S.1 Supplementary Information

S.1.1 Dependencies of GeneSPIDER

We here list and discuss external packages that GeneSPIDER depend on. Except for the foremost dependency MATLAB[16], these packages are freely available online for academic use. No dependency is currently a hard dependency, meaning that except for specific functionality, most of GeneSPIDER can be used without these packages.

- git for easily keeping up to date with the GeneSPIDER toolbox.
- Glmnet[9] is used within wrapper functions, such as `Methods.Glmnet`, `Methods.Bolasso`.
- JSONlab for exporting to storage format .json or .ubj.
- xml4mat is necessary for exporting data to the xml storage format.
- CVX for disciplined convex programming in some of our in-house inference methods.
- RInorm[18] for robust network inference using the algorithm by Nordron AB.
- ARACNe2 for the ARACNe wrapper. Configuration files needs to be set as per instructions for ARACNe default or per users specific use cases. The PATH to the ARACNe home directory needs to be set before MATLAB is started.

S.1.2 Installation instructions

To fetch the GeneSPIDER repository run the command:

```
git clone git@bitbucket.org:sonnhammergrni/genespider.git ~/src/genespider
```

or download it from `https://bitbucket.org/sonnhammergrni/genespider` to `~/src/genespider`. Change to the directory where you downloaded the repository by `cd ~/src/genespider`. Next, to fetch the complete GeneSPIDER package run the following:

```
git submodule init
git submodule update
```

GeneSPIDER will be available after adding the path `~/src/genespider` to your MATLAB path with the command:

```
addpath('~/src/genespider')
```

To develop and keep track of changes for each submodule separately, you need to check out the master branch from each submodule with the command `git checkout master`. Each toolbox is expecting to be treated as a MATLAB toolbox *i.e.* that a + prepended on each directory and the parent directory is added to the MATLAB path.

S.1.3 Notation used for steady-state data

Assuming that data is recorded during steady-state, the system (1) simplifies to a linear mapping

$$Y = -\breve{A}^{-1}(P - F) + E. \tag{S2}$$

Here $Y \triangleq [y_1, \ldots, y_M]$ is the measured steady-state response matrix after applying the perturbations $P \triangleq [p_1, \ldots, p_M]$ in M experiments and $\breve{A}$ is the interaction matrix.

An alternative representation that is commonly used in regression problems is obtained by taking the transpose of the variables and "true" network model. We obtain the matrix form of the standard linear data model used in errors-in-variables regression problems by introducing the notation used for regressors $\Phi \triangleq [\phi_1, \ldots, \phi_j, \ldots, \phi_N] = Y^T$, regressands $\Xi \triangleq [\xi_1, \ldots, \xi_i, \ldots, \xi_N] = -P^T$, regressor errors $\Upsilon \triangleq [\upsilon_1, \ldots, \upsilon_j, \ldots \upsilon_N] = E^T$, and regressand errors $\Pi \triangleq [\varepsilon_1, \ldots, \varepsilon_i, \ldots \varepsilon_N] = -F^T$.

$$\Phi = \breve{\Phi} + \Upsilon, \qquad \Xi = \breve{\Xi} + \Pi \tag{S3a}$$

$$\breve{\Phi}\breve{A}^T = \breve{\Xi} \qquad \Phi, \Xi \in \mathbb{R}^{M \times N}. \tag{S3b}$$

S.1.4 Toolboxes

S.1.4.1 Data structure toolbox

This toolbox provides the two main data type classes for networks and experimental data: `Network` and `Dataset`. This division reflects the fact that one network is commonly used to generate many different *in silico* datasets and that experimental data collected from an unknown network need to be stored without any network. Table S.1 lists all classes and functions available within this toolbox.

Storing and loading data and networks is an important part of network inference work, and we have therefore opted to include multiple storage formats for the data types. These include MATLAB's native format `.mat` files, which make it easy to save and load data, *Extensible Markup Language* (XML), specifically MATLAB markup language (mbml)[2] as `.xml` files, and *JavaScript Object Notation* (JSON), which offers a wide range of possibilities for sharing and importing data in MATLAB and other languages. JSON comes with both the standard `.json` format, but also as a binary version, Universal Binary JSON `.ubj`. Despite its name, the export2Cytoscape function also imports networks.Three normalisation methods are available for a dataset object, standard normalization $\hat{x}_i = \frac{x_i - \mu_i}{\sigma_i}$, min max range normalisation $\hat{x}_i = \frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)}$, and unit length normalisation $\hat{x}_i = \frac{x_i}{||x_i||}$. Here $\hat{x}_i$ is the normalised variable or sample of x_i. μ is the mean of sample i and σ the standard deviation, $||.||$ is the 2 norm and $\min$ and $\max$ are the minimum and maximum operators. Noise estimates needs to be redone after normalisation to be used to estimate new data properties.

Table S.1 Contents of the `datastruct` toolbox.

Class or function	Description
Dataset	Stores a data set consisting of a perturbation (P) and response (Y) matrix
Network	Stores a network matrix (A)
RSS	Calculation of the residual sum of squares for responses and perturbations
cutSym	Removes links in a symmetric matrix with a probability to be in or out degree
expectedSNRv	Calculation of the expected $\mathrm{SNR}_{\phi\ \mathcal{N}(\mu,\lambda)}$
export2Cytoscape	Exports a network to a tsv file that can be loaded in Cytoscape, as well as imports network files
optimalRandomP	Generates perturbations that counteract signal attenuation based on SVD of Y
randomNet	Creates a random network with N nodes and specific sparseness with no self loops
scalefree	Create a scale-free network with N nodes and specific sparseness
smallworld	Generate a small-world network
stabalize	Weights a static network structure
simts	Simple simulation of a time-series response of the linear ODE model
weightP	Adjusts elements of P to bring the singular values of Y close to one

S.1.4.1.1 The Network class is a container for networks, in this case a linear model, represented by a matrix A. All methods are listed in Table S.2. Note that some native MATLAB functionality is provided for this data type, such as `svd`, `logical`, `sign`, and `size` operations, and that also a method `fetch` for getting networks from the online repository at `https://bitbucket.org/sonnhammergrni/gs-networks` is provided. The Network class is capable of handling sparse matrices.

