdown n xs)
tselect tp ts xs = take (ceiling tp) $ drop (round ts) xs
tselect1d = tselect (1 * d)
tselect1w = tselect (1 * w)
tselect1m = tselect (1 * m)
m = 60 :: Double
h = m * 60 :: Double
d = h * 24 :: Double
w = d * 7 :: Double
y = w * 52 :: Double
```

M. `fig` – *fig drawing support*

The `fig` is used to generate PDF charts using Vivian McPhails Plot package [**?**].

```
-- create a record to hold the configuration for a plot
data FigCfg =
 FigCfg {
   plottitle :: String,
   xtitle :: String,
   xlow,xstep,xhigh :: Double,
   xfmt :: String,
   xticks :: Int,
   xgrid :: Bool,
   ytitle :: String,
   ylow, yhigh :: Double,
   yfmt :: String,
   yticks :: Int,
   ygrid :: Bool
   }

-- and define a default one
figCfg = FigCfg {
              plottitle = "",
              xtitle = "t (h)",
              xlow = 0,
              xstep = ((1.0)/(1.0 * h)),
              xhigh = nhrs,
              xfmt = "%.0f",
              xticks = 13,
              xgrid = False,
              ytitle = "kW" ,
              yfmt = "%.0f",
              ylow = 0, yhigh = 600,
              yticks = 13,
              ygrid = False
              }
fig filename lines cfg = do
   let xv = fromList [xlow cfg,
                      xlow cfg + xstep cfg..xhigh cfg] :: Vector Double
   figWrite filename (figNew (xv,lines) cfg)
   figShow filename
figLine (xs,s,c) = line ((generate xs), s) c
figPoint (xs,s,c) = point ((generate xs), s) (Bullet,c)

figLineStats (xs,s,c) =
 line ((generate xs),  s ++ " " ++ figStats xs) c

figWrite filename fig = do
 writeFigure PDF filename (1600,800) fig
figNew ds cfg = do
 figDefaults cfg
 withTitle $ setText $ plottitle cfg
 withPlot (1,1) $ do
   setDataset ds
   figX cfg
   figY cfg
   setRange YAxis Lower Linear (ylow cfg) (yhigh cfg)
   setRange XAxis Lower Linear (xlow cfg) (xhigh cfg)
   setLegend True NorthEast Outside

figShow filename =
 print filename
 -- rawSystem "evince" ["-f", filename]

figX cfg = do
 addAxis XAxis (Side Lower) $ do
   setGridlines Major $ xgrid cfg
   withAxisLabel $ setText $ xtitle cfg
   setTicks Major (TickNumber (xticks cfg))
   setTicks Minor (TickNumber 0)
   setTickLabelFormat $ Printf $ xfmt cfg

figY cfg = do
 addAxis YAxis (Side Lower) $ do
   setGridlines Major $ ygrid cfg
   withAxisLabel $ setText (ytitle cfg)
   setTicks Major (TickNumber (yticks cfg))
   setTicks Minor (TickNumber 0)
   setTickLabelFormat $ Printf $ yfmt cfg

figDefaults cfg = do
 withTextDefaults $ setFontFamily "OpenSymbol"
 withTextDefaults $ setFontSize 28
 withLineDefaults $ setLineWidth 2
 withPointDefaults $ setPointSize 2
 setPlots 1 1
```

N. `figStats` - *statistics in figs*

A simple statistics caculator that returns the `min..average..max` of a list for use in the figs legend.

```
figStats :: [Double] -> String

figStats xs  =
 "\n" ++ ls ++ ".." ++ ms ++ ".." ++ hs
 where ls = printf "%.0f" $ l
       ms = printf "%.0f" $ m
       hs = printf "%.0f" $ h
       l = minimum xs
       m = s/n
       h = maximum xs
       s = sum xs
       n = fromIntegral (length xs) :: Double
```

O. `freq` - *calculate frequency across a range*

Calculate the frequency of an value.

