**Function description**

In the following, all Matlab functions that were generated for the IKAP approach are descripted in detail. The functions can be found within the functions folder as part of this supplement, or at out github repository:

https://github.com/marcel-mischnik/IKAP.git

Step 1-1: SearchMotifs.m

function [data]=SearchMotifs(data,PSP)

% Searches the PhosphoSitePlus database (download Kinase-Substrate-Dataset.gz

% at http://www.phosphosite.org/staticDownloads.do) for kinases that are

% known to phosphorylate the central amino acid of the given motif. The motifs

% can have an arbitrary length, however all motifs should be equally long.

% The function adds a further column to the data containing the respective kinase(s).

% PSP should be a two-column cell array generated from the downloaded file with

% kinase names (e.g. gene names) in the first column and motif in the second (no headers).

% The motifs in PSP are assumed to be 15 amino acids long.

% Data should be a cell array beginning with one column for the protein name (or ID)

% the phosphosite is located on, followed by one column for the motif and a

% variable number of numeric columns. All IKAP functions assume a total number

% of two annotation columns. If more annotation columns shall be included, the functions

% have to be altered where neccessary. The header row should be removed.

data=[data cell(size(data,1),1)];

le=length(data{2,2});

mdata=(le+1)/2;

mpsp=8;

if isnumeric(data{1,3})

start=1;

else

start=2;

end

if mdata>=mpsp

for i=start:length(data)

motdata=data{i,2}(mdata-7:mdata+7);

s=strcmp(motdata,PSP(:,2));

if sum(s)>0

x=find(s==1);

for j=1:length(x)

if isempty(data{i,end}) && ~isempty(PSP{x(j),1})

data{i,end}=PSP{x(j),1};

elseif ~isempty(data{i,end}) && ~isempty(PSP{x(j),1})

data{i,end}=[data{i,end} PSP(x(j),1)];

end

end

end

end

else

for i=1:length(PSP)

motpsp=PSP{i,2}(8-mdata:8+mdata);

s=strcmp(motpsp,data(:,2));

if sum(s)>0

x=find(s==1);

for j=1:length(x)

if isempty(data{x(j),end})

data{x(j),end}=PSP{i,1};

else

data{x(j),end}=[data{x(j),end} PSP{i,1}];

end

end

end

end

end

for h=start:length(data)

data{h,end}=upper(data{h,end});

if ischar(data{h,end})

data{h,end}=cellstr(data{h,end});

end

data{h,end}=unique(data{h,end});

end

end

The first function searches the data for motifs that are known from the PSP database to be phosphorylated by a certain kinase. The database can be downloaded from http://www.phosphosite.org/staticDownloads.do. It should be reduced to one column containing the kinase names and one containing the motifs. Since the exact position within the motif is of great importance, the function compares the strings which are left and right from the central amino acid to those of the motifs in the database. It adds a column with the kinases found to the data matrix and takes as an input the original dataset as a cell array, the PSP database as a two-column cell array and the column containing the motif in the dataset. The data matrix should begin with a column containing the protein IDs (e.g. gene names) followed by a column for the motifs and a variable number of data columns. The motifs should not be longer than 15 amino acids, the phosphorylated amino acid should be in the middle.

Step 1-2: MakeKin.m

function [data\_red,kin]=MakeKin(data,kcolumn,kproteome)

% Produces a list of kinases (kin) that are found by search\_motifs.m.

% kcolumn corresponds to the column number of the data matrix containing the kinases.

%

% If proteome data is available (optional one column cell array containing protein IDs (kproteome)),

% the list is reduced to those kinases which are present in the proteome.