Table S.2 Methods in the `Network` class.

Method	Description
fetch	Get GeneSPIDER networks from the online repository
load	Loads a dataset/network back in to a `datastruct`
save	Saves a `datastruct` object to file, as a .mat, .json, .ubj, or .xml file.
populate	Populates the `Network` object with matching fields of an input struct
nnz	Returns the number of non zero entries in the network
size	Returns the size of the network
sign	Returns the signed structure of the network
logical	Returns the logical structure of the network
svd	Returns the singular values of A. To get the singular vectors use `svd(net.A)`
view	Makes a rough graphical network plot using biograph

S.1.4.1.2 The Dataset class is a container for data sets, i.e. perturbation and response data from experiments. It provides a number of functionalities related to data handling, listed in Table S.3.

Table S.3 Methods in the `Dataset` class.

Method	Description
bootstrap	Bootstraps a new data set from the old data set
eta	Calculates the sample-wise linear dependence η of Y and P
gaussian	Generates Gaussian noise matrices E and F with variance λ
include	Returns samples that can be include in LOOCO based on the η limit
populate	Populates the `Dataset` object with matching fields of an input struct
response	Returns the noisy steady-state response of the network
save	Saves a `datastruct` object to file, as a .mat, .json, .ubj, or .xml file
load	Loads a dataset file back in to a `datastruct`
fetch	Loads a dataset from the on-line repository via URL or name
scaleSNR	Scales the noise variance to achieve the desired SNR
std	Returns the standard deviation of all data points
true_response	Returns the noise-free steady-state response of the network
w_eta	Calculates SVD based sample-wise linear dependence η of Y and P
without	Creates a `Dataset` without sample i
std_normalize	Creates a `Dataset` with standard normalised expression values
range_scaling	Creates a `Dataset` with `min max` scaling
unit_length_scaling	Creates a `Dataset` with unit length scaling

S.1.4.2 Analysis toolbox

Provides fundamental, as well as complex functions for analysing data and models. These are listed in Table S.4.

Table S.4 Contents of the `analyse` toolbox.

Class or function	Description
Model	Calculates properties related to the supplied `Network`
CompareModels	Calculates similarity measures of weighted network adjacency matrices
Data	Calculates data properties of the supplied `Dataset`

S.1.4.2.1 The Model class is aimed at analysing models/networks. It provide measures to quantify the properties of the network. The methods are listed in Table S.5. Specification of how to treat links (directed or undirected) in measures that depend on the link type is supported.

Table S.5 Methods of `Model` class.

Method	Description
alpha	Returns the significance level (default 0.01)
analyse_model	Batch calculation of almost all the measures below (used internally)
calc_proximity_ratio	Calculates the proximity ratio or "small-worldness" tendency of the network
clustering_coefficient	Calculates the clustering coefficient
cond	Calculates the condition number of the network A
degree_distribution	Calculates the degree distribution
graphconncomp	Finds the strongly connected components of the graph
identifier	Returns the name of the network
median_path_length	Calculates the mean and median path length
time_constant	Calculates the smallest time constant of the system based on A
tol	Sets a tolerance value for computations if it is needed
type	Returns the type of the graph, i.e. 'directed' or 'undirected'

S.1.4.2.2 The Data class is included for analysis and quantification of data properties. The methods are listed in Table S.6. All measures are calculated and reported at an adjustable significance level, with default value $\alpha = 0.01$.

S.1.4.2.3 The CompareModels class can be used to compare networks to each other. A number of different measures are computed and reported, see Table S.7 - S.11. If the supplied golden standard network is not square, then `CompareModels` will calculate the similarity of off-diagonal elements, by assuming that the diagonal has been removed and truncated along the second dimension. Table S.12 is a reference for the methods that can be used with the CompareModels class. They add additional measures AUROC and AUPR (using "trapz" matlab function[16]) and methods to save, store and transfer results to other environments.

Table S.6 Methods of `Data` class.

Method	Description
alpha	Returns the significance level (default 0.01)
analyse_data	Batch calculation of almost all the measures below
calc_SNR_Phi_gauss	Calculates the SNR as defined in equation (S9)
calc_SNR_Phi_true	Calculates the SNR as defined in equation (S8)
calc_SNR_phi_gauss	Calculates the SNR as defined in equation (S10b)
calc_SNR_phi_true	Calculates the SNR as defined in equation (S6)
irrepresentability	Calculates the strong irrepresentable condition
tol	Sets a tolerance value for computations if it is needed

Table S.7 System measures

Name	Description
abs2norm	Absolute induced 2-norm
rel2norm	Relative induced 2-norm
maee	Max absolute element error
mree	Max relative element error
mase	Max absolute singular value error
mrse	Max relative singular value error
masde	Max absolute singular direction error
mrsde	Max relative singular direction error
maeve	Max absolute eigen value error
mreve	Max relative eigen value error
maede	Max absolute eigen direction error
mrede	Max relative eigen direction error
afronorm	Absolute Frobenius norm equivalent to 2-norm of A vectorised
rfronorm	Relative Frobenius norm
al1norm	l1-norm of zero elements
rl1norm	Relative l1-norm of zero elements
n0larger	# zero elements larger than smallest nonzero element of A
r0larger	# zero elements larger than smallest nonzero element of A/# zero elements in A