```
freq low n high xs = freq_ low' step' high' xs
 where
   low' = min low $ minimum xs
   step' = (high' - low') / n
   high' = max high $ maximum xs

freq_ :: Double -> Double -> Double -> [Double] -> [(Double,Double,Int)]
freq_ low w high xs =
 if low < high then r:rs
 else []
 where r = (low,(low+w),(length $ filter (between low (low+w)) xs))
       between l h v = l <= v && v < h
       rs = freq_ (low+w) w high xs
```

P. `runLengthCompress`

Run length compression replaces identical sequences in a a list with a tuple containing the value and the number of identical values (the run length). This method is used for:

1) Compressing data for both reading and writing.
2) Writing ASIM compatible data sets.
3) Displaying data to humans.

The `runLengthCompress` function takes a list of data at a given fixed time stamp (typically 1s) and returns the compressed information.

```
runLengthCompress :: [Double] -> [(Double,Int)]

runLengthCompress [] = []
runLengthCompress (x:xs) = (x,cnt+1):(runLengthCompress $ drop cnt xs)
 where cnt = length $ takeWhile (x==) xs
```

The `runLengthDecompress` decompresses the data from `runLengthCompress`.

```
runLengthDecompress :: [(Double,Int)] -> [Double]

runLengthDecompress [] = []
runLengthDecompress ((x,n):xs) = (replicate n x) ++ (runLengthDecompress xs)
```

The following examples may help:

```
examples_runLengthCompress = do
 examples "examples_runLengthCompress.txt" display exprn isok tests
 where display (a,ok) = "runLengthCompress" ++ " " ++ (show a)
       exprn (a, ok) = runLengthCompress a
       isok (a, ok) = exprn(a,ok) == ok
       tests = [(([]),[]),
                (([10]),[(10,1)]),
                (([10]),[(10,1)]),
                (([10, 10]),[(10,2)]),
                (([10, 8]),[(10,1),(8,1)]),
                ((replicate 10 13),[(13,10)]),
                (((replicate 10 13) ++ (replicate 4 3)), [(13,10),(3,4)])
               ]
```

The following properties hold for these functions:

```
newtype SmallDoubleList = SmallDoubleList [Double] deriving (Eq,Show)

instance Arbitrary SmallDoubleList where
 arbitrary = sized $ \s -> do
                n <- choose (0,s `min` 100)
                xs <- vectorOf n (choose (0,10))
                let xs' = (map (significantDigits 0) xs)
                return (SmallDoubleList xs')
 shrink (SmallDoubleList xs) = map SmallDoubleList (shrink xs)

-- runLengthDecompress is the inverse of runLengthCompress
prop_runLength_ident :: SmallDoubleList -> Bool
prop_runLength_ident (SmallDoubleList xs) =
 xs == (runLengthDecompress $ runLengthCompress xs)

-- compression never increases the size of the data
prop_runLength_size :: SmallDoubleList -> Bool
prop_runLength_size (SmallDoubleList xs) =
 (length $ runLengthCompress xs) <= length xs

prop_runLength_max :: SmallDoubleList -> Bool
prop_runLength_max (SmallDoubleList xs) =
 xs == [] || (maximum xs) == (maximum $ fst $ unzip $ runLengthCompress xs)

prop_runLength_min :: SmallDoubleList -> Bool
prop_runLength_min (SmallDoubleList xs) =
 xs == [] || (maximum xs) == (maximum $ fst $ unzip $ runLengthCompress xs)

prop_runLength_len :: SmallDoubleList -> Bool
prop_runLength_len (SmallDoubleList xs) =
 (length xs) == (sum $ snd $ unzip $ runLengthCompress xs)
showRunLengths xs = showRunLengths_ 0 $ runLengthCompress xs

showRunLengths_ :: Double -> [(Double,Int)] -> String
showRunLengths_ t [] = ""
showRunLengths_ t ((a,b):xs) =
 (showTime t) ++ "\t" ++ (showTime (fromIntegral b)) ++
 "\t" ++ (show a) ++ "\n" ++
 showRunLengths_ (t+(fromIntegral b)) xs
 where now = t + (fromIntegral b)

showTime at = printf "%6d:%02d:%02d" (h::Int) (m::Int) (s::Int)
 where s = round(at) `mod` 60
       m = round(at/60) `mod` 60
       h = round(at/3600)
```

Q. `compressDeadband` - *deadband compression*