% Generates a new data array (data\_red) that only includes those sites for which a kinase is known.

kin=cell(1,1);

for i=1:length(data)

kin=[kin data{i,kcolumn}];

end

kin(1)=[];

kin=unique(kin);

if nargin==3

kpr2={};

for i=1:length(kin)

k1=kin{i};

for j=1:length(kproteome)

k2=kproteome{j,1};

f=findstr(k1,k2);

if ~isempty(f)

kpr2=[kpr2 k1];

end

end

end

kpr2(1)=[];

kpr2=unique(kpr2);

kin=kpr2;

end

kin(1)=[];

data\_red=cell(1,kcolumn);

for j=2:length(data)

b=0;

if ~isempty(data{j,kcolumn})

for i=1:length(data{j,kcolumn})

if sum(strcmp(kin(:),data{j,kcolumn}{i}))>0

b=1;

end

end

if b==1

data\_red=[data\_red; data(j,:)];

end

end

end

data\_red(1,:)=[];

kin=kin';

end

This function creates a list of the kinases that are known to phosphorylate at least one motif in the dataset. If additionally proteome data are available (optional 3rd argument, list of protein names) the function reduces the kinase list to those which are present in the proteome. Finally, it produces a new data matrix which only contains those phosphosites for which a kinase is known.

Step 1-3: CreateTT.m

function [a]=CreateTT(data\_red,kcolumn,kin)

% Creates a truth table of logicals (a), indicating if a phosphosphosite is

% phosphorylated by a certain kinase. kcolumn is again the column containing the kinases.

a=zeros(size(data\_red,1),size(kin,1));

for i=1:length(data\_red)

for j=1:length(data\_red{i,kcolumn})

kd=data\_red{i,kcolumn}{j};

for l=1:length(kin)

kk=kin{l};

f=strcmp(kd,kk);

if f>0

a(i,l)=1;

end

end

end

end

end

The third function creates a truth table out of the reduced dataset, with 1 for true interactions and 0 for false. This will be used as a mapping in the next step to estimate the kinase activities and affinity parameters on the basis of only those measurement values, the respective kinases have an influence on.

Step 1-4: FitActivities.m using ComputeCostGlobal.m, ComputeCostLocal.m and APreadout.m

function [AP,AP\_tab,K,cost,mincost]=FitActivities(a,data\_red,kin,ncon,iter)

% Fits the activities of all kinases in kin to the measured data in data\_red using

% the cost functions ComputeCostGlobal.m and ComputeCostLocal.m as well as the truth table a.

% Ncon corresponds to the number of conditions tested (number of columns

% with measured values (6 in case of the HeLa data, 8 in case of the insulin data)).

% Iter determines the number of iterations (for reliable results this should be at least 5).

% The output AP is a list of affinity parameters, AP\_tab allocates these values to their

% respective kinase-target-pairs using the function sreadout. K contains the fitted kinase activities

% as a matrix with ncon columns and length(kin) rows. Cost is a matrix containing all calculated costs,

% mincost a scalar representing the best optimum found.

% Parameter bounds and starting values should be modified appropriately.

m=size(a,1);

n=size(a,2);

[x,~]=find(a==1);

sc=length(x);

tsm=zeros(sc,iter);

tkm=zeros(n,ncon,iter);

cost=[];

lb1=0.1\*ones(sc,1);

ub1=10\*ones(sc,1);

lbk=-15; % Lower bound for kinase activities. Modify according to your needs.

ubk=15; % Upper bound for kinase activities, Modify according to your needs.

lb2=lbk\*ones(n,1);

ub2=ubk\*ones(n,1);

options = optimoptions('fmincon', 'GradObj', 'on');

parfor i=1:iter

J1=10000;

J2=1000;

ts=10\*rand(sc,1);

tk=15+(-15\*rand(n,ncon)); % Starting values for kinase activities. Modify according to your needs.

while J1-J2>10

[ts,J1]=fmincon(@(ts)(ComputeCostGlobal(ts,tk,a,data\_red,ncon)), ts, [], [], [], [], lb1, ub1, [], options);

J=zeros(1,ncon);

tsm(:,i)=ts;

for j=3:ncon+2

tki=tk(:,j-2);

[tki,J(1,j-2)]=fmincon(@(tki)(ComputeCostLocal(ts,tki,a,data\_red(:,j))), tki, [], [], [], [], lb2, ub2, [], options);

tk(:,j-2)=tki;

J2=sum(J);

end

tkm(:,:,i)=tk;

cost=[cost; [J1 J2]];

end

end

J=zeros(1,iter);

for i=1:iter

[J(i)]=ComputeCostGlobal(tsm(:,i),tkm(:,:,i),a,data\_red,ncon);

end

[mincost,g]=min(J);

AP=tsm(:,g);