S.1.4.3 Methods toolbox

Provides mainly wrapper functionality for inference methods. These wrapper functions are written in relation to a specific inference method and accepts as input a standard set of variables: a `Dataset` object, an optional `Network` object, and a regularisation penalty value array. These are handled internally by the wrapper, which output an inferred network or array of inferred networks. The common interface for each function looks as follows:

```
zetavec = logspace(-6,0,10);
estA = Methods.method_name(Data,zetavec)
```

Here `Data` is the generated data object, and `zetavec` is a vector of regularisation parameters required by the method. The inferred networks are returned in `estA`. The available methods are listed in S.13. An alternative way of running the methods presented in this paper is also provided, where the regularization parameter range can be estimated by the method wrapper itself

```
[estA,zetavec,zetaRange] = Methods.method_name(Data,'full')
```

Calling the wrapper in this way will return a scaled zetavec $\in [0,1]$ where if possible all regularisation steps are included, and the scaling factors in zetaRange that can be used to rescale the zetavec to its actual range.

Table S.8 Signed topology measures

Name	Description
ncs	# Correct signs
sst	Similarity of signed topology
sst0	Similarity of signed topology of non-zero elements of A

Table S.9 Correlation measures

Name	Description
plc	Pearson's linear correlation coefficient

Table S.10 Graph measures

Name	Description
nlinks	# Links in estimated network
TP	# True Positives
TN	# True Negatives
FP	# False Positives
FN	# False Negatives
sen	Sensitivity TP/(TP+FN)
spe	Specificity TN/(TN+FP)
comspe	Complementary specificity 1-Specificity
pre	Precision TP/(TP+FP)
TPTN	Number of links that is present and absent in both networks (TP+TN)
structsim	Structural similarity (TP+TN)/#Nodesˆ2
MCC	Matthews correlation coefficient

Our in-house implementations of algorithms include Least Squares with Cut-Off (LSCO) and Total Least Squares with Cut-Off (TLSCO),

$$\hat{a}_{ij} \triangleq \begin{cases} a_{ij}^{*ls} & \text{if } a_{ij}^{*ls} \geq \tilde{\zeta} \\ 0 & \text{otherwise} \end{cases} \tag{S4}$$

where A_{*ls} is either the total or ordinary least squares estimate. We have also implemented a bootstrap approach for both the LSCO and the TLSCO algorithms.

An optimised implementation of the structurally constrained least squares (CLS) is also provided, which minimises the bias introduced by the regularisation term of *e.g.* LASSO, by solving

$$\hat{A} = \arg\min_{A} \sum \operatorname{diag}(\Delta^T R \Delta) \tag{S5a}$$

$$\text{s.t. } \Delta = AY + P, \tag{S5b}$$

$$R = \left(\hat{A}_{\text{init}} \operatorname{Cov}[y] \hat{A}_{\text{init}}^T + \operatorname{Cov}[p]\right)^{-1}, \tag{S5c}$$

$$\operatorname{sign} A = \operatorname{sign} \hat{A}_{\text{reg}}. \tag{S5d}$$

Here $\hat{A}_{\text{reg}}$ denotes the network estimate given by the regularisation method, *e.g.* LASSO, $\operatorname{Cov}[y]$ the covariance matrix of the response in an experiment or an estimate of it, $\operatorname{Cov}[p]$ the covariance matrix of the perturbation in an experiment or an estimate of it, and sign the signum function. The structure of the network is forced to be identical to the estimate given by the regularisation method by the last constraint in the optimisation problem. Ideally, the network estimate $\hat{A}$ should be used instead of $\hat{A}_{\text{init}}$, but then the problem is not convex. In practice one should therefore solve this problem iteratively, starting with $\hat{A}_{\text{init}} = \hat{A}_{\text{reg}}$ in the first iteration and then $\hat{A}_{\text{init}}$ equal

Table S.11 Directed graph measures

Name	Description
TR	True Regulation
TZ	True Zero
FI	False Interaction
FR	False Regulation
FZ	False Zero
dirsen	Directed sensitivity
dirspe	Directed specificity
dirprec	Directed precision
SMCC	Signed Matthews correlation coefficient

Table S.12 Methods in the CompareModels class

Name	Description
AUROC	Calculate area under ROC curve
ROC	Plot ROC curve
AUPR	Calculate area under PR curve
PR	Plot PR curve
save	save the comparison in specified format. Most formats that MATLAB datasets can be saved in are supported as well as json format if the jsonlab dependency is met

Table S.13 Method wrappers in GeneSPIDER. Each method can be called with a simple unified structure: `Methods.<function>(<data>,<parameters>)`.

function	reference	note
Glmnet	Friedman *et al.*[9]	LASSO/elastic net/ridge regression
NIR	Di Bernardo *et al.*[6]	Exhaustive subset regression
Bolasso	Bach[3]	Bootstrap utilising Glmnet
ccd	Abenius and U[1]	Cyclic coordinate descent
Glasso	Friedman *et al.*[8]	Graphical lasso
LARS	Sjöstrand[21]	Least angular regression
RNI	Nordling[18]	Robust network inference
julius	Julius *et al.*[13]	LASSO based convex programming
lsco	Tjärnberg *et al.*[23]	Least squares cut-off
fcls	Tjärnberg *et al.*[23]	Fast constrained least squares
ARACNe	**?**	Algorithm for the Reconstruction of Accurate Cellular Networks

to the estimate from the last iteration $\hat{A}$ until the estimate has converged with desired precision. This method is based on the method presented in[13], where the covariance is assumed to be identical in all experiments.

S.1.4.4 gsUtilities toolbox

Provides miscellaneous helper functions listed in S.14.

Table S.14 Contents of the `gsUtilities` toolbox.