```
{- ttt implement
compressDeadband :: Double -> Double -> [(Double,Int)] -> [(Double,Int)]
compressDeadband vdb zdb [] = []
compressDeadband vdb zdb xs = compressDeadband_ vdb zdb xs

compressDeadband_ vdb zdb [] = []
compressDeadband_ vdb zdb ((v,n):xs) =
-}
```

R. `significantDigits`

```
significantDigits :: Int -> Double -> Double
significantDigits n f = (fromInteger $ round $ f * (10^n)) / (10.0^^n)

prop_significantDigits_0 :: Int -> Bool
prop_significantDigits_0 v =
 (significantDigits 0 (fromIntegral v)) == (fromIntegral v)

-- ttt review this code and its necessity
```

S. `percent` - *format a percentage*

```
percent a b = 100 * (a/b)
showPercent a b = printf "%.1f" $ percent a b

percentChange a b = (percent a b) - 100
showPercentChange a b = printf "%.1f" $ percentChange a b
```

T. *Approximate equality*

Much of the code above can introduce approximation errors so two operators are defined for approximately equal for `Double` and `[Double]`.

```
(~==) :: Double -> Double -> Bool
a ~== b = abs(a-b) < 0.0000001

(~==~) :: [Double] -> [Double] -> Bool

[] ~==~ [] = True
(a:as) ~==~ [] = False
[] ~==~ (b:bs) = False
(a:as) ~==~ (b:bs) = a ~== b && as ~==~ bs
```

Testing support

APPENDIX C
TESTING

A. `check` *quickCheck property based testing*

The quickCheck library [?] provides property based checking which automatically generates test cases and checks them.against propertiers. All properties begin with `prop_` and are automatically checked.

An example of a property might be that for any list, two `reverse` operations on a list return the original list.

```
prop_reverse_0 :: [Double] -> Bool
prop_reverse_0 xs = reverse (reverse xs) ==  xs
```

In this program all properties will be checked when the command line argument `check` or `all` is passed to it/

```
check = do -- check all prop_* in this program
 putStrLn ""
 $(quickCheckAll) >>= \passed ->
   if passed then checkOk
   else checkFailed

verbosecheck = do -- verbose check all prop_* in this program
 putStrLn ""
 $(verboseCheckAll) >>= \passed ->
   if passed then checkOk
   else checkFailed

checkOk = putStrLn "check: All tests passed."

checkFailed = do -- on failure run a verbose check and fail!
 $(verboseCheckAll)
 error "check: fatal error some checks failed"
```

B. `Examples` - *generating example output*

The `examples` function generates a file of example outputs for use as both examples and tests. The `file` is the result of each function.

```
examples file display exprn isok xs = do
 let s = unlines $ map (example display exprn isok) xs
 putStr s
 writeFile file s

example display exprn isok v =
 "example " ++ (display v) ++
 " ->\n\t" ++ (show result) ++ " " ++
 (if isok v then "OK" else "FAILED")
 where result = exprn v
```

Haskell and Emacs

APPENDIX D
HASKELL, LITERATE PROGRAMMING AND EMACS

A. *Haskell*

Haskell is a lazy functional lanuage which a number of benefits..

- Variables in Haskell do not vary
- A function will always return the same result if passed the same input - no exceptions.
- Functional programs tend to be shorter (usually between 2 to 10 times shorter).

B. *Literate Programming*

Add in an intro to literate programming.

The main.lhs is a LaTeX file with Haskell code fragments delimited by `\begin{code}` and `\end{code}`

Individual components can be included or excluded using the comment package which is configured at the top of `main.lhs`.

Each subsection is typically laid out as:

```
%% pv:bit - description about PV
\begin{pv:bit}
\subsection{PV is interesting}
...
\end{pv:bit}
```

C. *Emacs*

The emacs setup uses orgstruct minor mode which is part of org mode and provides outlining. For the setup below:

```
%% Top level
%%% Next one done
%%% And another
%% Another Top level
```

The emacs setup file (`.emacs`) needs to include something akin to:

```
;; org mode setup
(add-to-list 'load-path "~/imports/org-8.2.6/lisp")
(defun my-org-latex-hook () (interactive)
 (orgstruct-mode)
;;  (orgstruct++-mode) -- evil breaks things
 (setq org-outline-regexp "^%%+ ")
 (setq org-outline-heading-regexp "^%%+ .*$")
 (setq orgstruct-heading-prefix-regexp "^%%+ ")
 (local-set-key [M-tab]
 (lambda () (interactive) (org-cycle t)))
 (local-set-key [tab]
 (lambda () (interactive) (org-cycle))))
(add-hook 'TeX-mode-hook 'my-org-latex-hook)

; mode for main.lhs
(defun my-lhs-mode () (interactive)
 (literate-haskell-mode)
 (my-org-latex-hook))

; hook to activiate it
(add-to-list 'auto-mode-alist '("\\.lhs\\'" . my-lhs-mode))
```

D. *Main Program*

```
main = do
 make "check" check
 -- make "figKalkPvAvailP.pdf" figKalkPvAvailP
 -- print $ events sKalkPvAvailP
 -- make "verbosecheck" verbosecheck
 make "figPvAvailP1" figPvAvailP1
 make "figPvAvailMaxDown120P1" figPvAvailMaxDown120P1
 make "figPvAvailMaxDown120PAboveSpin1" figPvAvailMaxDown120PAboveSpin1
 -- make "examples_fig" examples_fig
 -- make "examples_runLengthCompress" examples_runLengthCompress
 -- make "figLoad1P" figLoad1P
 -- make "figLoad2P" figLoad2P
 -- make "figLoad3P" figLoad3P
 -- make "figLoad4P" figLoad4P
 make "figFuelCurve1" figFuelCurve1
 make "figFuelCurve2" figFuelCurve2
 make "figFuelCurve3" figFuelCurve3
 make "figFuelCurve4" figFuelCurve4
 make "figFuelCurve5" figFuelCurve5
 -- make "figPv1" figPv1
 -- make "figPv2" figPv2
 -- make "figPerfect1" figPerfect1
 -- make "figPerfect2" figPerfect2
 -- make "putPerfect" putPerfect
 -- make "figSimplicitySpin" figSimplicitySpin
 -- make "figSimplicitySmooth" figSimplicitySmooth
 -- make "figSimplicityHyst" figSimplicityHyst
 -- make "figSimplicityCap" figSimplicityCap
 -- make "figSimplicityMinRun" figSimplicityMinRun
 -- make "figSimplicity30P" figSimplicity30P
 -- make "figPredictor30mP" figPredictor30mP
 -- make "figCompare1" figCompare1
 -- make "figCompare2" figCompare2
 make "figSmooth" figSmooth1
 make "examples_fuelUsed" examples_fuelUsed
 make "examples_tau" examples_tau
 make "examples_smooth" examples_smooth
 -- make "examples_minRunTime" examples_minRunTime
 -- make "examples_hysterisisDown" examples_hysterisisDown
 -- make "fuelImprove" fuelImprove
 -- make "fuelImprove2" fuelImprove2 -- disabled for now
 -- write1s "tt.e1s" sLoadMaxUp60sP
 return 0
```

E. *Time series IO support*

Hello

F. `read1s` *– read 1s data*

The `read1s` data is used to read data at 1s time steps in plain old ascii from a file. These files are typically generated by other tools such as SCADA or historians such as PI.

```
read1s filename  = unsafePerformIO $ read1s_ filename
read1s_ :: FilePath -> IO [Double]
read1s_ filename = do
 f <- readFile filename
 let l = lines f
 let s = map (read :: String -> Double) l
 return s
```

G. `write1s` *– write 1s to a file*

Write a 1s data set from `xs` into filename `filename`. This function is typically used for debugging.