K=tkm(:,:,g);

[AP\_tab]=APReadout(AP,a,data\_red,kin);

end

FitActivities estimates the activities of the kinases in kin using the Matlab built-in function fmincon. The number of iterations can be specified using the input argument iter. Each iteration starts with randomized parameter vectors and cycles between an optimization of the affinity parameters by means of the global cost function and an optimization of the kinase activities for each condition by means of the local cost function. Since the gradients for the parameters are delivered by the cost functions, the ‘GradObj’ argument in the option settings should be turned to ‘on’. The while-loop continues as long as the difference between both costs is larger than a certain value, in this case 10. The upper and lower bounds for parameter estimation as well as the starting values should be modified according to the user´s needs. The output variables AP and K contain the estimated affinity parameters and kinase activities, cost is a list of all calculated costs with mincost being the lowest. This function can also be executed on multiple cores using the parallel computing toolbox. In this case, the first for-loop needs to be replaced by a parfor-loop.

ComputeCostGlobal.m

function [J,grad]=ComputeCostGlobal(ts,tk,a,data\_red,ncon)

% Calculates the cost and the gradients for the affinity parameters over

% all conditions using the affinity factors (ts), the kinase activities (tk),

% the truth table (a) and the data (data\_red).

% Ncon corresponds to the number of conditions measured (number of data

% columns).

m=size(a,1);

n=size(a,2);

J=0;

grad=[];

p1=zeros(m,n);

[x,y]=find(a==1);

for i=1:length(x)

p1(x(i),y(i))=ts(i);

end

for i=1:ncon

p1=[p1; tk(:,i)'];

end

for j=1:ncon

for i=1:m

if ~isnan(data\_red{i,j+2})

a\_ex=p1(m+j,:).\*p1(i,:);

avg=sum(a\_ex)/sum(p1(i,:));

if ~isnan(avg)

J=J+(avg-data\_red{i,j+2})^2;

end

end

end

end

for i=1:m

for j=1:n

g=0;

for l=1:ncon

if ~isnan(data\_red{i,l+2}) && a(i,j)~=0

g=g+(2\*((p1(m+l,j)\*p1(i,j))/sum(p1(i,:))-data\_red{i,l+2})\*(p1(m+l,j)\*sum(p1(i,:))-sum(p1(m+l,:).\*p1(i,:)))/(sum(p1(i,:)))^2);

end

end

if a(i,j)~=0

grad=[grad; g];

end

end

end

end

The global cost function calculates the cost over all conditions and the gradients for the affinity parameters. The global cost is defined as the sum of the squared distances of the measured phosphosites and the mean of the products of activity and affinity of all kinases phosphorylating the phosphosite, over all conditions. The gradient for each affinity parameter is given as its partial derivative of the cost function.

ComputeCostLocal.m

function [J,grad]=ComputeCostLocal(ts,tki,a,b)

% Calculates the cost and the kinase activity gradients for the condition

% given in b (one data column of data\_red) using the affinity parameters

% ts, the local kinase activities tki and the truth table a.

m=size(a,1);

n=size(a,2);

J=0;

grad=zeros(n,1);

p=[ts; tki];

p1=zeros(m,n);

[x,y]=find(a==1);

sc=length(x);

for i=1:sc

p1(x(i),y(i))=p(i);

end

p1=[p1; p(sc+1:end)'];

for i=1:m

if ~isnan(b{i})

a\_ex=p1(m+1,:).\*p1(i,:);

avg=sum(a\_ex)/sum(p1(i,:));

if ~isnan(avg)

J=J+(avg-b{i})^2;

end

end

end

for l=1:n

N=0;

for i=1:m

if ~isnan(b{i}) && a(i,l)~=0

a\_ex=p1(m+1,:).\*p1(i,:);

avg=sum(a\_ex)/sum(p1(i,:));

if ~isnan(avg)

N=N+(2\*(avg-b{i})\*(p1(i,l)/sum(p1(i,:))));

end

end

end

grad(l)=N;

end

end

The local cost function calculates the cost for one condition and derives the gradients for the kinase activities within the respective condition. As in the global case, the cost is defined as the sum of the squared distances of the measured values and the means of the activity\*affinity products of the assigned kinases, but without summing over all conditions.