Class or function	Description
export2gnuplot	Export of vectors and variables to a gnuplot friendly tsv format
optionParser	Parses input options into a struct
rmdiag	Removes diagonal elements and shifts the upper triangular elements -1 along the second dimension
sic	Calculates the strong irrepresentable condition
standardize	Standardises and normalises a given matrix

S.1.5 Definitions of data properties

We here define several data properties provided by GeneSPIDER.

S.1.5.1 Signal to Noise Ratio

The signal to noise ratio (SNR) can be defined in many different ways and we have implemented several of them in order to evaluate them. In general the SNR should be the ratio of the signal and uncertainty of the gene of interest[18]

$$\mathrm{SNR}(\phi_j) \triangleq \frac{\left\|\phi_j\right\|}{r_{\mathscr{U}_{\phi_j}}} \tag{S6}$$

where the radius of uncertainty set $\mathscr{U}_{\phi_j}$ is

$$r_{\mathscr{U}_{\phi_j}} \triangleq \sup_{\tilde{\phi}_j \in \mathscr{U}^{\alpha}_{\phi_j}} \left\|\tilde{\phi}_j - \phi_j\right\| \tag{S7}$$

From (S2) we can derive an expression of the SNR considering only output noise

$$\mathrm{SNR}_{\Phi\, true} \triangleq \frac{\underline{\sigma}(Y)}{\overline{\sigma}(E)} = \frac{\underline{\sigma}(\Phi)}{\overline{\sigma}(\Upsilon)}. \tag{S8}$$

Here $\overline{\sigma}$ represent the largest singular value and $\underline{\sigma}$ represent the smallest non-zero singular value. Note that the noise matrix E is only available for *in silico* data. We therefore also define a corresponding measure based on assumption of the noise being normally distributed with variance λ

$$\mathrm{SNR}_{\Phi\, \mathcal{N}(\mu,\lambda)} \triangleq \frac{\underline{\sigma}(\Phi)}{\sqrt{\chi^{-2}(\alpha, NM)\lambda}}. \tag{S9}$$

Here $\mathcal{N}(\mu,\lambda)$ indicates that the noise is assumed to follow a normal distribution with mean μ, variance $\lambda_{(i)}$ and $\chi^{-2}(\alpha, NM)$ is the inverse chi-square distribution with NM degrees of freedom at significance level α. We also define SNRs for individual genes:

$$\mathrm{SNR}_{\phi\, \mathcal{N}(\mu,\lambda)} \triangleq \arg\min_i \frac{\|\phi_i\|}{\sqrt{\chi^{-2}(\alpha, N)\lambda_i}} \tag{S10a}$$

$$\mathrm{SNR}_{\phi\, true} \triangleq \arg\min_i \frac{\|\phi_i\|}{\|\upsilon_i\|} \tag{S10b}$$

$$\langle \mathrm{SNR} \rangle_{\phi\, \mathcal{N}(\mu,\lambda)} \triangleq \mathrm{mean}_i \frac{\|\phi_i\|}{\sqrt{\chi^{-2}(\alpha, N)\lambda_i}} \tag{S10c}$$

$$\langle \mathrm{SNR} \rangle_{\phi\, true} \triangleq \mathrm{mean}_i \frac{\|\phi_i\|}{\|\upsilon_i\|}. \tag{S10d}$$

S.1.5.2 Sample-wise linear dependence

An experiment can only be predicted through a model based upon data reflective of the underlying system properties, *e.g.* a closely related experiment. This implies that it is essential to filter the experiments before using any leave-one-out cross-validation or cross-optimisation strategy[23]. If an experiment is linearly dependent on other experiments then the latter contain information about the former, and thus one should estimate the linear independence of the samples, η_{y_k} and η_{p_k},

$$\eta_{y_k} \triangleq ||Y_{t\neq k}^T y_k||_1 \qquad \eta_{p_k} \triangleq ||P_{t\neq k}^T p_k||_1. \tag{S11}$$

Only samples fulfilling

$$\mathscr{V} \triangleq \{k | \eta_{y_k} \geq \sigma_N(Y) \text{ and } \eta_{p_k} \geq \sigma_N(P)\} \tag{S12}$$

should be included.

S.1.6 Generating example data as used in results section

GeneSPIDER provides four MATLAB toolboxes: `datastruct`, `analyse`, `Methods`, and `gsUtilities`. Each toolbox is aimed at a specific function and their usage is exemplified here. The data used in the examples below can be downloaded from the online repository at `https://bitbucket.org/sonnhammergrni/gs-networks`. The network is Nordling-D20100302-random-N10-L25-ID1446937 and dataset is Nordling-ID1446937-D20150825-E15-SNR3291-IDY15968. Note that if the code below is used to generate a new network and dataset, then they will differ from the presented ones due to the use of random number generators to create the network and noise matrices.

S.1.6.1 Network generation

We start by generating a stable random network with 10 nodes and sparsity 0.25. The following code snippet demonstrate how to create a `datastruct.Network` object with the above specifications.

```
N = 10; S=0.25;
A = datastruct.randomNet(N,S)-eye(N);
A = datastruct.stabalize(A,'iaa','high');
Net = datastruct.Network(A,'random');
setname(Net,struct('creator','Nordling'));
Net.description = ['This is a sparse network with 10 nodes,'...
    '10 negative self-loops and 15 randomly chosen'...
    'links generated by Nordling 2010-03-02.'...
    'The coefficients are chosen such that they form one'...
    'strong component and a stable dynamical system with'...
    'time constants in the range 0.089 to 12 and an'...
    'interampatteness level of 145 that is in-between'...
    'the estimated level of an E. coli (Gardner et al. 2003 Science)'...
    'and Yeast (Lorenz et al. 2009 PNAS) gene regulatory network.'...
    'The coefficients of the network have not been tuned to explain'...
    'any of the data sets in the mentioned articles.'];
```

`datastruct.stabalize` takes the random network and the desired IAA as input parameters and stabilises the network by making the real part of all eigenvalues negative while adjusting the IAA level. The `setname` method is used to specify the fields of the `Network` object. The name is automatically generated based on the network properties to ensure that each one is unique.