```
write1s filename xs = do
 let ls = map (show :: Double -> String) xs
 writeFile filename $ unlines ls
```

APPENDIX E
BITS OF INFO

A. *Colins Comments*

```
From: Colin Bonner <c.bonner@fulcrum3d.com>
Sent: Friday, 20 March 2015 3:29 PM
To: Maker, Phillip; tom.loveard@gmail.com; antonin braun
(braun.antonin@gmail.com)
Subject: Re: Record of conversation

Hi Gents,

Great to touch base with everyone today.

I'd like to comment on the data analysis and thought an e

Regardless of the comparative metric, we are aiming for a
IMO this means we need a specific number of data points i
in wind energy industry when validating  remote sensing d
minimum period of 3 months is required AND 6 data points
bin between 4.0~4.5, 4.5~5.0, ..., 15.5~16.0m/s. The mini
higher order stats and is quite commonly observed at the
reality of not being able to test forever...

I'd like to propose the tests run for a minimum of 3 mont
```

PWC's system OR 6 months occurs. This does introduce the issue of defining the "events". First thoughts would to be define something like:
1) Event A: drop in PV that lasts < 30 sec -- this is interesting form a bat/fly wheel perspective.
2) Event B: drop in PV that lasts >= 30sec & < 2 min -- just interesting?
3) Event C: drop in PV that lasts > 2 min -- interesting for spinning reserve management.

Drops in PV would be defined as per tender RFQ. Might only need 2 events, or define events in terms of Energy (power loss x time)... I'd like to see N be 500~1000. Phil probably has history data from Bullman to see how realistic this is. F3D's work further south suggests this could happen in a few months.

Regarding the analysis for the paper, I think it's essential to do the method 2 Phil drafted. The reality is that correct and incorrect predictions do not have the same weight for hybrid grids when grid stability is included in the analysis (more so on high pen).
I know AEMO/ANU/NICTA/CSIRO are actively researching how to define the "cost" of errors in med/high penetration of PV for their machine learning algo for solar forecasting for this very reason (They are looking at sky imagery, sat imag and distributed irradiance sensors). AFAIK they are heading towards a sqrd-error + penalty. A financial analysis with a penalty for missed predictions (similar to PPA's for bat storage systems) would be very interesting.

I'm putting this out there for discussion.

Cheers,
Colin

Colin Bonner
Director

SODAR | RESOURCE MONITORING | NOISE MONITORING |
CLOUD TRACKING
Level 11, 75 Miller St, NORTH SYDNEY, NSW 2060 AUSTRALIA
M +61 414 785 489 | info@fulcrum3D.com | www.fulcrum3D.com

From: Maker, Phillip <Phillip.Maker@powerwater.com.au>
Sent: Friday, 20 March 2015 4:37 PM
To: Colin Bonner; tom.loveard@gmail.com; antonin braun (braun.antonin@gmail.com)
Subject: Record of conversation

Gentlemen,

Thanks for the meeting, I hope it was worthwhile, any suggestions taken on board.
Here is my summary of the meeting:

1. Agreement in Principle on things in the kickoff document (at least for now).
2. Ill rewrite it a bit next week and forward it to all parties for signoff/ack.
3. Kit delivery times: could everyone just confirm these timings next but my notes Indicate:
a. Colin 3rd week APR, 2 week testing in Darwin
b. Antonin 3rd week MAY, 2 weeks testing in Darwin
c. Tom fine with APR, 2 weeks testing.
4. PWC will assist in the testing as your kit arrives, particularly the MODBUS stuff.
a. Im guessing we do a formal test of everything MODBUS in Darwin
b. And perhaps all the update/download procedures etc.
c. Well provide a Telstra 3G network connection for you dial in on.
5. Re mounting: PWC to provide a drawing (these are a bit mysterious sometimes In the north but Ill give it a go).
6. In general well be responsible for the physical install with your advice. If we need to Buy a few poles/brackets that is all with us. (including cost).
7. Phil to ring atonin re MODBUS stuff
8. Princples/Paper due next week for initial review

Anything else?

Ta