APreadout.m

function [AP\_tab]=APReadout(AP,a,data\_red,kin)

% Creates a cell array containing the affinity parameters for each

% kinase-target-link.

[x,y]=find(a==1);

sc=length(x);

AP\_tab=cell(sc,3);

for i=1:sc

AP\_tab{i,1}=data\_red(x(i),1);

AP\_tab{i,2}=kin(y(i));

AP\_tab{i,3}=AP(i);

end

end

This function creates a cell array that allocates the estimated affinity parameters to the respective kinase-target-links.

Step 1-5 (optional): IdentKin.m using FitIdent.m

function [plJ,plk]=IdentKin(ks,a,c,AP,K,data\_red,ncon)

% Calculates identifiability profiles for the kinases given in ks. This

% should be a vector containing the kinase numbers (not names).

% Inputs:

% ks: vector with kinase numbers

% a: truth table

% c: column number of the condition in data\_red the identifiabiliy shall be calculated on

% AP: vector with fitted affinity parameters

% K: matrix with fitted kinase activities

% data\_red: reduced dataset

% Outputs:

% plJ: matrix with calculated costs along the dimension of the respective kinase

% plk: matrix with tested kinase values

p=[AP; K(:,c-2)];

plJ=zeros(length(ks),7);

plk=zeros(length(ks),7);

adds=-3:3;

[x,~]=find(a==1);

sc=length(x);

lkpr=size(a,2);

lbk=-15; % Lower bound for kinase activities. Modify according to your needs.

ubk=15; % Upper bound for kinase activities. Modify according to your needs.

options = optimoptions('fmincon', 'GradObj', 'on');

parfor l=1:length(ks)

i=ks(l);

for j=1:7

pnew=p;

pnew(sc+i)=p(sc+i)+adds(j);

plk(l,j)=p(sc+i)+adds(j);

if j==4

[plJ(l,j)]=ComputeCostGlobal(AP,K,a,data\_red,ncon);

else

lb=[0.1\*ones(sc,1); lbk\*ones(lkpr,1)];

ub=[10\*ones(sc,1); ubk\*ones(lkpr,1)];

lb(sc+i)=pnew(sc+i);

ub(sc+i)=pnew(sc+i);

plJ(l,j)=FitIdent(a,pnew,K,lb,ub,data\_red,options,sc,c,ncon);

end

end

end

end

IdentKin.m performs an identifiability analysis of the estimated kinase activities using the fitting function FitIdent.m, which makes use of fmincon. The activities at the condition c of the kinases given in ks (numbers as they are listed in kin) are decreased and increased in a step-wise manner (increment ranging from -3 to 3) and fixed, followed by an optimization of the remaining parameters (both affinities and activities). The function outputs the matrices plJ, containing the obtained costs, and plk, containing the activity values with which the optimizations were performed. By applying a parfor-loop, it can be executed in parallel.

FitIdent.m

function [J]=FitIdent(a,pnew,K,lb,ub,data\_red,options,sc,c,ncon)

% Fitting function for identifiability analysis. Should only be used as

% part of IdentKin.m.

J1=100000;

J2=10000;

ki=K;

ki(:,c)=pnew(sc+1:end);

ts=pnew(1:sc);

while J1-J2>10

[ts,J1]=fmincon(@(ts)(ComputeCostGlobal(ts,ki,a,data\_red,ncon)),ts, [], [], [], [], lb(1:sc), ub(1:sc), [], options);

J=zeros(1,size(K,2));

for j=3:size(data\_red,2)-1

kp=ki(:,j-2);

[kin,J(1,j-2)]=fmincon(@(kp)(ComputeCostLocal(ts,kp,a,data\_red(:,j))), kp, [], [], [], [], lb(sc+1:end), ub(sc+1:end), [], options);

ki(:,j-2)=kin;

J2=sum(J);

end

end

J=J2;

end

This function performs the optimization task within the identifiability analysis.

Step 2-1: ComputeDistance.m

function [dist,valids]=ComputeDistances(data,ncon,kin,K)

% Calculates p-values for each kinase-phosphosite-pair by means of a

% correlation coefficient.