The displayed output of the `Network` object is in this case:

```
Net =

  10x10 Network array with properties:

              network: 'Nordling-D20100302-random-N10-L25-ID1446937'
                    A: [10x10 double]
                    G: [10x10 double]
                names: {'G1' 'G2' 'G3' 'G4' 'G5' 'G6' 'G7' 'G8' 'G9' 'G10'}
                 desc: 'This is a sparse network with 10 nodes, 10 negative
                        self-loop and 15 randomly chosen links generated by
                        Nordling 2010-03-02. The coefficients are chosen such
                        that they forms one strong component and a stable
                        dynamical system with time constants in the range 0.089
                        to 12 and an interampatteness level of 145 that is in
                        between the estimated level of an
                        E. coli (Gardner et al. 2003 Science) and
                        Yeast (Lorenz et al. 2009 PNAS) gene regulatory network.
                        The coefficients of the network have not been tuned to
                        explain any of the data sets in the mentioned articles.'
```

The displayed output shows the non-hidden properties of the `Network` object. `network` is the name of the object, which contains the name of the creator `Nordling`, the date of creation `D`, the type of network `random`, the number of nodes, and the number of edges `L`. `A` is the network matrix. `G` is the static gain matrix (inverse of `A`), which is precomputed to save time when used in an inference algorithm. `names` contains the name assigned to each node, which are generated automatically if they are not specified. `desc` is a description of the network. The Network class can handle sparse matrices.

For the example in this article, we generated 10 networks of each size, $N \in \{10, 50, 100\}$, each of the four classes, random, small-world, scale-free and small-world-scale-free, and each of two IAA levels, $\kappa \in \{low, high\}$, giving a total of 240 networks. The IAA degree of each network is shown in Figure S.1. The degree distributions for each network topology class and size are shown in Figure S.2.

S.1.6.2 Data generation

We now use the generated network to simulate perturbation experiments to obtain an expression dataset. The following code snippet simulates N single gene perturbation experiments where each gene is perturbed one by one followed by $N/2$ experiments in which genes are perturbed randomly.

```
SNR = 7;
P = double([eye(N),full(logical(sprandn(N,round(N/2),0.2)))]);
Y = Net.G*P;
s = svd(Y);
stdE = s(N)/(SNR*sqrt(chi2inv(1-analyse.Data.alpha,prod(size(P)))));
E = stdE*randn(size(P));
F = zeros(size(P));
```

We have created a perturbation matrix `P` and a corresponding response matrix `Y`. The standard deviation has been selected such that the SNR became 7 when it was used to generate the noise matrix `E`. We didn't use the input noise matrix `F` here, but it needs to be specified, so it was set to zero. With this information, we build a data struct, which we later use to populate the `Dataset` object.

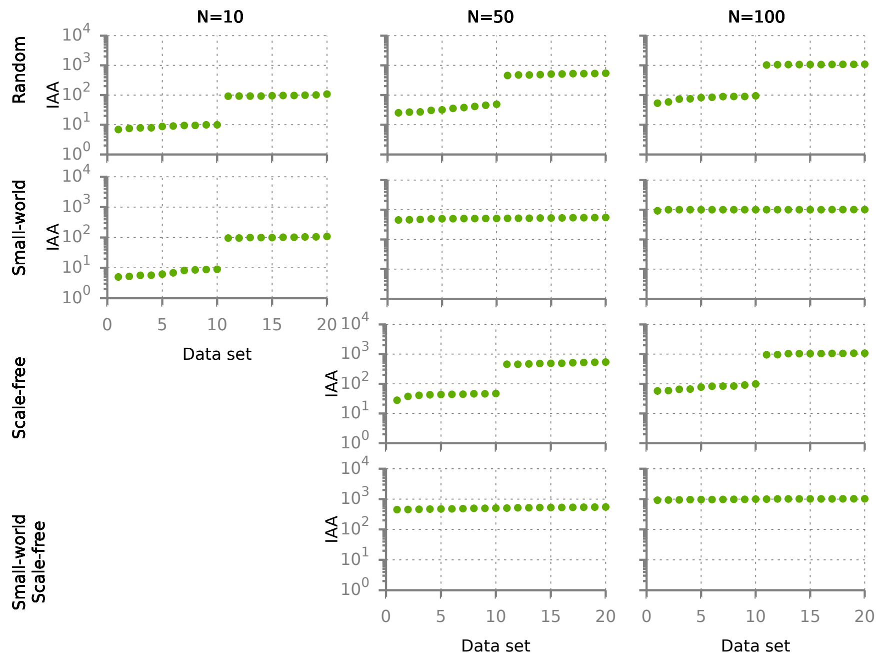


Fig. S.1 Interampatteness degrees for the networks in the benchmark suite. Each point represents a network, with its IAA degree on the y-axis and its number in order of increasing IAA on the x-axis. The numbering is the same as in Figure 2.

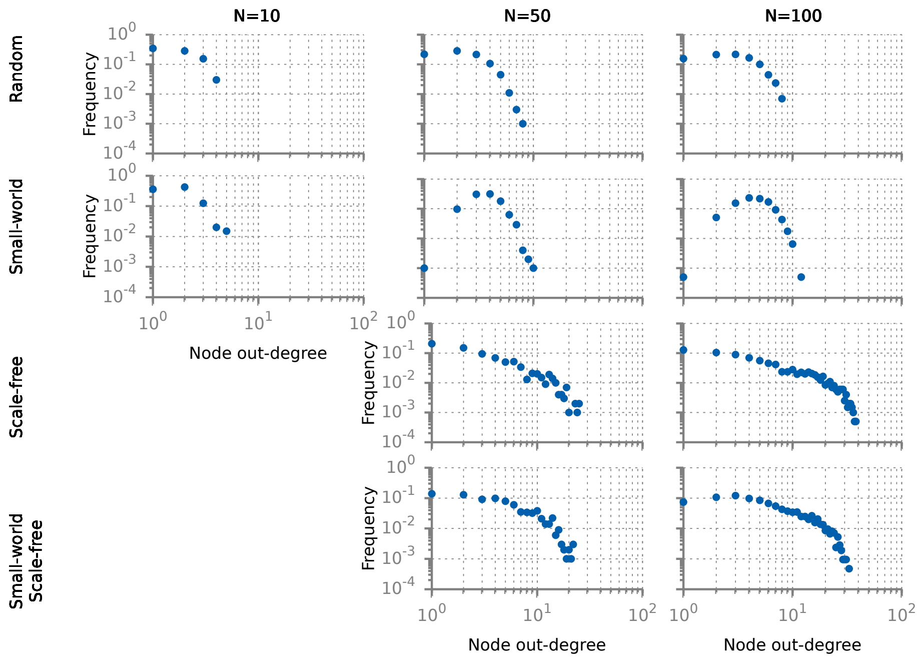


Fig. S.2 Out-degree distributions for the networks in the benchmark suite. Each distribution is based on 20 individual networks. The y-axis shows the degree frequency, and the x-axis shows the degree.

```
D(1).network = Net.network;
D(1).E = E;
D(1).F = F;
D(1).Y = Y+D.E;
D(1).P = P;
D(1).lambda = [stdE^2,0];
D(1).cvY = D.lambda(1)*eye(N);
D(1).cvP = zeros(N);
D(1).sdY = stdE*ones(size(D.P));
D(1).sdP = zeros(size(D.P));
```

The two easiest ways to populate the `Dataset` object with generated data is to either initialise it with the data and/or network or to use the function `populate`. To initialise the `datastruct.Dataset` object with data we do the following:

```
Data = datastruct.Dataset(D,Net);
setname(Data,struct('creator','Nordling'));
data.description = ['This data set contains 15 simulated experiments with additive'...
    'white Gaussian noise with variance 0.00028 added to the response'...
    'in order to make the SNR 7 and the data partly informative for'...
    'network inference. The singular values of the response matrix'...
    'are in the range 0.77 to 1.2.'];
```

The displayed output of the `Dataset` object is in this case:

```
Data =

Dataset with properties:

      dataset: 'Nordling-ID1446937-D20150825-E15-SNR3291-IDY15968'
      network: 'Nordling-D20100302-random-N10-L25-ID1446937'
            P: [10x15 double]
            F: [10x15 double]
          cvP: [10x10 double]
          sdP: [10x15 double]
            Y: [10x15 double]
            E: [10x15 double]
          cvY: [10x10 double]
          sdY: [10x15 double]
       lambda: [0.00028399 0]
        SNR_L: 3.2912
        names: {'G01'  'G02'  'G03'  'G04'  'G05'  'G06'  'G07'  'G08'  'G09'  'G10'}
  description: 'This data set contains 15 simulated experiments with additive
               white Gaussian noise with variance 0.00028 added to the response
               in order to make the SNR 7 and the data partly informative for
               network inference. The singular values of the response matrix
               are in the range 0.77 to 1.2.'
```

It is important to be able to connect a dataset to a specific network if the data was generated *in silico*, hence the network name is reported in the `Data` object.

S.1.6.3 Analysis

The `analysis` toolbox provides tools to analyse data, networks, and benchmark results.

First we demonstrate how to load the correct network and dataset from the online repository:

```
v = version('-release');
if str2num(v(1:end-1)) >= 2015
    disp('Fetching example data online')
    Net = datastruct.Network.fetch('Nordling-D20100302-random-N10-L25-ID1446937.json')
    Data = datastruct.Dataset.fetch('Nordling-ID1446937-D20150825-E15-SNR3291-IDY15968.json')
```

```
else
    disp('Older versions of MATLAB does not support fetching datasets online.')
end
```

S.1.6.3.1 Network analysis: To analyse the network we input it to the `analyse.Model` module:

```
net_prop = analyse.Model(Net);
disp(net_prop)
```

It produces the output:

```
net_prop =

  Model with properties:

             network: 'Nordling-D20100302-random-N10-L25-ID1446937'
     interampatteness: 144.6937
    NetworkComponents: 1
        AvgPathLength: 2.8778
                 tauG: 0.085032
                   CC: 0.1
                   DD: 1.5
```

Six measures are calculated. The interampatteness degree, `interampatteness`, is the number reported by `cond(A)` in MATLAB. `NetworkComponents` is the number of strongly connected components, as reported by the MATLAB function `graphconncomp`. `AvgPathLength` is the average path length of the graph of the network in question, as reported by `graphallshortestpaths` in MATLAB. `tauG` is the time constant of the system. `CC` is the average Clustering coefficient, which can be interpreted as the neighbourhood sparsity of each node in the network, not considering the node itself. `DD` is the average degree distribution of the model. The property `analyse.Model.type` can be set to `directed` (default) or `undirected` depending on the network and the properties one wishes to calculate. This is a persistent property, so the value will remain the default one until it is changed.