% Inputs:

% data: complete data matrix

% ncon: number of conditions

% kin: kinase list

% K: estimated kinase activities

% Outputs:

% dist: cell array allocating a p-value to each combination of

% phosphosites (y-axis) and kinases (x-axis).

% valids: numeric matrix indicating on how many valid measurement values

% the correlation coefficient is calculated in each case.

dist=cell(size(data\_red,1),size(kin,1)+2);

valids=zeros(size(data\_red,1),1);

for i=1:size(data\_red,1)

da=data\_red(i,1:size(data\_red,2));

da(strcmp('NaN',da))={NaN};

dist(i,1:2)=da(1:2);

dat=cell2mat(da(3:2+ncon));

d=find(~isnan(dat));

for m=1:length(kin)

[~,P]=corrcoef(dat(d),K(m,d));

minp=min(P);

if ~isnan(minp(1))

dist{i,m+2}=minp(1);

else

dist{i,m+2}=1;

end

valids(i)=length(d);

end

end

dist=[cell(1,length(kin)+2); dist];

dist(1,3:end)=kin';

end

The first function of part 2 computes correlation coefficient based p-values for all kinase-phosphosite combinations. The resulting distance matrix is a cell array with kinases on the x-axis and phosphosites on the y-axis. In addition it produces the matrix valids, which contains the number of valid measurement values that went into the calculation of each p-value.

Step 2-1-2 (optional): MakeDataVal.m

function [data\_val]=MakeDataVal(data)

% Creates a dataset from which the phosphosites that were used for

% parameter estimation are removed. This guerantees for an unbiased validation.

data\_val={};

for i=1:length(data)

if isempty(data{i,end})

data\_val=[data\_val; data(i,:)];

end

end

end

This function generates a dataset from which the links that have an annotated kinase in PSP are removed. This should be done previous to validation.

Step 2-2: MakePsig.m

function [psig]=MakePsig(dist,kin,q)

% Produces a list of significant links applying the Benjamini-Hochberg procedure

% with a desired false discovery rate q.

% The resulting cell array contains one column with protein IDs, one with

% kinase IDs and one with the respective p-value.

[~,crit\_p]=fdr\_bh(cell2mat(dist(2:end,3:end)),q);

if iscell(dist)

di=cell2mat(dist(2:end,3:end));

else

di=dist;

end

[x,y]=find(di<=crit\_p);

psig=cell(length(x),5);

psig(:,1)=dist(x,1);

psig(:,2)=kin(y);

psig(:,3)=dist(x,2);

for i=1:length(psig)

psig{i,4}=di(x(i),y(i));

if isnumeric(psig{i,1}) || isempty(psig{i,1})

psig{i,1}='NAN';

else

psig{i,1}=upper(psig{i,1});

end

end

psig(:,5)={0};

end

Psig is a list of all kinase-substrate pairs that have a p-value below the given false discovery rate q, calculated with the Benjamini-Hochberg method.

Step 2-3: MakePnsig.m

function [pnsig]=MakePnsig(dist,psig,kin)

% Produces a list of randomly selected links (pnsig) that has the same size as

% psig.

pnsig=cell(length(psig),5);

r1=randi(size(dist,1)-1,length(psig),1);

r2=randi(length(kin),length(psig),1);

cd=dist(2:end,3:end);

sites=dist(2:end,1:2);

pnsig(:,1)=sites(r1,1);

pnsig(:,2)=kin(r2,1);

pnsig(:,3)=sites(r1,2);

for i=1:length(psig)

pnsig(i,4)=cd(r1(i),r2(i));

if isempty(pnsig{i,1}) || isnumeric(pnsig{i,1})

pnsig{i,1}='NAN';

end

end

pnsig(:,5)={0};

pnsig(:,1)=upper(pnsig(:,1));

end

This function is used to create a list of the same size as psig but with randomly selected links. By comparing psig and pnsig within the following functions, it is possible to validate the newly found kinase-substrate links.