Individual properties can also be calculated, *e.g.* all clustering coefficients can be calculated by

```
disp(['Clustering coefficients of the network ',Net.network])
CCs = analyse.Model.clustering_coefficient(Net)
```

S.1.6.3.2 Data analysis: To analyse the data we input the `Dataset` object to the `analyse.Data` module:

```
data_prop = analyse.Data(Data);
disp(data_prop)
```

It will result in the following output:

```
data_prop =

Data with properties:

        dataset: 'Nordling-ID1446937-D20150825-E15-SNR3291-IDY15968'
   SNR_Phi_true: 7
  SNR_Phi_gauss: 3.2912
   SNR_phi_true: 10.991
  SNR_phi_gauss: 10.341
```

The SNRs reported here correspond the definitions in equations (S9) - (S10b) by default. However, the SNR is calculated for all *i* with the following two functions:

```
disp('SNR estimate based on actual noise matrix E for each variable')
SNRe = analyse.Data.calc_SNR_phi_true(Data);
disp(SNRe)

disp('SNR estimate based on variance estimate each variable')
SNRl = analyse.Data.calc_SNR_phi_gauss(Data);
disp(SNRl)
```

S.1.6.3.3 Performance evaluation: To analyse the performance of an inference method we first need to generate an output. This is accomplished easily thanks to the wrappers. Each method has an associated wrapper that parses the data of the method itself. To run the Glmnet LASSO implementation we execute:

```
[estA,zetavec,zetaRange] = Methods.Glmnet(Data,'full');
```

The variable `zetavec` is the returned regularisation parameters that was used within the algorithm. The option "full" will instruct the method to try to generate the complete regularization path from full to empty network with the ζ values scaled between 0 and 1. It should be noted that not all methods can reliably do this. For those cases a zetavec can be specified and supplied to the method. `zetaRange` gives the scaling factors used for the parameters.

```
zetavec = logspace(-6,0,100)
estA = Methods.Glmnet(Data,zetavec);
```

and the method will use that vector of values to infere the networks.

To analyse the performance of the model, we input the network estimates produced by the algorithm to the model comparison method:

```
M = analyse.CompareModels(Net,estA);
```

The `max` operation can now be used to find the optimal performance for each calculated measure:

```
maxM = max(M);
```

Note that `maxM` will contain the maximum of all measures calculated in `analyse.CompareModels`. If one wants to get all measures when a specific measure is maximised, one should specify that as an input.

```
max_MCC_M = max(M,'MCC');
```

This will return all applicable measures to that point.

The measures currently available are detailed in tables S.7 - S.11. `CompareModels` will calculate similarity of non-diagonal elements if the input gold standard model is not square, assuming that the diagonal has been removed and truncated along the second dimension.

S.1.6.4 Benchmark Results (continued)

The disparity between AUROC and MCC performance metrics in Fig. 4 is broken out for comparison into its individual components in the online supplemental section, where one can see how they vary with varying sparsity. In the AUROC plots, various points have their density (red) and MCC (black) displayed to allow for verification of the overall performance bar chart of the results section.

References

1 Abenius, T. and U, A. X. (2010). Summary of methods. *Control*, pages 18–21.

2 Almeida, J. S., Wu, S., and Voit, E. O. (2003). Xml4mat: Inter-conversion between matlab tm structured variables and the markup language mbml.

3 Bach, F. R. (2008). Bolasso: Model consistent lasso estimation through the bootstrap. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 33–40, New York, NY, USA. ACM.

4 Banga, J. R. and Balsa-Canto, E. (2008). Parameter estimation and optimal experimental design. *Essays In Biochemistry*, **45**, 195–210.

5 Bonneau, R., Reiss, D. J., Shannon, P., Facciotti, M., Hood, L., Baliga, N. S., and Thorsson, V. (2006). The inferelator: an algorithm for learning parsimonious regulatory networks from systems-biology data sets de novo. *Genome biology*, **7**(5), R36.

6 Di Bernardo, D., Gardner, T. S., and Collins, J. J. (2004). Robust identification of large genetic networks. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, **497**, 486–497.

7 Franceschini, G. and Macchietto, S. (2008). Model-based design of experiments for parameter precision: State of the art. *CHEMICAL ENGINEERING SCIENCE*, **63**, 4846–4872.

8 Friedman, J., Hastie, T., and Tibshirani, R. (2008). Sparse inverse covariance estimation with the graphical lasso. *Biostatistics (Oxford, England)*, **9**(3), 432–441.

9 Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, **33**(1), 1–22.

10 Gardner, T. S., Bernardo, D., Lorenz, D., and Collins, J. J. (2003). Inferring genetic networks and identifying compound mode of action via expression profiling. *Science*, **301**(7), 102–105.

11 Gustafsson, M. and Hörnquist, M. (2010). Gene expression prediction by soft integration and the elastic net-best performance of the dream3 gene expression challenge. *PloS one*, **5**(2), e9134.

12 Gustafsson, M., H?rnquist, M., and Lombardi, A. (2005). Constructing and analyzing a large-scale gene-to-gene regulatory network-lasso-constrained inference and biological validation. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **2**(3), 254–261.

13 Julius, a., Zavlanos, M., Boyd, S., and Pappas, G. J. (2009). Genetic network identification using convex programming. *IET systems biology*, **3**(3), 155–166.

14 Kreutz, C. and Timmer, J. (2009). Systems biology: experimental design. *FEBS Journal*, **276**(4), 923–942.

15 Marbach, D., Costello, J. C., Küffner, R., Vega, N. M., Prill, R. J., Camacho, D. M., Allison, K. R., Kellis, M., Collins, J. J., and Stolovitzky, G. (2012). Wisdom of crowds for robust gene network inference. *Nature Methods*, **9**(8), 796–804.

16 MATLAB (2014). 8.3.0.532 (r2014a).

17 Michalik, C., Stuckert, M., and Marquardt, W. (2010). Optimal experimental design for discriminating numerous model candidates: The awdc criterion. *Industrial & Engineering Chemistry Research*, **49**(2), 913–919.

18 Nordling, T. E. M. (2013). *Robust inference of gene regulatory networks*. Ph.D. thesis, KTH School of Electrical Engineering, Automatic Control Lab.

19 Nordling, T. E. M. and Jacobsen, E. W. (2009). Interampatteness - a generic property of biochemical networks. *IET systems biology*, **3**(5), 388–403.

20 Schwaab, M., Luiz Monteiro, J., and Carlos Pinto, J. (2008). Sequential experimental design for model discrimination. *Chemical Engineering Science*, **63**(9), 2408–2419.

21 Sjöstrand, K. (2005). Matlab implementation of LASSO, LARS, the elastic net and SPCA. Version 2.0.

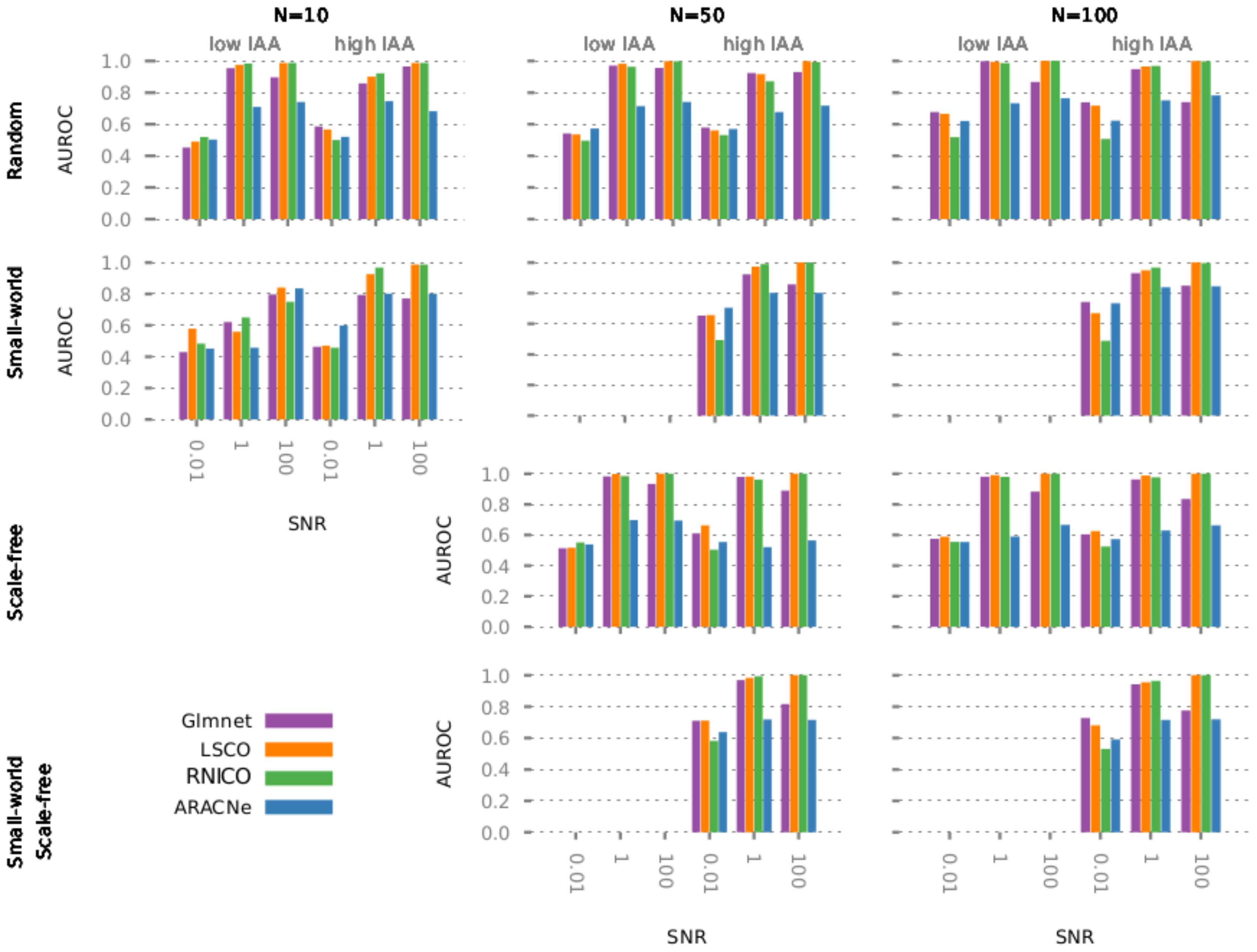
22 Tegner, J., Yeung, M. K. S., Hasty, J., and Collins, J. J. (2003). Reverse engineering gene networks: Integrating genetic perturbations with dynamical modeling. *Proceedings of the National Academy of Sciences*, **100**(10), 5944–5949.

23 Tjärnberg, A., Nordling, T. E. M., Studham, M., and Sonnhammer, E. L. L. (2013). Optimal sparsity criteria for network inference. *Journal of Computational Biology*, **20**(5), 398–408.

24 Tjärnberg, A., Nordling, T., Studham, M., Nelander, S., and Sonnhammer, E. (2015). Avoiding pitfalls in l1-regularised inference of gene networks. *Mol. BioSyst.*, pages 287–296.

25 Zhao, P. and Yu, B. (2006). On model selection consistency of lasso. *The Journal of Machine Learning Research*, **7**, 2541–2563.

Figure S.3



N=10
N=50
N=100
low IAA
high IAA
Random
Small-world
Scale-free
Small-world Scale-free
AUROC
1.0
0.8
0.6
0.4
0.2
0.0
0.01
1
100
SNR
Glmnet
LSCO
RNICO
ARACNe