Step 2-4: validDB.m

function [psig,pnsig,pDB]=validDB(psig,pnsig,DB)

% Performs a database search with the kinase-target-links in psig and pnsig

% and calculates a p-value based on a fisher exact test. The database

% variable should be a cell array with two columns, the first for targets,

% the second for kinases. Make sure the identifiers conform to those in your data (type, case).

a=psig(:,1);

b=psig(:,2);

c=DB(:,1);

d=DB(:,2);

e=pnsig(:,1);

f=pnsig(:,2);

parfor i=1:length(psig)

st=strcmp(a{i},c);

sk=strcmp(b{i},d);

snt=strcmp(e{i},c);

snk=strcmp(f{i},d);

for j=1:length(st)

if st(j)==1 && sk(j)==1

psig{i,5}=1;

end

if snt(j)==1 && snk(j)==1

pnsig{i,5}=1;

end

end

end

su=sum(cell2mat(psig(:,5)));

su1=sum(cell2mat(pnsig(:,5)));

pDB=fexact(su,2\*length(psig),su+su1,length(psig));

end

This function searches the database of your choice for the kinase-target-links in psig and pnsig and calculates a p-value for the positive findings based on a fisher exact test. If the pair is found, the function assigns a 1 to the respective row in the fifth column of psig or pnsig. It can be executed in parallel. If no parallelization is used, the parfor-loop needs to be replaced by a normal for-loop.

Step 2-5: validMOT.m

function [psig,pnsig,pSUM,pNUM]=validMOT(psig,pnsig,nwk)

% Searches for motifs in the NetworKIN database and calculates p-values

% based on the respective likelihoods using a fisher exact test. The

% database can be downloaded at http://www.networkin.info/donwload.shtml.

% pSUM is computed based on the sum of the likelihoods in psig and pnsig,

% whereas pNUM refers to the number of likelihoods larger

% than 1. The variable nwk should be a cell array with four columns:

% targets, likelihoods, kinases, motifs.

psig(:,6)={0};

pnsig(:,6)={0};

f=nwk(:,4);

g=psig(:,3);

h=pnsig(:,3);

bs=cell(length(nwk),1);

b1s=cell(length(nwk),1);

le=length(psig{1,3});

les=(le-1)/2;

parfor i=1:length(nwk)

s1=strfind(g(:),f{i});

s2=strfind(h(:),f{i});

[x1]=cellfun(@isempty,s1);

[bs{i}]=find(x1==0);

[x2]=cellfun(@isempty,s2);

[b1s{i}]=find(x2==0);

end

for i=1:length(bs)

if ~isempty(bs{i})

for j=1:length(bs{i})

if strcmp(psig{bs{i}(j),2},nwk{i,1}) && strcmp(psig{bs{i}(j),1},nwk{i,3})

if strcmp(nwk{i,4}(1:5),psig{bs{i}(j),3}(les-4:les)) || strcmp(nwk{i,4}(7:end),psig{bs{i}(j),3}(les+1:les+5))

psig{bs{i}(j),6}=nwk{i,2};

end

end

end

end

if ~isempty(b1s)

for j=1:length(b1s{i})

if strcmp(pnsig{b1s{i}(j),2},nwk{i,1}) && strcmp(pnsig{b1s{i}(j),1},nwk{i,3})

if strcmp(nwk{i,4}(1:5),pnsig{b1s{i}(j),3}(les-4:les)) || strcmp(nwk{i,4}(7:end),pnsig{b1s{i}(j),3}(les+1:les+5))

pnsig{b1s{i}(j),6}=nwk{i,2};

end

end

end

end

end

m1=floor(sum(cell2mat(psig(:,6))));

m2=floor(sum(cell2mat(pnsig(:,6))));

sigmot1=length(find(cell2mat(psig(:,6))>=1));

sigmot2=length(find(cell2mat(pnsig(:,6))>=1));

pSUM=fexact(m1,2\*length(psig),m1+m2,length(psig));

pNUM=fexact(sigmot1,2\*length(psig),sigmot1+sigmot2,length(psig));

end

This function searches the NetworKIN database for the motifs in psig and pnsig. If the motif is found, it notes the respective likelihood in the sixth column of psig or pnsig. Based on these columns, the function calculates the p-values pSUM, which is based on the sum of the likelihoods in psig and pnsig, and pNUM, which corresponds to the number of likelihoods that are larger than 1. The first loop can be executed in parallel.